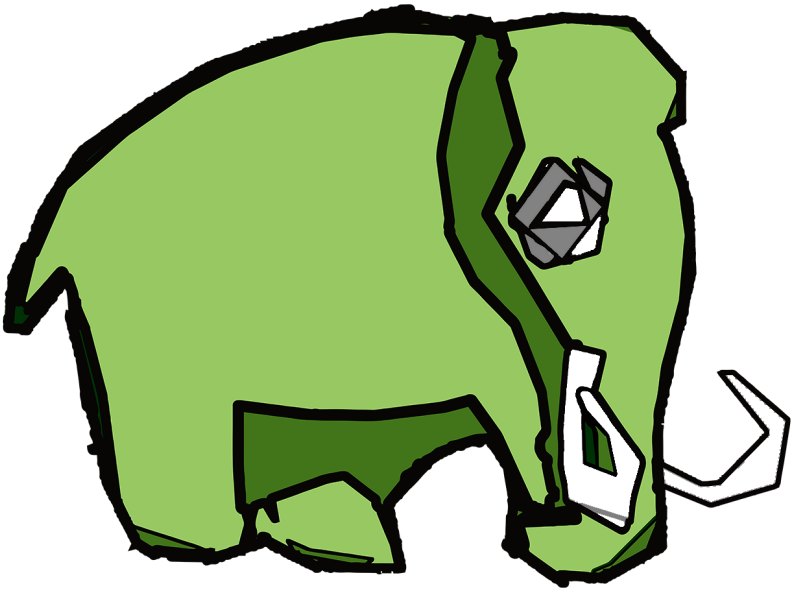
Mastodon
documentation



Tobias Pietzsch & Jean-Yves Tinevez

October 11, 2020

Institut Pasteur

CBG
Max Planck Institute
of Molecular Cell Biology
and Genetics

Preamble.

This document constitutes the Mastodon documentation. It is divided in four parts, that group sections by interest.

- The first part contains three tutorials, aimed at end-users and focused on cell tracking. They are meant to guide users with the Mastodon software and cover three applications of cell tracking Mastodon:
  - automated cell or particle tracking;
  - manual curation and correction of tracking results;
  - manual and semi-automatic tracking.
- The second part contains technical information. ...
- The third part is aimed at advanced users, ...
- The last part is made of tutorials aimed at Java developers ...

Contents

Part I.
Using Mastodon.

1. Getting started with Mastodon. Automated tracking.

This tutorial is the starting point for new Mastodon users. It will walk you through basic operations in Mastodon, opening a dataset and creating a Mastodon project, automatically detect cells and link them, and show you how to use the main views of Mastodon. We don't go into details, and will revisit the features we survey here later.

1.1. The image data.

1.1.1. Exporting your image to BigDataViewer file format.

Mastodon uses BigDataViewer (BDV) files as input images. You need to prepare your images so that they can be opened in the BigDataViewer .

BDV files are used more and more by several software projects in the Fiji ecosystem and beyond. This tutorial focuses on Mastodon not on BDV, however we will take a very small detour to explain what makes it fit and how to turn your images into this format. If you know already, you can skip this part, because we simply recapitulate what is being explained in the original BigDataViewer publication [1].

For this tutorial we will use a ready-made dataset, in the adequate format, but it is a good idea to know how to export or create an image in such a format. We lazily rely on the excellent BigDataViewer documentation and point directly to the BigDataViewer instructions to prepare your images, *e.g.* depending on whether

- they are opened as an ImageJ stack, or
- they come from a SPIM processing pipeline, or
- they come from the BigStitcher plugin [2], *etc* .

Once you have prepared your images for opening in the BigDataViewer , you should have a `.xml` file and a possibly very large `.h5` file on your computer. The `.xml` file must be the output of the BigDataViewer data preparation. It should start with the following lines:

```
<?xml version="1.0" encoding="UTF-8"?>
<SpimData version="0.2">
  <BasePath type="relative">.</BasePath>
  <SequenceDescription>
    <ImageLoader format="bdv.hdf5">
      <hdf5 type="relative">datasethdf5.h5</hdf5>
...
```

1.1.2. Key advantages of the BigDataViewer file format.

The BDV file format solves mainly two challenges in image visualization and analysis, that arise with modern microscopy, namely:

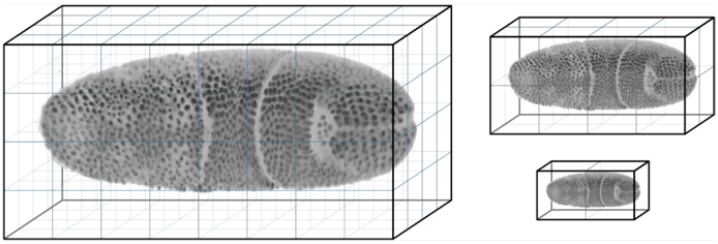


Figure 1.: Illustration of the BDV file format storage strategy. The image is stored over several resolution levels (multi-scale pyramid) and in chunks.

- Modern microscopes can generate images that are very large in size. Much larger that what can be fitted in RAM, even with the increase in computer power. It is now common to find single movies acquired on SPIM microscopes that are several TBs in size. Computers with several TBs of RAM are not so common.
- Multiple views of the same sample can be acquired, and they need to be visualized in the same viewer. The first use case is also the multi-view images generated by SPIM microscopes, but we can also think of correlative light-electron microscopy.

If we focus on the first challenge, you see that we need to stream the image data directly from the disk, instead of fully loading it into RAM. But at the same time, we need a tool that allows for interactive browsing of the data. The view must be responsive to the user input, and not block when it has to load the data from the file. The BDV file format offers a clever file format design that does this, coupled to a specialized viewer. The image data are stored in small chunks corresponding to a neighborhood. As the viewer shows a slice through the image, the required chunks are loaded on demand and cached. All the chunks are organized in a HDF5 file, which is like a file-system in a file[1], and accessing single chunks is fast with current computer hardware. On top of this, the image is also stored as a multi-scale pyramid[2], to speed-up zooming and unzooming (Figure 1). The BDV display component exploits this file format in a clever way, and ensures that the view still answers to user interactions (mouse pan, zoom, clicks *etc*) even if the chunks are not full loaded.

There are several implementations of this strategy, for instance in Imaris[3] and with the new file format N5[4] proposed by the Saalfeld lab. Some of them are inter-compatible. We will pick the BDV file format all along this document. The BigDataViewer has proved its value and impact on our field. For instance our previous work on cell lineaging in large images, MaMuT, is based on BDV [3].

1.1.3. The tutorial dataset.

Mastodon was created specially because we needed to harness very big, multi-view images. We wanted to generate comprehensive lineages and follow a large number of cells over a very long time. This accumulation of inflated words is tied to the very large - in objective disk space occupation - images we deal with using modern microscopy tools. Such datasets might not be optimal for a first

[1] Hierarchical Data Format on Wikipedia.
[2] Multi-scale pyramid on Wikipedia.
[3] IMARIS 5.5 File Format Description (IMS).
[4] N5 API on GitHub.

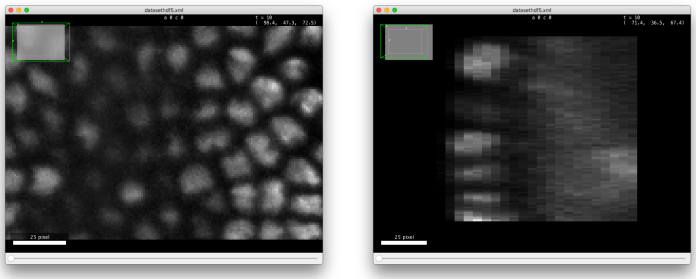


Figure 2.: The tutorial dataset opened in BigDataViewer , seen along XY (left) and XZ (right)

contact with Mastodon. So just for this tutorial we will use a smaller dataset. It is a small region cut into a movie following the development of a drosophila embryo, acquired by William Lemon in Patrick Keller lab (HHMI, Janelia Farm). This was created from the example dataset released with the TGMM software [4]. You can find it on Zenodo[5] there: DOI 10.5281/zenodo.3336346

It is a zip file that contains 3 files:

```
14M datasethdf5.h5
.7K datasethdf5.settings.xml
.7K datasethdf5.xml
```

The `.h5` file is the HDF5 file mentioned above, that contains the image data itself. The `datasethdf5.xml` is a text file following the XML convention, specific to the BDV file format, that contains information about the the image data and metadata. When we want to open a BDV file, we point the reader to this file. The `datasethdf5.settings.xml` is an optional file that stores user display parameters, such as channel colors, min and max display value, as well as bookmarks in the data. We refer you to the BDV documentation about this file. Mastodon uses this settings file to store that same information.

If you open this data in the BigDataViewer (in Fiji in the Plugins > BigDataViewer > Open XML/HDF5 menu), you should see something like in figure 2. There is about 70 cells in each of the 30 time-points, arranged in a layer at the top of the sample. The deeper part of the sample (low Z coordinates) has some hazy, diffuse signal from which we cannot individualize cells. As time progresses, the cells move towards the middle part and bottom (high Y coordinates) part of the image, and some of them move deeper in Z, initiating gastrulation.

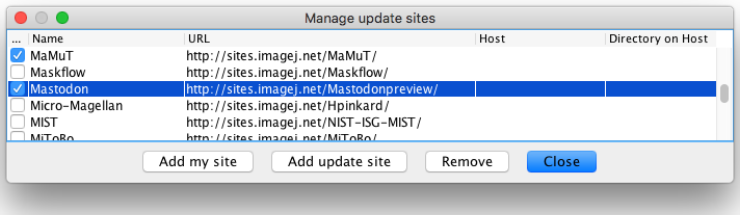
The goal of this short tutorial is to track all these cells in Mastodon.

1.2. Getting Mastodon.

As of today, Mastodon is available as a preview. We are still working on adding and validating features. Nonetheless the preview has everything we need to track these cells. Also, Mastodon is independent of ImageJ or Fiji, it can operate as a standalone software. However we currently distribute it via Fiji, because the updater and the dependency management are so convenient. So the first thing to do is to grab Fiji[6], if you do not have it already.

[5] https://zenodo.org/record/3336346
[6] http://fiji.sc/

Then launch the Fiji updater and once your Fiji is up to date, click on the `Manage update site` button. We will add the add the Mastodon update site. You should find the Mastodon preview site in the list. Select it, update Fiji and restart it. After restarting, you should find the command Plugins > Mastodon (preview) at the bottom of the menu.



1.3. Creating a new Mastodon project.

After launching the command, this plain, sober window appears.

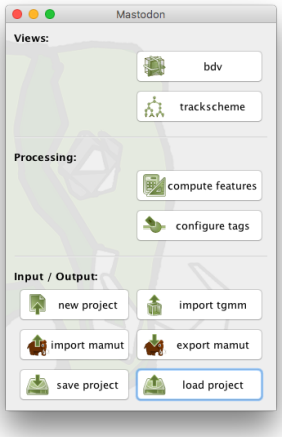
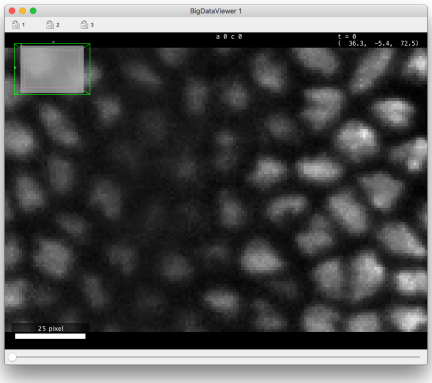


Figure 3.: The main window of Mastodon.

Click on new project, and browse to the `datasethdf5.xml` file of the XML/HDF5 file pair of the tutorial dataset. All the buttons that were grayed out should be now enabled. Click on the bdv button. A BDV window should appear and if it does everything is right.



It is almost a regular BDV window and if you already know who to use it and the key bindings you should find your marks quickly. The BDV view displays a *slice* of the image through arbitrary orientation. Below we give the commands and key-bindings for navigation in a Mastodon-BDV window. They are indeed close to what is found in the standard BigDataViewer but some changes. **Please note:** You can reconfigure almost everything in Mastodon, as we will see later, including key-bindings. In this tutorial and the next ones, the key-bindings we present are for the `Default` configuration. In the table 1 page 12 you will find the key bindings to navigate through the image data.

Now you want to save the project. Go back to the main window, and click on the save project button. This will create a single file, called for instance `drosophila_crop.mastodon` file. This file is actually a zip file that contains the tracks and *links* to the image data. The image data is kept separate from the Mastodon file, which allows for using it with another software, independently. So if you want to transfer or move a full Mastodon project, you need to take the `.mastodon` file and all the `.xml` and `.h5` files from the BigDataViewer dataset.

Next time you want to open this project, just click on the load project button and point the file browser to the `.mastodon` file. The image data will be loaded along with the lineages.

1.4. Detecting cells.

We want to track automatically all the cells in this dataset, and the first step is therefore to detect them. Mastodon ships a wizard to perform cell detection. It is very much inspired by the TrackMate GUI, and if you know this software you will find your marks here. Also, the algorithms are very close to what was in TrackMate [5], but they have been heavily optimized for Mastodon.

The detection wizard can be launched from the Plugins > Tracking > Detection... menu item. You should have a window like the one depicted in Figure 4.

Like for TrackMate, the automated tracking user interface uses *wizards* to enter parameters, select algorithms, *etc.* You can navigate back and forth with the Next and Previous buttons. The Log button will bring an independent panel where all the activity in the wizard are logged as text.

This first panel allows for selecting the target *source* on which the detection will be run. Since we use the BigDataViewer for images, a channel or a view is stored and displayed as a source. A source can have multiple resolutions stored, as explain in paragraph 1.1.2, but for the data used in this

Table 1.: Default navigation key-bindings for Mastodon-BDV views.

Action	Key
	View.
Move in X & Y.	Right-click and Drag
Move in Z.	Mouse-wheel. Press and hold ⇧ to move faster, ctrl to move slower.
Align view with X / Y / Z axes.	• Align with XY plane: ⇧ + Z • Align with YZ plane: ⇧ + X • Align with XZ plane: ⇧ + C or ⇧ + Y The view will rotate around the location you clicked.
Zoom / Unzoom.	ctrl + ⇧ + Mouse-wheel or ⌘ + Mouse-wheel. The view will zoom and unzoom around the mouse location.
	Time-points.
Next time-point.	] or M
Previous time-point.	[or N
	Bookmarks.
Store a bookmark.	⇧ + B then press any key to store a bookmark with this key as name. A bookmark stores the position, zoom and orientation in the view but not the time-point. Bookmarks are saved in display settings file.
Recall a bookmark.	Press B then the key of the bookmark.
Recall a bookmark orientation.	Press O then the key of the bookmark. Only the orientation of the bookmark will be restored.
	Image display.
Select source 1, 2 …	Press 1 / 2 …
Brightness and color dialog.	Press S. In this dialog you can adjust the min & max for each source, select to what sources these min & max apply and pick a color for each source.
Toggle fused mode.	Press F. In fused mode, several sources are overlaid. Press ⇧ + 1 / ⇧ + 2 …to add / remove the source to the view. In single-source mode, only one source is shown.
Visibility and grouping dialog.	Press F6. In this dialog you can define what sources are visible in fused mode, and define groups of sources for use in the grouping mode.
Save / load display settings.	F11 / F12. This will create a `XYZ_settings.xml` file in which the display settings will be saved.

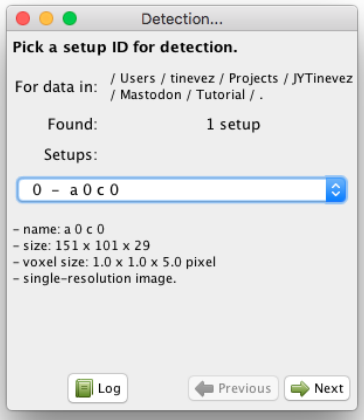
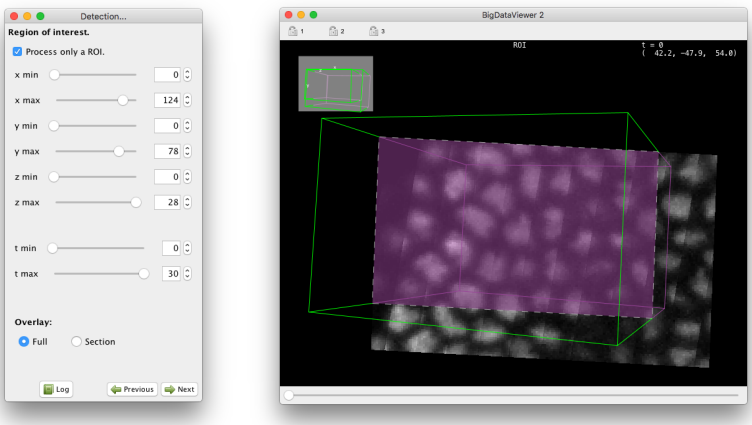


Figure 4.: First panel of the detection wizard.

tutorial this is not the case. The sources are nicknamed 'setups' in this panel. They are numbered from 0 and can be selected from the drop-down list. Below the list we try to display the metadata we could retrieve from the BigDataViewer file. Just pick the first and only channel, and click Next.

You can now choose to operate only on a rectangular ROI in the image. If you check the **Process only a ROI** button, new controls appear in the panel, and a ROI is drawn into an open BDV view (a new one is created if one is not opened). The ROI is painted as a wire-frame box, green for vertices that point towards the camera from the displayed slice, and purple for vertices that points away from the camera, below the displayed slice. The intersection of the ROI box with the displayed slice is painted with a purple semi-transparent overlay, with a white dotted line as borders. You can control the ROI bounds with the controls in the panel, or by directly dragging the ROI corners in the BDV view. Time bounds can also be set this way. In our case we want to segment the full image over all time-points, so leave the **Process only a ROI** button unchecked.

You cannot have non-rectangular ROIs in Mastodon. Nonetheless they are super useful as is. You can for instance combine several detection steps using different parameters in different region of your image. Or different time interval.



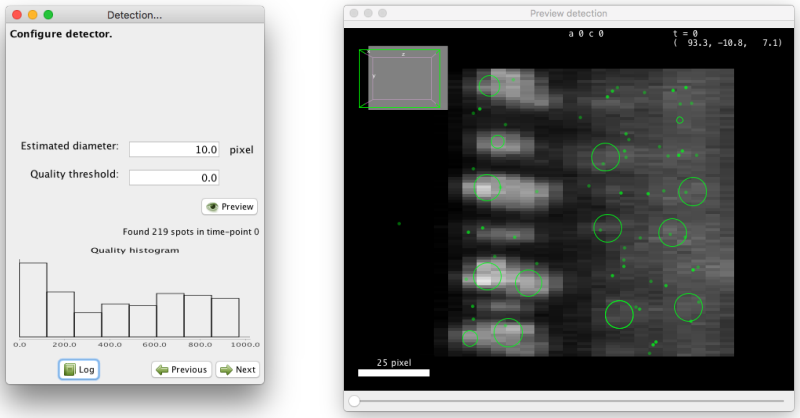


Figure 5.: Previewing detection results.

The next panel lets you choose the detector you want to use. In vanilla Mastodon, three detectors are available. Right now, we will use the default one, the **DoG detector**, which should be good enough for most cases. DoG means 'difference-of-Gaussians'. It is an efficient approximation of the LoG ('Laplacian of Gaussian') filter, and there is also a detector in Mastodon based on the latter.

These detectors excel at finding roundish structures in the image that are bright over a dark background. The structures must have a shape somewhat close to a sphere, but they can accommodate a lot of variability. As a rule of thumb, if you can define rough estimate of the radius of these structures, they are eligible to be picked up by our detectors. This also implies that in Mastodon, we cannot segment complex shapes, or object labeled by their contour (*e.g.* cell membranes), or even exploit these shapes to have an accurate measurements of the volume. This is an important limitation of Mastodon.

For now, select the **DoG detector** and click Next.

Here is briefly how it works. The LoG detector, and its approximation the DoG detector, is the best detector for blob-like particles in the presence of noise [6]. It is based on applying a Laplacian of Gaussian (LoG) filter on the image and looking for local maxima. The result is obtained by summing the second order spatial derivatives of the gaussian- filtered image, and normalizing for scale. Local maxima in the filtered image yields spot detections. Each detected spot is assigned a **quality** value, that is obtained by taking the intensity value in the LoG filtered image at the location of the spot. So by properties of the LoG filter, this quality value is larger for :

- bright spots;
- spots which diameter is close to the specified diameter.

The DoG detector requires only two parameters: the estimated diameter of the object we want to detect, and a threshold on the quality value, that will help separating spurious detections from real ones. The panel you are presented let you specify these parameters, and preview the resulting detection. Try with 10 pixels for **Estimated diameter** and 0 for the **Quality threshold**. The click on the Preview button. A preview panel should open shortly, showing detection results on the current frame (Figure 5).

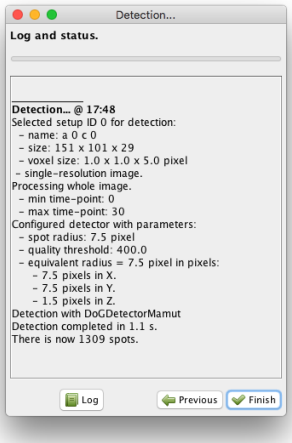


Figure 6.: Detection results.

These values are close but not quite. You can see that the diameter value is too small to properly grasps the elongated shape of the cells along Z (the BDV view on the left panel above is rotated to show a YZ plane). Also the threshold value is too low, and some spurious detections are found below the epithelium. These spots have a low quality, that manifests as a peak at low value in the quality histogram displayed on the configuration panel. From the shape of the histogram, we can infer that a threshold value around 100 should work. However we also need to change the diameter parameter, which will change the range of quality values. After trial and errors, values around 15 pixels for the diameter and 400 for the threshold seem to work.

Note that you can run the preview on any frame. You just have to move the time slider on the preview window. Once you are happy with the parameters, click on the Next button. All the frames specified in the ROI (if any) will be processed. In our case detection should conclude quickly and the following panel should appear:

We now have more than 1000 cells detected and this concludes the detection step (Figure 6). Click on the Finish button, and the wizard will disappear.

If you have complex images mixing several size of objects, or detection parameters that work for one part of the movie but not for another one, you could restart a new detection now, selecting for instance other parts of the movie with the ROI. You can do this and more, but for this kind of approach, the **Advanced DoG detector** offers more configuration capabilities, that will review later.

1.5. Linking cells.

What we just did is the detection step. It yields one Mastodon spot per cell, but the notion of cell identity propagated over time is missing yet. The particle linking step just does that. A particle linking or tracking algorithm accepts a collection of spots, ordered by frames (time-points), and tries to link each spot to the next spot(s) in the next frame or so. All the spots you can reach by starting from one spot and navigating across links build a *track*, and in our case it represents one cell (or any other object) followed over time. In most cases there is one spot per frame for a track, meaning that that a spot has at most one incoming link (spot from previous frame) and one outgoing kink (spot in next

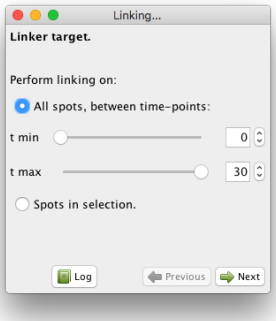


Figure 7.: The first panel of the linking wizard.

frame). But some algorithms can accommodate *e.g.* dividing cells (2 outgoing links for the mother cell going to the two daughter cells) and merging events. There is a vast literature behind tracking algorithms, and it is an active domain of Research. A relatively recent paper compare implementation and list some pros and pitfalls of many of them [7].

1.5.1. Selecting target spots for linking.

Like for the detection step, linking in Mastodon happens in a wizard. And also like for detection, the linking algorithms currently available in Mastodon are adapted from TrackMate. Launch the wizard from the GUI, with the Plugins » Tracking » Linking... menu item. The first panel you are shown lets you select what spots to include in linking (Figure 7). There are two modes:

- Either you take all the spots between a start and and end frame. By default, all frames are selected.
- Either you specify you want to link only the spots that are in the selection. This mode offers a lot of flexibility when facing complicated cases. It is best use along with the selection creator, that we will introduce later in this manual.

For now, just leave the parameters as they are, which will include all spots in the linking process, and click next. You can now choose between several linking algorithms.

1.5.2. Available linking algorithms in Mastodon.

In Mastodon, they fall mainly in two categories.

The first two LAP trackers are based on the **Linear Assignment Problem (LAP) framework**, first developed by Jaqaman *et al.* [8], with important differences from the original paper described elsewhere [5]. We focused on this method for it gives us a lot of flexibility and it can be configured easily to handle many cases. You can tune it to allow splitting events, where a track splits in two, for instance following a cell that encounters mitosis. Merging events are handled too in the same way. More importantly are gap-closing events, where a spot disappear for one frame (because it moves out of focus, because detection failed, ...) but the track manages to recuperates and connect with reappearing spots later.

In Mastodon the LAP algorithms exists in two flavors: a simple one and a not simple one. There are again the same, but the simple ones propose fewer configuration options and a thus more concise configuration panel. In short:

- The simple one only allows to deal with gap-closing events, and prevent splitting and merging events to be detected. Also, the costs to link two spots are computed solely based on their respective distance.
- The not simple one allows to detect any kind of event, so if you need to build tracks that are splitting or merging, you must go for this one. If you want to forbid the detection of gap-closing events, you want to use it as well. Also, you can alter the cost calculation to disfavor the linking of spots that have very different feature values.

The third tracker is called **Linear motion Kalman linker**. It can deal specifically with linear motion, or particles moving with a roughly constant velocity. This velocity does not need to be the same for all particles. It relies on the Kalman filter[7] to predict the most probable position of a particle undergoing (quasi) constant velocity movement.

1.5.3. How to pick the right linking algorithm?

The right choice of a particle linking algorithm is conditioned by the expected motion of the object you track. As a rule of thumb, you can make a decision following these simple rules:

- If the objects you track are transported by an active process and have a motion for which the velocity vector changes slowly, then pick the **Linear motion Kalman linker**.
- If the object motion is random (like in Brownian motion) or unknown, pick on the LAP linker. If the objects you track do not divide, nor merge, pick the **Simple LAP linker**.
- If the objects divide or merge, or if you want to specify linking costs based on numerical features (like spot mean intensity), then pick the **LAP linker**.

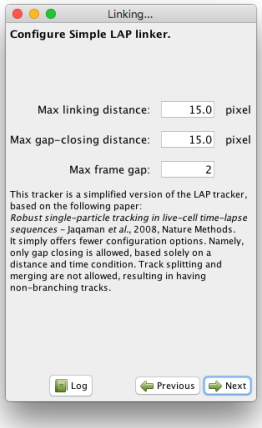
In our case, we need the **Simple LAP linker**. Select it and click Next.

1.5.4. Running the Simple LAP linker.

This linker only requires the specification of three parameters.

The first one is the **Max linking distance** during frame-to-frame linking. It is the distance beyond which linking a spot to another one in the next frame will be forbidden. For instance, if you know that your objects move by at most 5 µm from one frame to the next, pick a value slightly larger, for instance 6 µm. Distances are expressed in whatever physical units the BDV dataset specified. In our case it is pixels.

[7] Kalman filter on Wikipedia.



In practical cases, it can happen that the detection step might miss an object in some frames, then detect again later. These gaps will result in generating several small tracks for a single objects, which is one of the main course of spurious results when analyzing tracks. The simple LAP linker can bridge over missed detections. It does so by inspecting small tracks that results from frame-to-frame linking, and tries to connect the end of one with the beginning of another one. The last two parameters of the linker specifies how they are bridged. The **Max gap-closing distance** specifies how far can we look for candidates when we try to bridge the end of a track with the beginning of another one. The **Max frame gap** specifies how far they can be in time. For instance a Max-frame-gap of 2 means that we can bridge the end of a track at frame t with the beginning of a track at frame $t + 2$. Which results of bridging over detections missed by no more than 1 frame.

In our case, the default parameters turn to work fine. Click Next and the linking will proceed. Click on the Finish button to end the tracking process. If you have a BDV window opened, it should be updated with the tracking results, like in figure 8. By default the tracks are represented by colored lines, extending backward in time. Points in tracks that are close to the current time-point are green and fade to ref for points that are far back in time. When you change the Z focus, the spots are painted as circle of radius corresponding the the intersection of the sphere with the current Z-plane. When the spot sphere does not interest with current Z-plane, it is painted as a small dots. The points of the track away in time that are not close to the current Z-slice are faded away. We will see later how to customize the display of tracks.

1.6. Wrapping up.

This concludes our first tutorial on automated tracking with Mastodon. To continue with the next chapter, save the project with the tracks you just generated.

As you can see, after creating a project from a BDV file, the process consists mainly in running in succession the two wizards, one for detection, one for particle linking. Even if they provide fully automated tracking, Mastodon is made for interacting with the data as you generate it. We will see in a next section how to manually edit a spot, a link or a track even at the finest granularity. But keep in mind that the tools we quickly surveyed can be used interactively too. First the wizards let you go back to change a tracking parameter and check how the results are improved or not. Second, because

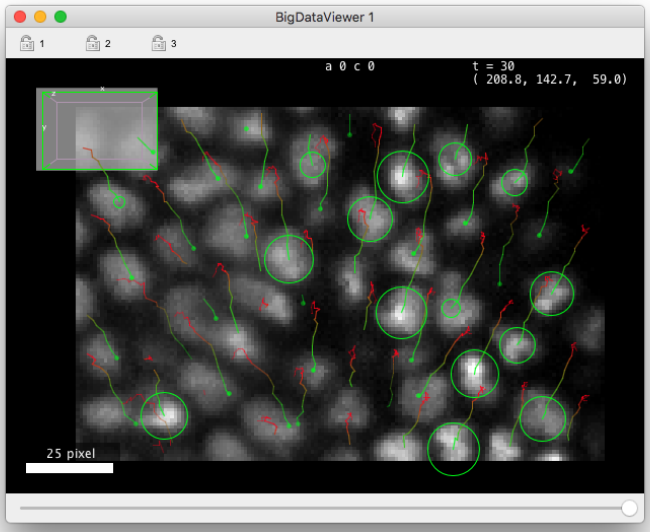


Figure 8.: How tracking results are displayed in BDV views.

you can specify a region-of-interest (ROI) in the detection step, and select the spots you want to track in the linking step, several runs of these wizard can be combined on different parts of the same image, to accommodate *e.g.* for changing image quality over time, or cell shape over time. Mastodon aims at being the workbench for tracking that will get you to results, accommodating a wide range of use cases.

2. Manually editing tracks in Mastodon. TrackScheme.

One of the key feature of Mastodon, the implementation of which made our lives extremely hard and rich, is the ability to edit manually at any time any spot or link within an annotation that can contains billions of them, while retaining a good and pleasant response time from the software. I am not even speaking of the undo/redo mechanism, which getting right was also very interesting.

In this tutorial, we will show how to do just that, from existing tracking data, the one we generated in the previous section. We will use it as an opportunity to present TrackScheme , the track visualizer of Mastodon, as well as introduce several useful editing features such as the undo/redo mechanism mentioned above. We will also introduce the various visual hints that help the user knowing what spot or link is currently edited or inspected, such as the **focus**, the **highlight** and the **selection**.

Manual editing is often **not desirable**, first because it is not objective which might be detrimental in situations where the performance of an algorithm is evaluated, or when an unknown motion characteristics is investigated. Second, because it does not look like a realistic solution when *e.g.* thousands of cells are to be followed over thousands of time-points. Regarding the latter however, several courageous individuals sacrificed their time, energy and sometimes sanity in doing so, for the sake of finally getting a scientific answer when there was no other working solutions. See for instance [3] and [9]. If you are going this way, Mastodon aims at providing tools to do so, in the hopefully fastest and least painful way possible. Also, you can combine the fully automated approach of the previous chapter with manual editing to correct hopefully a few numbers of mistakes. Finally, there is also a semi-automated tracking tool, that we will introduce later.

This tutorial will start by presenting you the manual editing tools of Mastodon, as well as TrackScheme and the focus, highlight and selection tools. Because this catalogue of actions can be a bit dry and long to swallow, we will in the last paragraph use it to remove and rebuild some tracks from the results of the last tutorial.

2.1. TrackScheme, the lineage view and editor.

First, load or retrieve the tracking data of the previous chapter. You should have a few hundreds of tracks. To open a TrackScheme view window, press the trackscheme button on the main window (Figure 3 page 10). A window as shown in figure 9 should open.

The TrackScheme view offers a special way of displaying tracks. If you are familiar with TrackMate, you will see that we brought the same kind of features here, but scaled to large data. You can think of TrackScheme as a workbench for tracks, where you will edit, cut, stitch and rename them. It displays a kind of "track map", where a track is laid on a panel, arranged vertically over time, as a Parisian subway train map. Tracks are displayed hierarchically, discarding the spatial location of each spot. Each track is laid out going through time from top to bottom. One horizontal line corresponds to a single time-point in the movie. One vertical column corresponds to a single track, that is all the spots that are connected by links over time, including divisions and merges. It is a great tool particularly to study and edit cell lineages.

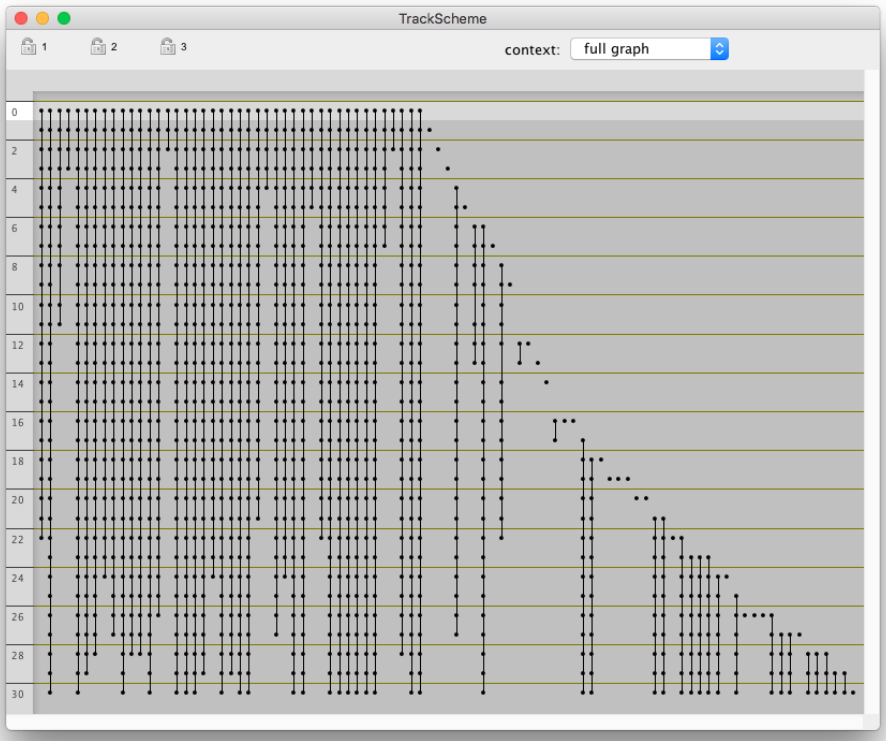


Figure 9.: The TrackScheme window.

When opened, the view in TrackScheme is scaled so that the full data is shown, all tracks over all time-points. Even on our small dataset, most of the tracks and spots appear as single lines or dense boxes. To see the details of each track, you need to zoom in and navigate around the data. The navigation actions their mappings are listed in the table 2 below.

Table 2.: Default navigation key-bindings for Mastodon-TrackScheme views.

Action	**Key**
	View.
Move around.	Right-click and Drag or Mouse-wheel
Zoom / unzoom in X.	⇧ + Mouse-wheel
Zoom / unzoom in Y.	ctrl + Mouse-wheel
Zoom / unzoom in X & Y.	ctrl + ⇧ + Mouse-wheel
Full zoom, full unzoom.	Press Z. The view zoom at max level to the mouse location. Pressing Z again to unzoom fully.
Zoom in a box.	Press and hold Z then drag a box. The view will zoom to the box.

Try to zoom in until you can see the label of a few spots. TrackScheme implements adaptive level of details depending on the zoom level, so that we can accommodate plotting a large amount of

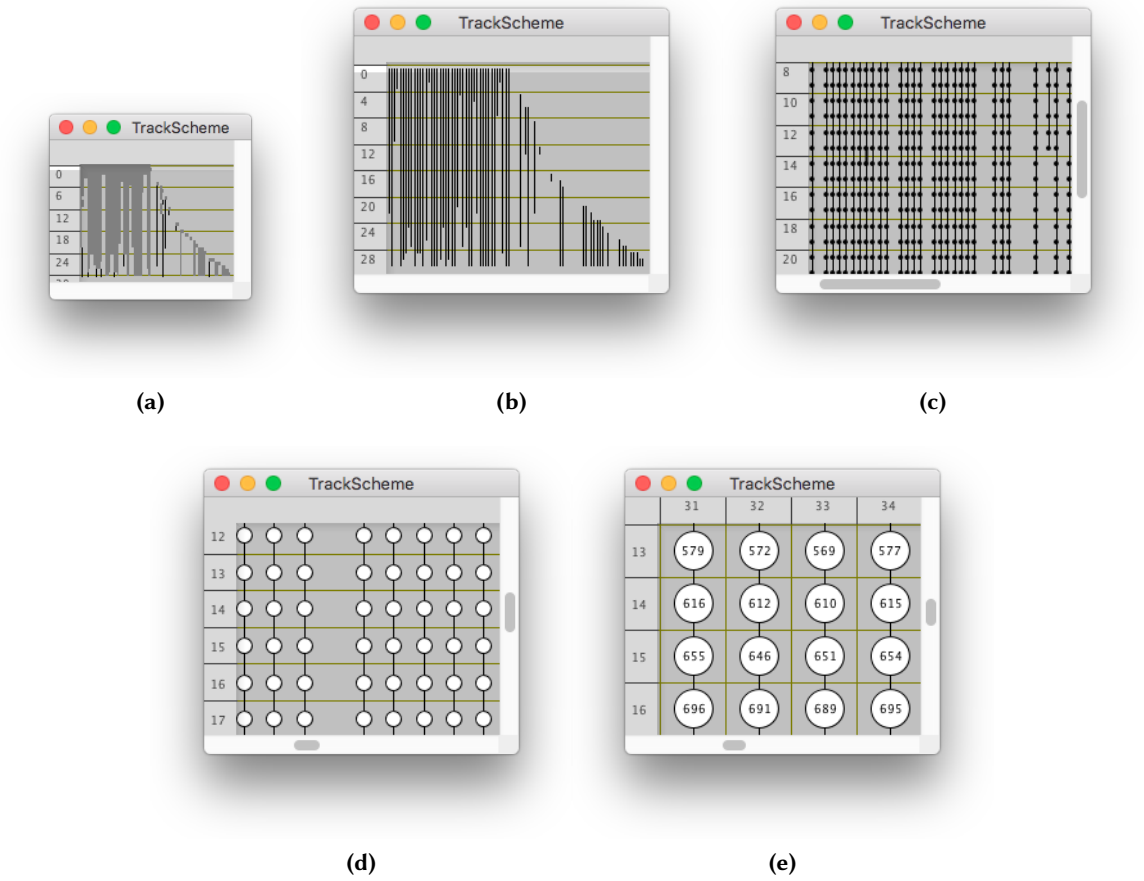


Figure 10.: TrackScheme displays spots and links differently depending on the zoom level. **a.** At low zoom level tracks that coalesce are shown as gray boxes. **b.** Zooming in, the individual tracks appear first as black lines. **c.** When they are separated enough, spots appear as black dots. **d.** With an even higher zoom level they are shown as empty circles. **e.** Until they grow big enough so that the spot labels can be painted. Notice the size of the sliders at the bottom and at the right of the view.

data without compromising the responsiveness of Mastodon too much (Figure 10). On the finer level of details, spots are plotted as circles, with the spot label shown. As you zoom out, they becomes just empty circles, then points, then they disappear to only show the track as a line. When the zoom level is so low that several tracks coalesce, they are drawn as a gray box.

2.2. The focus and the spot labels.

You probably noticed that some spots are painted differently from their neighbors. Mastodon manages three special collections of spots and links to facilitate making sense of the data across views:

- the **selection**, painted in green, that manages a classical selection of spots and links;
- the **highlight**, used to highlight the spot or link currently under the mouse;
- the **focus**, particularly used in TrackScheme, to indicate what spot is currently focused on by the keyboard interaction.

We will first present the focus (see Table 3).

Table 3.: Default key-bindings for the focus in TrackScheme and BDV views.

Action	Key
Navigation with the Focus.	
Follow a spot across time within a track with the focus.	↑ and ↓
Jump to the beginning of a branch.	⌥ + ↑
Jump to the end of a branch.	⌥ + ↓
Jump to the beginning of another branch.	ctrl + ⌥ + ↑
Jump to the end of another branch.	ctrl + ⌥ + ↓
Focus in TrackScheme.	
Move the focus around from one track or one track branch to another.	← and →
Edit the label of the focused spot.	⏎

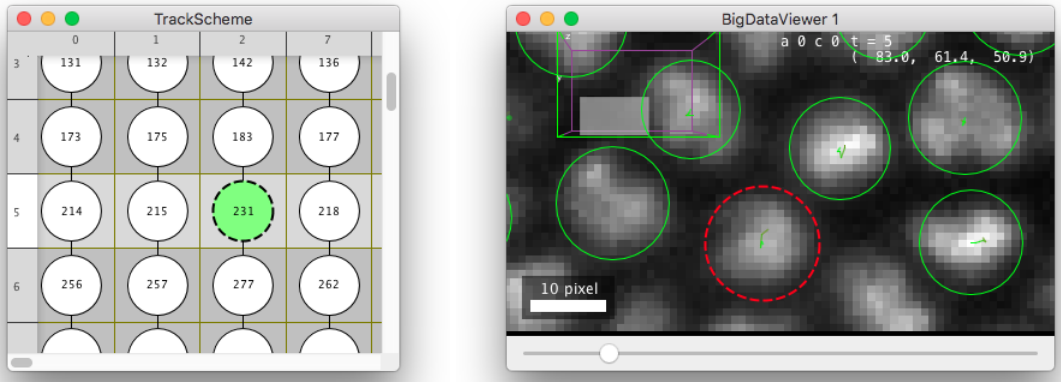


Figure 11.: Focused spots are painted in TrackScheme (left) and in BDV (right) views as ellipses with a dashed-contour.

2.2.1. Moving the focus.

The spot that has the focus is painted with a dashed contour (Figure 11, left). Only one spot can have the focus and it is set by mouse or keyboard interaction. Click on one spot in TrackScheme when fully zoomed, and move across track and time with the arrow cursor keys ← → ↑ and ↓. If the focus reaches the border of the window, the view will be moved to follow it. You can think of the focus as the caret in a text editor. It is meant to facilitate keyboard interaction.

You can also jumps across branches. A branch in a track is a linear section of the track between divisions, fusions or the track end or start. ⌥ + ↑ and ⌥ + ↓ will move the focus to the branch start and end. In the case where you could navigate to several branches, for instance you are currently at a cell division and you can go the end of either daughter cells branches, you can navigate to one or the other with ⌥ + ↓ or ctrl + ⌥ + ↓.

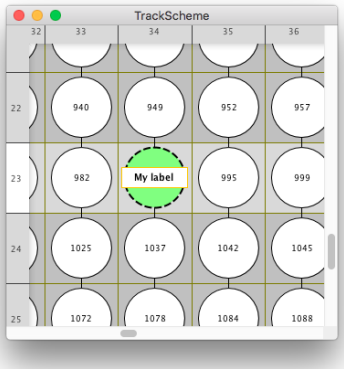


Figure 12.: Editing a spot label in TrackScheme .

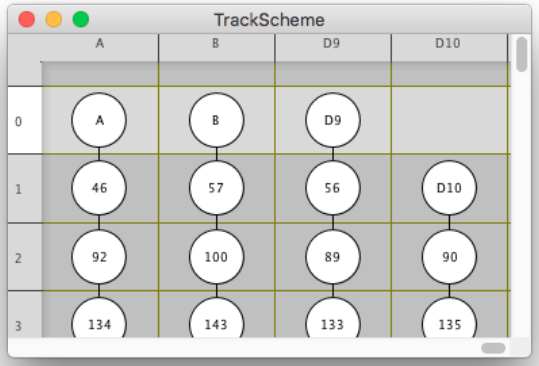


Figure 13.: The left to right order of tracks in TrackScheme. Track names are set by the label of their first spot in time, regardless of the time-point this spot is in. Tracks are then laid out by alphanumerical order of the track label.

2.2.2. Editing the spot labels.

By the way, the focus is used to rename individual spots. Zoom in so that we can see the label of the spots and move the focus to a spot. Press ⏎. A small editing box appears inside the spot and lets you change its label (Figure 12).

By default the spot label display the spot ID. If you edit the label then it shows the new label you entered. Editing the label of a spot does not affect its ID or any other properties. The spot label is just a convenient text field that you can use to annotate cells and search them later. Only the spots have a label. The links don't.

2.2.3. The order of tracks in TrackScheme.

The spot labels also control how the tracks are ordered in TrackScheme. We said before that in TrackScheme all the spatial information is discarded. However the tracks are laid out in a deterministic order. This order is set by the label of of a track.

There is no special structure to follow individual tracks in Mastodon. For this, we simply use the first spot of a track. So when we speak of the label of a track, we simply means the label of the first spot (in time) in the track. The tracks are arranged from left to right following the alphanumerical order of the track labels. So you can change the tracks arrangement by editing theirs first spot's label. For instance a track named `A` will be laid out to the left of a track named `B`, and a track named `D9` will be put to the left of a track named `D10` (Figure 13).

If you change the track labels now, the tracks will not move immediately in TrackScheme. For the new arrangement to happen, you need to either open another TrackScheme window, or to edit the data, which we will see soon.

Figure 14.: Three views of the same dataset in sync. Notice that the lock number 1 is activated on the three views (in yellow). The second view is zoomed in a XY plane. The third view is dezoomed and align with XZ.

2.2.4. The focus in BDV views.

The focus also works in the BDV views. In these views, the spot that has the focus is also painted with a thick dashed circle (Figure 11, right). Parenthetically, notice that the focus is shared across all opened views. If you have a TrackScheme view and a BDV view opened, and that they both contain a display of the same spot, setting the focus in one view will update this very spot display in the other views.

When you are in a BDV view and have set the focus, you can also navigate with the arrow keys. After clicking inside a cell, you can navigate in time and follow a cell lineage through the movie with ↑ and ↓. If you reach the border of the view or if the cell moves in Z, the view will be translated to keep the cell in focus. However, you cannot edit the spot labels in a BDV view nor jump from one track to another with ← →. Navigating across branch with ⌥+↑ and ⌥+↓ *etc* also works.

2.3. Synchronizing several views together.

You probably noticed that both BDV and TrackScheme views have a toolbar on the of the view itself. This toolbar can be hidden and shown by a press of the T key. In both view types, the toolbar contains 3 gray locks, that are used to link several views together for navigation. For instance, if you click on the lock 1 on a BDV view and the same lock on a TrackScheme view, they will be synchronized. If you move with the focus in the TrackScheme view, the BDV view will be translated to show the focused spot. If you Double-click on a spot in any view, all views in sync will translate to show the spot. This is very handy to navigate around in TrackScheme, for instance following a cell over time while the BDV view displays where it is in the sample. You can even combine several BDV views in sync at different magnification to have both an overview of the cell position in the sample and a close view of the cell itself (Figure 14).

2.4. The highlight.

The highlight is the second visual hint we will introduce. You probably already noticed it and used it: the display of spots and links that are just below the mouse cursor changes. Highlighted spots and

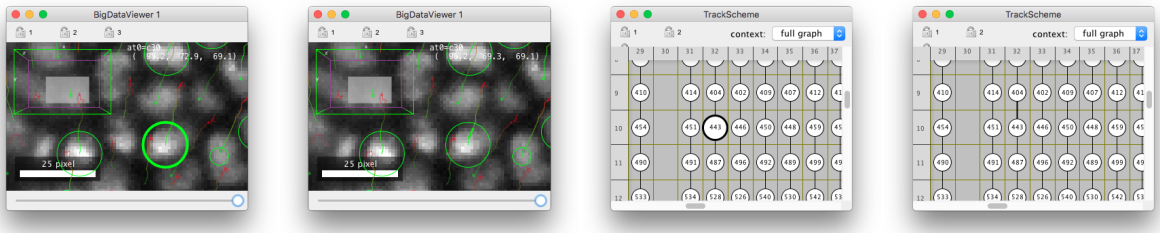


Figure 15.: The highlight in Mastodon. The highlighted spot or link will appear painted with a thicker line, both in the BDV and TrackScheme views.

links are painted with a thick continuous line (Figure 15). To highlight a spot or link you just have to lay the mouse over it. As for the focus, the highlight is common to all views, and if you highlight a spot or a link in a view, its display is changed in all the views that show it.

2.5. Deleting individual spots and links.

The highlight is especially important for track editing. The commands that delete a single object are actually applied to the highlighted spot or link. For instance, to delete a spot or a link in TrackScheme , simply move the mouse cursor over it so that it is highlighted and press D. You should see the tracks being rearranged following the deletion. In a BDV view, the highlight and deletion mechanism is the same.

2.6. Linking spots together.

Creating links between existing spots happens in a similar way, except that you have to point to a source spot and another target spot for a single link. In TrackScheme , move inside a spot until it is highlighted (no mouse-click). You must be at a zoom level large enough so that a spot is at least painted as a dot or a circle (Figure 10c, d, e). Then press and hold the L key. Without releasing the L key, move the mouse out of the spot. You should see a red line representing the link to create (Figure 16). Move the mouse to the desired target spot, until it becomes highlighted. The red line should snap to it and become thicker. If you now release the L key now, a new link between the source and target spot is created and in TrackScheme it should cause a rearrangement of the tracks. Notice that you cannot add a link between spots in the same frame. Also notice that you can toggle links this way. If you draw a link between two spots that are already connected, their link will be removed.

In BDV views, it is again very similar (Figure 17). Move the mouse over a spot until it is highlighted then press and hold the L key. The view will automatically move to the next frame. The link to create will be painted as a white line, and the ghost shape of the source spot is painted as a dashed white ellipse. Move the mouse to the target spot in this frame until it is highlighted, then release the L key. The link is created. If you press ⇧ + L in a spot, the BDV view will move to the **previous** frame, and create a backward link.

You cannot move in Z while creating a link this way. You must orient the BDV view so that the

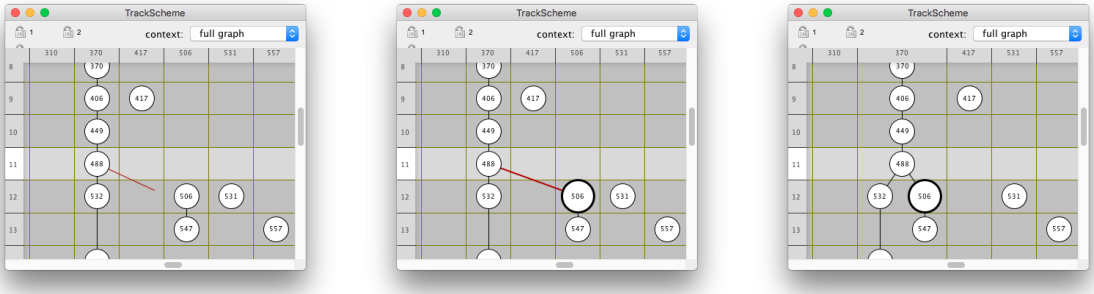


Figure 16.: Manually creating links in TrackScheme. Press and hold L while hovering the mouse over a spot. Then drag the link in red to the target spot. A new link will be created between the two spots when you release L. Doing this between two spots that are already connected removes the link between them.

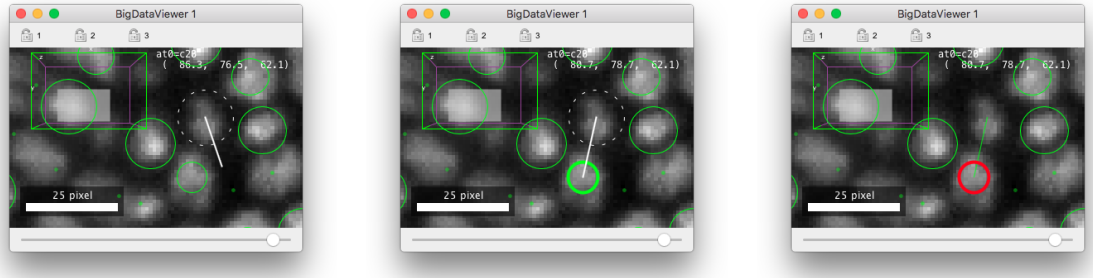


Figure 17.: Manually creating links in BDV views. Press and hold L while hovering the mouse over a spot. The viewer moves to the next frame and paints the source spot as a dashed, white outline, and the link to create as a white line. Then drag the white link to the target spot. A new link will be created between the two spots when you release L. As for TrackScheme , doing this between two spots that are already connected removes the link between them.

source and target spot are roughly in the same displayed slice. However you can move in time. While still holding the L key, press the N or [and M or] key to navigate backward and forward in time. This way you can create links between spots separated by more than one time-point.

2.7. The selection.

The selection is the third visual cue of Mastodon and probably the most important one. It behaves and has the role of a classical selection tool you would find in another software that manages a collection of data items. It is meant to select a subset of spots and links and apply operations to this subset. Contrary to the focus and the highlight, any number of spots and links can be put in the selection.

The selection in TrackScheme is built in a classical way. Click and drag from an empty part of the view. A red box appears to let you select an area of the view. Release it when it contains the part of the tracks you want to select, and selected items should now be painted in green (Figure 18). All spots and tracks in the selection box are put in the selection, even if the zoom level is too small for them to be painted. ctrl + A selects the whole data. There are variants to select only all the spots (ctrl + ⌥ + A) or only all the links (ctrl + ⇧ + A).

⇧ + click on a spot or a link to toggle it in the selection. To clear the selection, click on an empty

part of the TrackScheme view. In TrackScheme you can also add spots - but not links - to the selection using the focus. For instance, if you combine navigating with the focus while holding the ⇧ key, spots that you navigate to will be added to the selection. ⇧ + ⌥ + ↓ will add all the spots from the focused spot to the end of the branch it belongs to. Finally there are commands to add a full track from the currently focused spot. ⇧ + Space will select the whole track of the focused spot. ⇧ + Page↓ selects all the spots and links in the track that are forward in time relative to the focused spot, and ⇧ + Page↑ does the converse backward in time.

The selection is shown in BDV views as red or magenta spots and links by default. In BDV views you can only edit the selection via ⇧ + click. The interaction with the focus also work as described above. These default key-bindings are recapitulated in the table 4.

Table 4.: Default selection key-bindings for Mastodon views.

Action	**Key**
Setting selection.	
Add a spot or link to the selection.	⇧ + Left-click. Do it again to remove a selected spot or link from the selection.
Select all spots and links.	ctrl + A
Select just all the spots.	ctrl + ⌥ + A
Select just all the links.	ctrl + ⇧ + A
Select all the spots and links in the track of the currently focused spot.	⇧ + Space
Select all the spots and links *upward* (backward in time) in the track of the currently focused spot.	⇧ + Page↑
Select all the spots and links *downward* (forward in time) in the track of the currently focused spot.	⇧ + Page↓
Add the previous / next spot in time to the selection.	⇧ + ↑ / ⇧ + ↓
Add all spots to the selection from current spot to the beginning of a branch.	⇧ + ⌥ + ↑
Add all spots to the selection from current spot to the end of a branch.	⇧ + ⌥ + ↓
Add all spots to the selection from current spot to the beginning of another branch.	⇧ + ctrl + ⌥ + ↑
Add all spots to the selection from current spot to the end of another branch.	⇧ + ctrl + ⌥ + ↓
Selection in TrackScheme.	
Add the spot to the left / right of the focus to the selection.	⇧ + ← / ⇧ + →
Draw a selection box	Left-click and Drag.

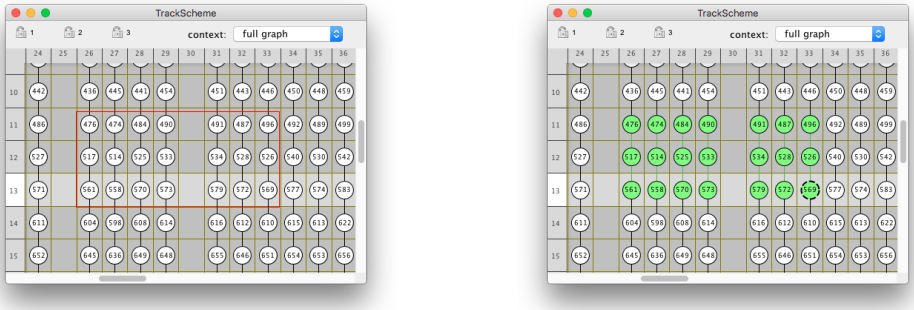


Figure 18.: The selection box in TrackScheme .

2.8. Editing spots and links with the selection.

The selection actions are all applied in bulk. You can delete all the spots and the links currently in the selection with ⇧+⌫. If your selection includes spots that are not linked, pressing ⇧+K will link them sequentially. If the selection contains more than one spot per frame, only one of them, picked randomly, will be linked.

2.9. Manually adding spots and linking them.

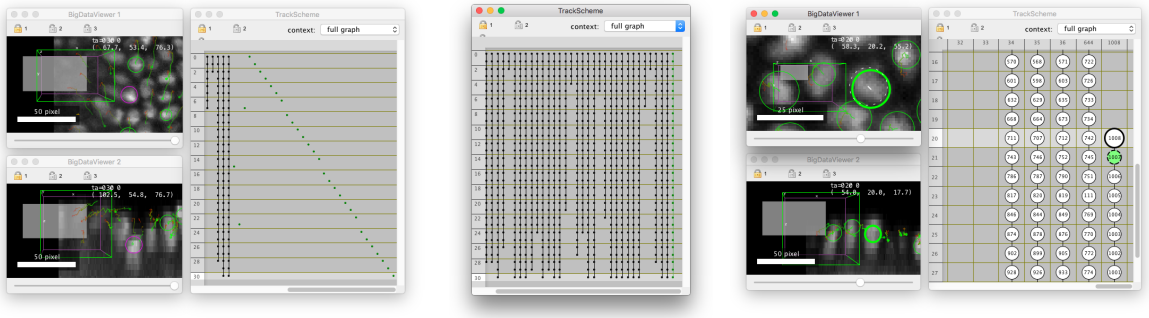
Adding new spots requires specifying where to add them. Since TrackScheme does not include spatial information you cannot use it to add spots. You can only add spots in the BDV views.

To add a new spot, simply hover the mouse in a BDV view where you want the spot to be created, and press A. A new spot will appear, with a radius given by the last radius value you used. You can respectively decrease or increase the radius of a spot by pressing Q and E. Modifiers affect the amount of radius changes. ⇧+Q and ⇧+E change the radius by a larger amount and ctrl+Q and ctrl+E by a finer amount.

Spots created this way are not linked. It might not be the quicker way to follow manually a cell over several frames. To do so, you can use a little trick. When you want to add a spot, and link it to an existing one, arrange the BDV view so that this source spot is visible in the displayed time-point. Then place the mouse cursor *inside* the source spot, and press and hold A. Like for creating links, the BDV moves to the next time-point, paint the source spot and link with white, dashed lines, and let you position the new target spot. While you keep A pressed, you can move the target spot around. When it is in the desired position, release A. The new spot is created and linked to the source spot. This way, you can quickly follow a cell or an object and manually creates a track for it by just positioning the mouse with a few presses of A.

2.10. Moving spots around.

To move an existing spot we use a similar interaction with the spots. In a BDV view, place the mouse cursor inside the spot you want to move, then press and hold Space. While you hold Space pressed the spot will move where you move the mouse. Release the Space key at the desired location.



(a) (b) (c)

Figure 19.: Manual tracking in Mastodon. **a.** After selecting the spots we added to each time-point. **b.** After linking them with shift + K. **c.** Backtracking a cell with shift + A.

2.11. The undo/redo mechanism.

One of the most useful actions is undoubtedly undo and redo. To undo just press ctrl + Z and to redo use ctrl + ⇧ + Z. There is no limit to the number of undos you can do. And everything can be undone in Mastodon, even tag definitions. In our humble opinion, this is one of the core reasons for Mastodon, a tracking and lineaging software that aims at combining automated and manual approaches, to exist.

2.12. Putting things in practice.

Make sure you have one BDV view and one TrackScheme view open, and that both are linked using the same lock 🔒. In TrackScheme , select one of the spot or link that belongs to one of the long tracks; the ones that extend from the first time-point to the last. Press ⇧ + Space. You selected the whole track to which the spot or link you selected belong to. Press ⇧ + ⌫. The track has been removed.

Let's remove a couple of other tracks another way. Click and drag in TrackScheme to draw a selection box around some long tracks. Then press ⇧ + ⌫. Our model should now lack several good tracks we will try to put back manually. For clarity, also remove all the tracks on the right part of the TrackScheme view, that do not start at the first time-point.

In the BDV view, go to the first time-point, and look for a cell whose spot has been removed. Pan the view (right-click and drag), adjust the zoom (ctrl + ⇧ + Mousewheel) and the Z position (Mousewheel with or without ⇧) so that the cell is well in view. Then put the mouse cursor over it and press A. A spot has been added on the cell. The fine adjustment of the spot position in 3D can be made easier with a second BDV view. Open another BDV view, zoom and rotate it so that it is aligned with the XZ plane by pressing ⇧ + Y. Add it to the lock group by toggling the right lock 🔒 in the toolbar. Go back to the first BDV view, and align it with the XY plane by pressing ⇧ + Z. You must now find the spot we just added. If you cannot find it in any of the two BDV view, look for it in TrackScheme . It will be in the rightmost column at the first line, since we added the spot to the first time-point. Double-clicking on it in TrackScheme will center the two BDV views on it. Now that the two BDV views are centered on the spot, you can adjust its XY and Z positions over the cell using the two views. In any of the two BDV views, put the mouse cursor, then press and hold Space. Move the spot the

desired location and release Space. The spot might not have the right diameter to encompass the cell. Change its radius by pressing ⇧ + Q and ⇧ + E (without ⇧ for finer adjustments) until you are happy with it.

You must now do it for the same cell in the subsequent time-points. Press M in a BDV view to move to the next time-point. Again, put the mouse cursor roughly at the center of the cell and press A. Adjust its position with Space and move to the next time-point.

We have to repeat this for all the time-points of the movie (Figure 19a). This might be tedious but does not have to be too long. When I need to do this I adopt a posture similar to what I have when I play PC video-games: the right hand on the mouse, the left hand over the left side of the keyboard, over the A, D, Q, and E. The ⇧ key can be accessed with the little finger, the Space key with the thumb. The N and M keys to navigate over time are accessible with the left thumb also.

We created a bunch of spots, but they are not linked. Move to the TrackScheme view. You will find the spots you last created there always at rightmost part of the view. Since they are not linked, they should appear each in their separate column, in a stairway manner. Select them all by dragging a selection box around them and press ⇧ + K. They should now link together into a new track (Figure 19b). With one of the spot of this newly created track selected, press ⌥ + ↑ to jump to the first spot in the track. Then press ↩ to edit its name into something like `My first track`.

We can create a track directly. Let's do this by backtracking a cell from the last time-point to the first. In a BDV view, move to the last time-point, and place the mouse cursor over an annotated cell. Create a spot over it by pressing A and center and resize it until satisfaction. Now keep the mouse cursor inside this spot, and press ⇧ + A and hold the A key. The BDV view moves to the previous time-point and creates a new spot there, already linked to the other one. By repeating this you can create a track that will backtrack the cell until it appears in the movie (Figure 19c).

We have now restored two of the tracks we removed at the beginning of this paragraph. Hopefully now you feel enough at ease to do these manipulations quickly, efficiently and without too much hassle. But one of the concluding beauty of this chapter is that you can restore the data as we had it at the beginning by roughly 100 presses of the ctrl + Z keys.

3. Getting your bearings in large datasets.

In the previous chapter we have seen how to edit single spots and links in Mastodon, what can be called point-wise editing. As we said before, the goal of Mastodon is to let you harness very large images, for which the number of annotations can be very large too. It can be very easy to get lost within such large images and loose track of where we are within the sample image and what cell we follow. So we have added several features that are made especially to get your bearings in large datasets. More than anything, these features are about giving visual cues that ease orientation, and exploit events and signal that would help a human brain get a sense of orientation. We took some inspiration from video-games, that are very good at communicating condensed and synthetic information to the player (but only for a limited part; there is no screenshake when you delete a spot).

3.1. Bookmarks in the BDV views.

The BigDataViewer [1] is the image component of Mastodon and is meant to deal with very large images. It offers interactive and responsive user interaction, and to achieve this without any hardware acceleration, it resorts to only displaying a 2D slice through the data. This 2D slice can be arbitrarily positioned and oriented. When it comes to annotating a 3D image, using a 2D slice is a good approach. A full 3D view might actually hinders proper and efficient annotation of the data with a flat 2D screen and a mouse. The 3D view leads to ambiguities about the depth positioning of your cursor, and the image data that stands between the camera eye and the plan of interest may hide it. Parenthetically, these issues with interacting with 3D data are best solved with virtual reality devices, but Mastodon is not a tool that exploit them. The 2D view offer clarity but conversely does not offer a great feeling of the context. We have to live with that.

However to facilitate orienting yourself, or retrieving a key point in the data, you can register bookmarks in the BDV views. The bookmarks were already implemented in the BigDataViewer tool itself before its use in Mastodon. They let you store a position and orientation in space as bookmarks. You can later call them again and retrieve said position.

- First move to the position and orientation you want to store in a bookmark.
- Then press ⇧+B. You should see a message prompting you to press another key (Figure 20).
- Pick one and press it. This key will be used as a tag for this bookmark.
- To later retrieve the position and orientation of this bookmark, press B then the bookmark's key. The view should animate and restore the stored position and time-point.
- O does the same things, but only restore the bookmark orientation, not its position.

You can have many bookmarks, all identified by the key you press after the bookmark command. Bookmarks are saved with the BDV settings file that also saves the channel color and display range. You can save such a file with the File » Save BDV Settings or the F11 key. The settings file is loaded when a new BDV window is displayed.

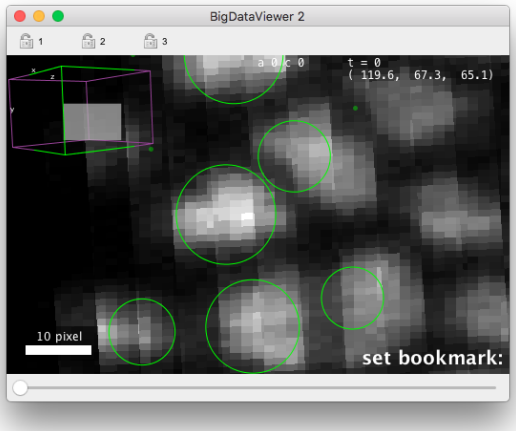


Figure 20.: Settings a bookmark in a BDV view.

3.2. View modes in BDV.

TODO TODO TODO

3.3. Linking several views together.

You can generate as many views as you want in Mastodon, and you can link several of them via the lock system . This is a good way to improve the perception of context, by linking several views that display for instance a close-up view of the data and another view displaying a bird-eye view of it. We already presented this feature in the Section 2.3 page 25. The figure 14 page 25 shows an example of a view configuration with three views in sync, showing each a different level of desired information. We direct you there for details on how to use the lock system.

3.4. In TrackScheme everything is animated.

If you went through the previous tutorial, you probably noticed that editing events are associated with animations in TrackScheme . For instance, if you delete a link in the middle of a track, the 'bottom' part of the track will move to the left side of TrackScheme, in a quick animation. If you undo the deletion, the branch will move back to its original place the same way. Deleting a spot makes it fade rather than disappear. These animations are more than a toy. Something we learned that hard way with MaMuT [3] is that point-wise editing the data can completely confuse and disorient the user. A single link deletion will generate a big rearrangement in the track hierarchy, and therefore will change the TrackScheme view a lot. Without any subtle cues to the user, these changes will disorient them quickly. Animating the editing events is a great way to hint them about what happens to the data modify in a user-friendly way.

We also added some fluidity and inertia in TrackScheme navigation. In the TrackMate [5] and MaMuT [3] version of TrackScheme , panning and zooming were done in discrete discontinuous steps, that would also lead to confusion. In Mastodon, moving and zooming are done continuously. There is

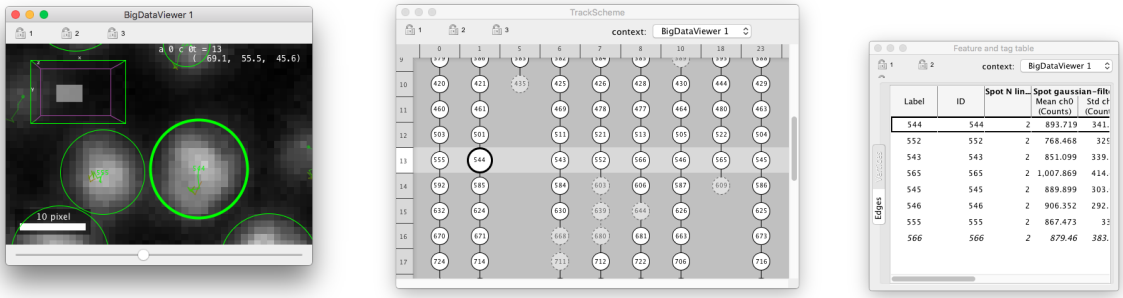


Figure 21.: Context in Mastodon. The context displayed in TrackScheme (center) and the table (right, explained in the section 4.3 page 42) is determined by what is currently displayed in the BDV window (left). The content displayed in TrackScheme and the table id updated live as the view in the BDV window is changed by the user.

even some inertia again to emulate interacting with a tangible panel.

3.5. Spatial context in TrackScheme.

There are situations where the density of cells displayed in TrackScheme is so high that even at high zoom level this view is barely useful. The hierarchical arrangement of tracks in TrackScheme is handy to grasp how tracks behave over time. But since it eliminates spatial information, the neighbors of a track do not bring information to it.

The spatial context in TrackScheme reconciles spatial information with the hierarchical layout. In TrackScheme views, the toolbar has a context list box, from which you can select between `full graph` and the names of all the BDV views currently opened (typically `BigDataViewer 1`, *etc*). When `full graph` is selected, the full lineage data is shown in TrackScheme . This is the classic view. If you pick an item corresponding to an opened BDV view, then this TrackScheme view will only display the lineages of the cells currently displayed in the target BDV view. And the TrackScheme view will be updated (and animated) as you pan, move in Z, zoom or unzoom the BDV view (Figure 21).

Try it now with the data from the previous tutorial. Open a BDV view and a TrackScheme view. In the TrackScheme view, select the `BigDataViewer 1` item (it might not be 1 in your case). Then in the BDV view, zoom so that almost only one cell is displayed. The TrackScheme view should display considerably fewer tracks. As you unzoom, some lineages will appear in TrackScheme . You will probably see that some lineages appear in gray, with dashed lines. They are called "ghost" lineages: These are the lineages of cells that are not within the BDV view at the time-point currently displayed, but that will enter this view later or earlier in time. The context feature is immensely useful when studying tissue development or coordinated cells movement.

4. Numerical features and tags. The table view.

Mastodon is a tracking and lineaging tool. Its output is a collection of tracks, and the analysis of these tracks to yield statistics on *e.g.* velocity, displacement *etc* is carried out in another software package such as MATLAB or Python. Nonetheless you will find in Mastodon tools to compute *numerical features* on data item. Numerical features are numbers that can be calculated on spots, links and tracks of the data. For instance there are feature for the number of links that touch a spot, or the displacement of a link or the number of spots in a track. You can find them within Mastodon because it is convenient, but also because they are very useful for the interactive exploration of your data. Coupled with feature-based coloring, the display and sorting of values in the table view and the selection creator tool, they can considerably accelerate and facilitate making sense of the data.

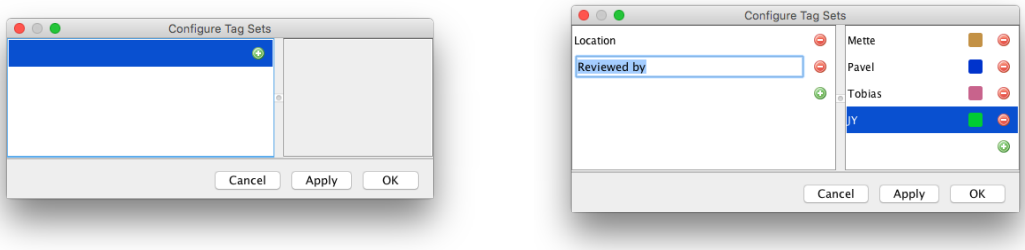
Numerical features are numbers that classically relate to a physical quantity. When we need to *categorize* items, we rely on *tags*. We describe them just below. This chapter will also show you how to compute numerical features and create a coloring view from the feature values and tags. Doing so, we will introduce the third kind of data view in Mastodon: the data tables.

4.1. Tags and tag-sets.

As we said above, every-time you need to categorize certain data items, or need to visualize categories, you should rely on tags. Let's suppose that you are investigating the trajectories of cells in a developing embryo from an early stage to a stage where the embryo is polarized. Some cells will migrate to the anterior part, some others to the posterior part, *etc* . You might want to tag cell tracks with the `Anterior` or `Posterior` tag, to investigate where do these cell come from in the early embryo. Or let's say that you are curating the results of the automated tracking on a large images. The tracking results might have some inaccuracies, and you want to correct them for important tracks. Because there is a lot of tracks, you share the workload with some colleagues. You work asynchronously with them, editing the Mastodon file one after another. Doing so, you can use tags in Mastodon to mark some tracks as reviewed by you. Your colleagues will use a tag for themselves, to ensure that no two scientists are reviewing the same track twice. All the cells that are not tagged in this categorisation are still waiting to be reviewed.

In Mastodon, a categorization corresponds to a **tag-set**. A tag-set defines a property that can have a reasonable number of discrete values, or **tags**. In the first of the two examples above, `Location` would be a tag-set to specify the location of cells. `Anterior` and `Posterior` would be two tags belonging to the `Location` tag-set. In the second example, `Reviewed by` would be a tag-set, and `Mette`, `Pavel`, `Tobias` and `Jean-Yves` would be 4 tags of this tag-set.

You can assign tags to spots and links. To assign a tag to a whole track, you have to assign this tag to all the spots and links of this track. One data item (a spot or a ling) can have 1 or 0 tags per existing tag-set. But they can be categorized by as many tag-sets as there is. For instance, a spot can have the tag `Anterior` in the `Location` tag-set, and the tag `Pavel` in the `Reviewed by` tag-set. Or it can be not tagged in the `Reviewed by` tag-set. But it cannot have both the tag `Mette` and the tag `Tobias` because they belong to the same tag-set. Each tag-set works independently, and clearing a



(a) **(b)**

Figure 22.: Creating tag-sets and tags. **a.** The empty tag-set dialog. **b.** After adding two tag-sets and six tags.

tag-set does not affect the others even for one data item. Now that we set things straight, let's see how to create tag-sets. We will base the demonstration in this chapter on the data we generated in the tutorial chapter 1.

4.1.1. Creating tag-sets.

On the main window (see figure 3 page 10), there is a button configure tags. Pressing it opens the tag-set dialog. Right now, it appears as an empty table made of two columns (Figure 22a). This is where you enter tag-sets and tags.

Press the ⊕ button on the left column to create a new tag-set. A default name is shown for the tag-set, that you can edit. You can also directly press the ⏎ key to immediately start editing the new tag-set. Let's say we want to create the location tag we talked about before. Type `Location` in the text field. An empty line followed by a ⊕ button should appear on the right column. This is where you will enter the tags of this tag-set. Click on this button, or press the ⇥ key followed by the ⏎ key to create a new tag. For instance, the `Anterior` tag. Note that a tag is only made of a label (the text) and a color. The color will be used in Mastodon views. Create a second tag for the same tag-set called `Posterior`. Pick the color as you like. Now try to create another tag-set called `Reviewed by` and create some tags in this tag-set (Figure 22b). The tag-set dialog is normally fully navigable with the ⇥, ↑ and ↓ keys, so that you can enter tags quickly if you have a lot of them. You can create new tag-set or new tags with the ⏎ key, and delete them either with the ⌦ key or by pressing the ⊖ button.

When you are finished, press the OK button.

4.1.2. Assigning tags to data items.

Tags are set via the selection tool, presented above (chapter 2.7 page 27). Once you have some spots and links in the selection, you can assign a tag to it via the Edit ▸ Tags ▸ The tag set name ▸ The tag label menu. The menu content will be updated with the tag-sets and tags you defined in the tag-set dialog, described above (Figure 23). This will work in any Mastodon views, BDV or TrackScheme .

TrackScheme ships a second way to set tags quickly from the keyboard. After selecting the spots and links of interest, press the Y key. A floating menu should appear on the left part of the view

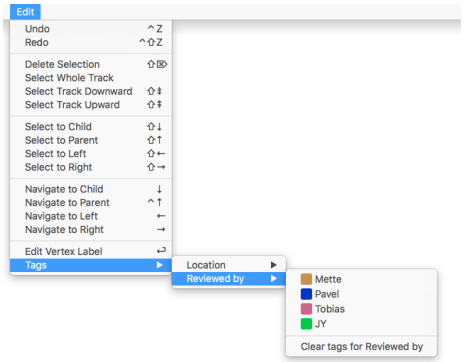


Figure 23.: The menu item to assign tag-sets and tags to the current selection.

Figure 24.: Assigning tags in TrackScheme . After pressing the Y key this floating menu is shown, that can be navigated with the digit keys.

panel (Figure 24). There is a bit of naming clash in the floating menu. Here, tag-sets are called *tags* and tags are called *labels*. Select the desired tag-set with the 1, 2, .. keys. The menu now shows the tags defined within this tag-set, that you can select the same way. Note that there is a way to remove all the tags over all the tag-sets on the selection by pressing ⇧ + ⌫ on the first menu, or just the tags of the selected tag-set by pressing 0 on the second menu.

4.1.3. Coloring views by tag-sets.

The tags we just defined and assigned can be used in with the views, to highlight the items that are tagged. In the View ⟩ Coloring menu of any view in Mastodon, you will find a sub-menu updated with the tag-sets you created among other choices (Figure 25). By default, newly created views are colored with the None coloring mode, which simply colors all the spots and links the same way, taking colors from render settings. If you select a mode corresponding to a tag-set, tagged spots and links will appear painted with the color you chose for the tags of this tag-set (Figure 26). This is very handy to mark some locations in the image or highlight interesting tracks in the data. Later we will see that tags can be used to retrieve specific items for further processing. Finally, in TrackScheme there is an option to show a legend of the current coloring mode. You can toggle it on or off and set the location of this legend in the View ⟩ Colorbar menu.

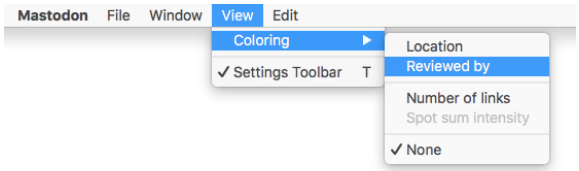


Figure 25.: The coloring menu, updated with the tag-sets.

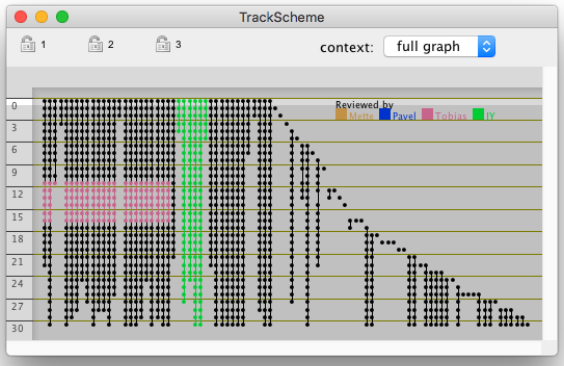


Figure 26.: Coloring with tag-sets in TrackScheme .

4.2. Numerical features.

Numerical features are values that are calculated from the data. For instance the mean intensity within a spot, or the displacement along a link. They are very generic: the main restrictions is that there must be a data item (a spot or a link) per feature value. But the feature itself can be scalar, non-scalar, real, integer, a string, a vector, *etc* . They are *labile*. Because they are defined for a data item, they will become invalid as soon as the data item changes. Think of what happens to the spot mean intensity if the spot is moved over the image for instance. Because we want to accommodate extensibility and large data, we have to use a special system that we describe below.

4.2.1. Feature computation.

Numerical feature values are calculated by **feature computers**. Feature computers are actually specialized Mastodon plugins, made so that it is easy for a 3rd party (you) to implement their own features in Mastodon. We explain you to write your own feature computer in the second part of this manual, dedicated to technical information.

Because feature computation can take very long on large images, you have to trigger it manually. On Mastodon main window, you can find a button `compute features` button. Pressing it will show the feature computation dialog (Figure 27). The feature computers are listed on the left panel. Clicking on the computer name displays some information about the feature they compute in the right panel. Note that they are named 'features' on this panel, but they are in reality the feature computers. For instance if you click on the `Spot gaussian-filtered intensity`, you will see in the information panel that this computer generates a feature for the mean intensity (weighted by a gaussian) and its

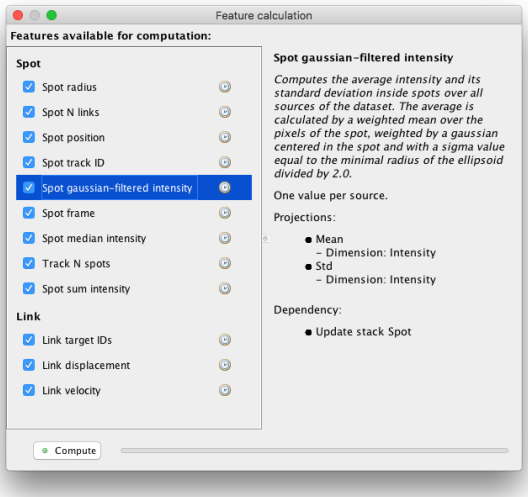


Figure 27.: The feature computation dialog.

standard deviation within a spot. Note also that they can have dependencies. For instance, the `Link velocity` feature computer depends on the `Link displacement` feature to be present at the time of computation.

The check-box on the left of each feature computer name triggers whether they will be part of the next feature computation. Press the Compute button to trigger computation of features.

Once the computation of all the features is complete, all the small clock icons that were shown right to the feature computer names now turned to a green dot . This is how we keep track of the validity of the feature values. Since the feature computation is triggered manually, and that a feature value might invalidated if the data changes (new spots added, removed, moved, changed the radius, added or removed some links), this icon serves as a signal for feature value de-synchronization. If the icon is shows as a clock , it means that the data changed since the last feature computation, and that the feature values are out of sync. If is shows a green dot , then the data did not change since last computation, and the feature values are sure to be valid. This is very important for proper interpretation of the data, and you will have to show the computation dialog often just to check the feature values validity. By the way, you can check now how the validity flag works. While keeping the feature computation dialog open, move a spot in a BDV view. You should see that all the green dot icons now turn to the clock icon. Also, if you now deselect some feature computers before launching a new computation, the validity flag will not turn to the green dot icon for those feature computers.

You probably have noticed, thanks to the progress bar, that the intensity-related features are the ones that take the most time to compute. Indeed, they require loading all of the data blocks on which there are spots. This could rapidly become cumbersome for large datasets, if you have to recompute all of the features every-time a spot is added or edited. Fortunately we implemented an update mechanism for feature computation. After the first computation, which possible takes very long, only the spots that have been added or modified since the last computation are considered for computation.

Now that we have feature values computed, we would like to inspect them and export them for

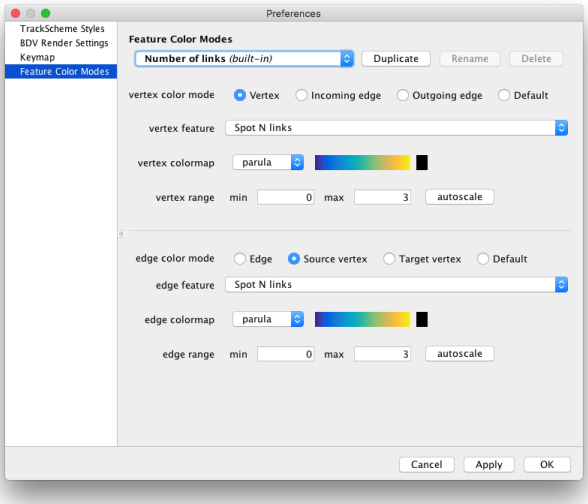


Figure 28.: The feature color mode configuration preference dialog.

further analysis. This is the role of the table view, but before getting to it, we will make a little detour to showing how to use features to generate coloring, accelerating updates of computation and saving them to disk.

4.2.2. Coloring views by numerical features.

We have seen above that tag-sets could be used to generate coloring of the data items shown in a view. Indeed, in the View » Coloring menu of each view, that tag-sets are listed and when selected, are used to assign a color to each data item. We can do something similar with feature values, except that feature based coloring requires more input from us.

Feature color modes need to be created first, and this is done in a dedicated user interface. Select the File » Preferences menu item in the main window or any view. The preferences dialog is shown. It is organized with a side-bar on the left that contains the various items that can be configured in Mastodon. Parenthetically, you can see that you can configure the display style of the TrackScheme and BDV views, and the keymaps. But we will see this later. In the sidebar select Feature Color Modes. The panel on the right now display the feature color mode configuration panel (Figure 28).

Its top line has a drop-down list that contains all the color modes already defined. Right now, there is only one, called **Number of links**. As the ***built-in*** suffix indicates, it is a built-in color modes, and it cannot be edited. To create a new one you must duplicate it and rename the new one. Do so by clicking on the Duplicate button, then on Rename. Let's create a color mode that color spots and links based on the mean intensity inside the spots, that we would call **Mean intensity**.

The rest of the configuration panel is made of two parts. The top part configures the vertex coloring, or how we color spots. The bottom part configures the edge coloring, or how we color links. In Mastodon the data is organized in a mathematical graph[8], in which the vertices are the spots, and the edges are the links that connect spots from one frame to another, so you will sometimes find in

[8]Mathematical graphs on Wikipedia https://en.wikipedia.org/wiki/Graph_(discrete_mathematics)

Mastodon and in this manual the vocables vertex and edge to design a spot and a link respectively. The vertex color mode specifies where do we take colors from. You can choose between:

- `Vertex`, which means a spot will take its color from a feature value it owns.
- `Incoming edge`, which means a spot will take its color from a feature value owned by the single incoming link that targets this spot. This is the link backward in time. If there are no such links or more than one, then the default color is used.
- `Outgoing edge` is the same thing, but for the link forward in time.
- `Default` mode does not rely on feature values but simply uses the default color in the view.

Depending on the mode you chose, the content of the drop-down list below, called `Vertex feature`, will change to reflect either the list of spot features or the link features. Select **Vertex** as a color mode and **Spot gaussian-filtered intensity** as feature. Two new drop-down list appear on the right of the feature list. One contains the list of projections in the feature and the second one contains the list of channels in the dataset.

We need to explain a bit what are **feature projections**. We said above that a feature could be roughly anything numerical, and was not necessarily a scalar. It could be a vector, a tensor, a complex number, *etc* . However to be usable and useful in Mastodon, features are required to expose a sensible list of projections that compose them. Feature projections are scalar and real values that can decompose or project a feature on a real axis. How they are defined is up to the person that created the feature computer, but we can rely on the *hope* that they choose wisely. For instance, a feature that gives the velocity vector of a link will reasonably expose 3 projections, one for each of the X, Y and Z component of the vector. Or maybe the polar angle, azimuthal angle and norm of this vector. Or maybe the 6 projections since they can be calculated on the fly. A complex feature value will reasonably expose 2 projections, one for the real part, one of the imaginary part. *Etc.* The `Spot gaussian-filtered intensity` feature has two projections, one for the mean value of the intensity within a spot and the standard deviation of this intensity. Since both can be computed on any of the channel present in the dataset, their number is multiplied by the number of channels. The configuration panel changes according to the number of projections in a feature and its multiplicity (Figure 29). For features that are made of one real value with no multiplicity, the projection list is superfluous and not shown. In our case, we simply want the mean of the only channel in the dataset.

Coming back to the color mode configuration, next we need to pick a color-map. A color-map acts as the LUT for an image, and maps a color to a certain value. Mastodon ships about 20 of them, many taken from the MatPlotLib project[9]. Finally, you have to specify a min value and a max value that will act as the brightness and contrast values for an image. Values below the min you defined will all be displayed with the first color of the color-map and values larger than the max with last. The black square you see next to the graded representation of the color-map is the color used for data items for which the feature value is not present or undefined (division by zero, *etc*). The `autoscale` button computes the min and max automatically from the feature currently selected with the values taken from the last feature computation.

[9] https://matplotlib.org/3.1.1/gallery/color/colormap_reference.html

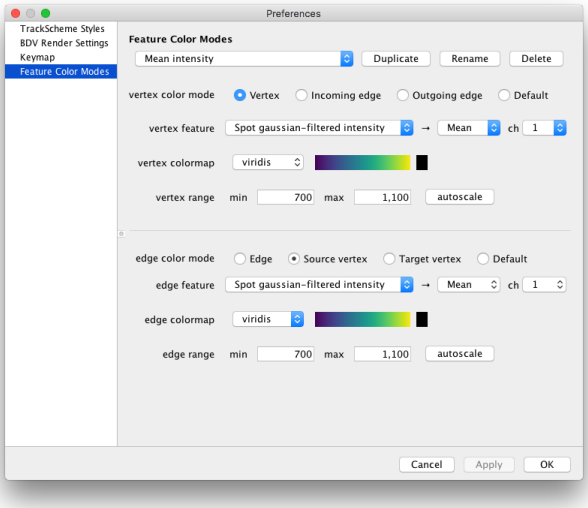


Figure 29.: Our custom feature color mode, based on spot intensity.

The edge coloring works exactly the same, except for the edge color mode.

- Edge means that a link will be colored by a feature value it owns.
- `Source vertex` will take a feature value from the source spot of this link, that is, the first in time.
- `Target vertex` does the same but for the last spot in time of this link.
- `Default` mode does not rely on feature values but simply uses the default color in the view.

To build our example feature color mode, choose `Source vertex` as color mode, and use for instance the `viridis` color-map along with 700 and 1100 for min and max values. Do the same for the vertex color mode (Figure 29).

Like for tag-sets, the View > Coloring menu is now updated with items corresponding to the color modes we created. If they are grayed-out, it means that the feature values they depend on is not yet computed. This kind of view immediately reveals important aspect of the data, even at a very high level. For instance with our custom color mode, we can quickly find cells that are the brightest, and visually inspect how the intensity in cells change over time (Figure 30).

4.3. The data table views.

4.3.1. The main table view.

The BDV and TrackScheme views are not suitable to display all the feature values we computed. The coloring we have been using with them is good only for visualization purpose. There is a nice view to properly inspect and exploit feature values in subsequent steps in your analysis: the table view. In practice, the table view is simply a tabular representation of the data items in Mastodon. Spots and links are displayed in a list where a single row corresponds to a data item, and columns to feature values and tags. You can create a new table view by using the menu Window > New data table. If you

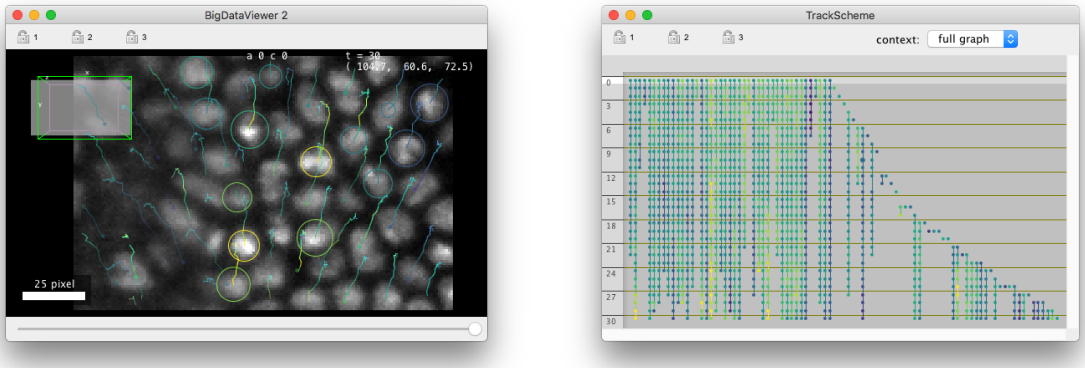


Figure 30.: Feature color mode in a BDV view and in a TrackScheme view.

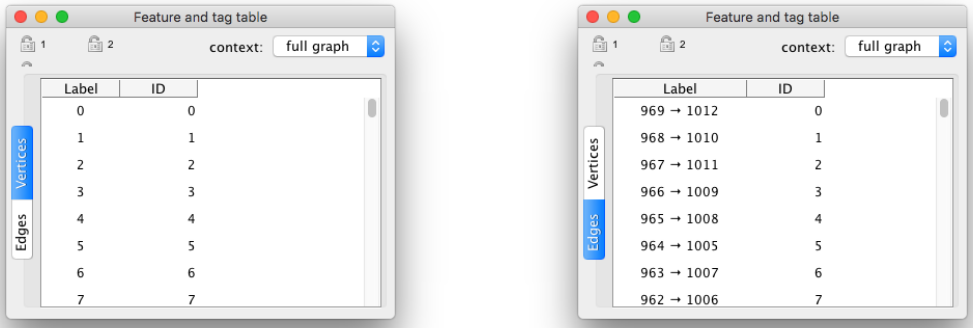
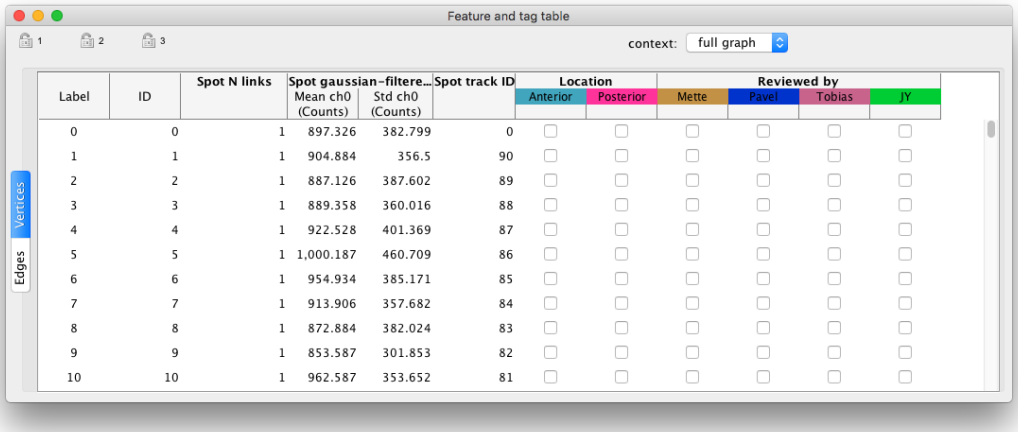


Figure 31.: The table view, with no features computed and no tags defined. Left: the table for spots (`Vertices` pane). Right: the table for links (Edges pane).

did not compute features and did not define and tag-set, it should look like the table in figure 31. The view is made of two tables, one for spots (`Vertices` pane) and one for links (Edges pane). Right now it is pretty empty. The spot and link tables only show the label and ID of the spots. Navigating in this table is done classically: the arrow keys ↑ and ↓ jump from one row to the next, and ← and → from one column to the next. Page↑ and Page↓ jump page per page. ctrl + Page↑ and ctrl + Page↓ alternate between the spot table and the link table.

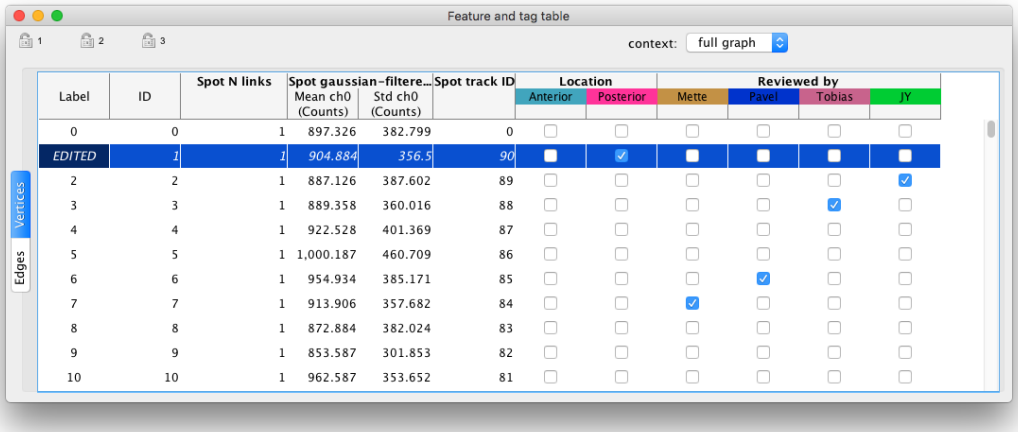
After computing some features and defining some tag-sets, the table show new columns (Figure 32). Note that the column headers represent the feature with their projection and physical units on several rows. For instance, the `Spot gaussian-filtered intensity` feature name is displayed on the first row of the column header. The header is split in two columns on the second row, one for each projection included in the feature (mean and standard deviation of a single channel). And in the third and last row, the units of each projection is display in brackets (Counts in this case). The header of the tag-set columns are similar. The first row shows the name of the tag-set, and the second row shows each of the tag the set contains, with the tag chosen color as background.

The table view can be used to edit in part the data (Figure 33). For instance you can edit the spot label directly in the table. Just navigate to the row of spot you want to change the name of and the `Label` column, then press F2. The label field becomes editable. When you are done editing, press



Label	ID	Spot N links	Spot gaussian-filtere... Mean ch0 (Counts)	Spot gaussian-filtere... Std ch0 (Counts)	Spot track ID	Location: Anterior	Location: Posterior	Reviewed by: Mette	Reviewed by: Pavel	Reviewed by: Tobias	Reviewed by: JY
0	0	1	897.326	382.799	0	☐	☐	☐	☐	☐	☐
1	1	1	904.884	356.5	90	☐	☐	☐	☐	☐	☐
2	2	1	887.126	387.602	89	☐	☐	☐	☐	☐	☐
3	3	1	889.358	360.016	88	☐	☐	☐	☐	☐	☐
4	4	1	922.528	401.369	87	☐	☐	☐	☐	☐	☐
5	5	1	1,000.187	460.709	86	☐	☐	☐	☐	☐	☐
6	6	1	954.934	385.171	85	☐	☐	☐	☐	☐	☐
7	7	1	913.906	357.682	84	☐	☐	☐	☐	☐	☐
8	8	1	872.884	382.024	83	☐	☐	☐	☐	☐	☐
9	9	1	853.587	301.853	82	☐	☐	☐	☐	☐	☐
10	10	1	962.587	353.652	81	☐	☐	☐	☐	☐	☐

Figure 32.: The table view, with features and tags.



Label	ID	Spot N links	Spot gaussian-filtere... Mean ch0 (Counts)	Spot gaussian-filtere... Std ch0 (Counts)	Spot track ID	Location: Anterior	Location: Posterior	Reviewed by: Mette	Reviewed by: Pavel	Reviewed by: Tobias	Reviewed by: JY
0	0	1	897.326	382.799	0	☐	☐	☐	☐	☐	☐
EDITED	*1*	*1*	*904.884*	*356.5*	*90*	☐	☑	☐	☐	☐	☐
2	2	1	887.126	387.602	89	☐	☐	☐	☐	☐	☑
3	3	1	889.358	360.016	88	☐	☐	☐	☐	☑	☐
4	4	1	922.528	401.369	87	☐	☐	☐	☐	☐	☐
5	5	1	1,000.187	460.709	86	☐	☐	☐	☐	☐	☐
6	6	1	954.934	385.171	85	☐	☐	☐	☑	☐	☐
7	7	1	913.906	357.682	84	☐	☐	☑	☐	☐	☐
8	8	1	872.884	382.024	83	☐	☐	☐	☐	☐	☐
9	9	1	853.587	301.853	82	☐	☐	☐	☐	☐	☐
10	10	1	962.587	353.652	81	☐	☐	☐	☐	☐	☐

Figure 33.: After editing spot labels and tags.

⏎. The tags are displayed as check-boxes in the table, that you can set directly by clicking on them. Or you can navigate the desired row and column and set them with the Space key.

The highlight and selection are also shared with table views. When the table view is not active, selected items are shown with a gray background. The highlight spot or link is shown in the table with a thick black border (Figure 34). To add rows to the selection, the default key-bindings are again standard. Press ⇧ + Left-click to add a range of rows to the selection from a rable view, or use ⇧ + ↑ or ⇧ + ↓ or ⇧ + Page↑ and ⇧ + Page↓. By pressing ctrl + Left-click or ⌘ + Left-click you can toggle single rows in and out of the selection. Of course, all of the commands related to the selection we have seen before also apply to the table views (Table 4 page 28).

Another feature of data tables is that they can be made slave of a spatial context, like for Track-Scheme . When another BDV view is active, you can select its name in the drop-down list on the top-right part of the table. Then the table only shows the data items that are currently displayed in the master BDV view. The notion of spatial context is explained above (section 3.5 page 34). See also the figure 21 page 34.

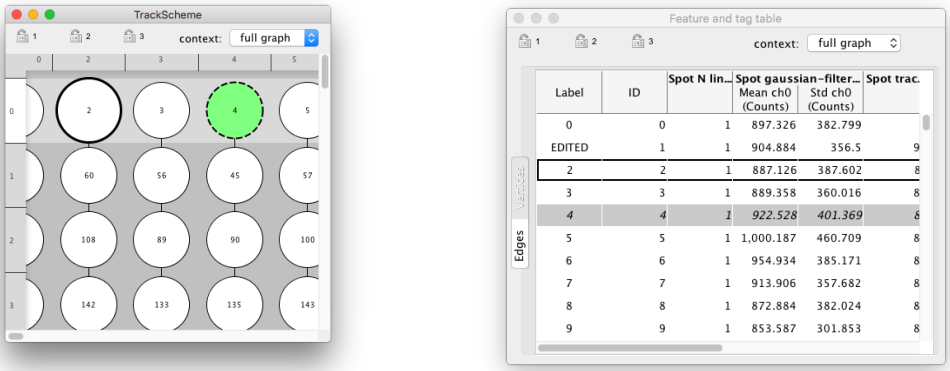


Figure 34.: Highlight and selection in the table view.

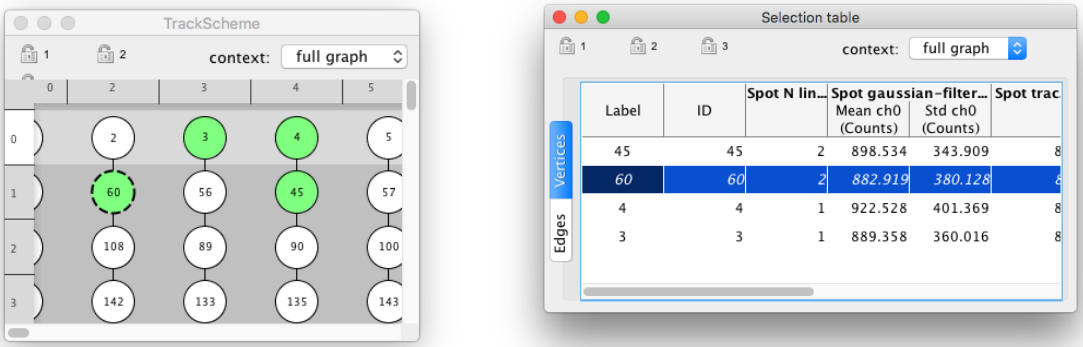


Figure 35.: The selection table.

4.3.2. Sorting rows.

The table can be sorted by clicking on the header of the column you want to use for sorting. It works for labels, IDs, feature values and tags

4.3.3. The selection table.

There exists a variation of the table view, but that display only what is currently in the selection. To display such a table, go to Window > New selection table in the menu. The selection table that appears only shows what is in the selection, and is constantly updated to reflect changes in the selection (Figure 35). You cannot use it to edit the selection like in the main table. However the row you pick in this table will set the focus and highlight in other views. Everything else applies to the selection table.

4.3.4. Feature-based coloring in table views.

Of course, feature based coloring works with the table views (Figure 36). And it can give a pleasant display when combined with sorting rows by a feature column.

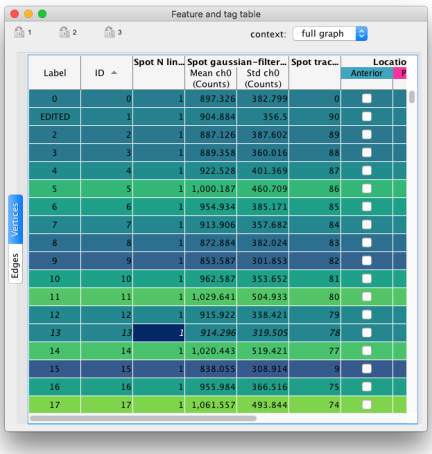


Figure 36.: Feature-based coloring in table views.

4.3.5. Exporting table data.

The data currently displayed in a table view can be exported to CSV. When a table window is active, select the menu item File ⟩ Export to CSV. You will have to specify a saving location and a name. But two CSV files will be produced: one for the spot table (appended with `-vertices.csv`) and one for the link table (appended with `-edges.csv`) Only the data currently displayed in the view are saved, and ordered as in the view. This means that if you call the File ⟩ Export to CSV command from a selection table, only the current selection will be saved.

Table 5.: Default navigation key-bindings for Mastodon-table views.

Action	Key
Navigation.	
Move from row to row	↑ and ↑.
Move column to column	← and →.
Display the spot table / the link table	ctrl + Page↓ and ctrl + Page↑.
Editing.	
Edit spot label	F2 when focus is in the label column.
Toggle tag	Space when focus is in the desired tag column.
Selecting.	
Add next / previous row to selection	⇧ + ↑ and ⇧ + ↓
Add range to selection	⇧ + Page↑ and ⇧ + Page↓ or ⇧ + Left-click
Toggle row into selection	ctrl + Left-click or ⌘ + Left-click

5. Semi-automated tracking.

Let us suppose we are dealing with a difficult image in which me must track only a subset of cells. The cells are difficult to track automatically, for instance because there are many spurious structures labelled. Or because there are other cells of different sizes in the tissue we are studying. Or because the image quality varies in time and space. The data is such that that fully automated algorithms introduced in chapter 1 won't give us fully accurate results. We can use the manual editing tools introduced in chapter 2 and curate the results of automated tracking, manually removing spurious detections and links, and fixing incorrect ones. Another approach would be to start from a blank annotation and track manually only the cells we are interested in. Both approaches might be long and tedious. We introduce in this chapter tools for semi-automated tracking, that should alleviate the work of the second approach.

Semi-automated tracking is simply a way of following a specific cell that you picked-up manually. The tracker will follow the cell over time, and create spots and links for a certain amount of time-points. It searches for the best spot candidate in the next time-point around the location of the spot. It then creates a new spot there and links it to the previous one. This procedure is repeated this for a certain number of time-points you can set, creating or augmenting a track starting from the spot you selected.

The semi-auto tracker can be configured to work backward in time (backtracking), to have a certain search radius, or a certain sensitivity to spot quality (as defined in chapter 1.4). The way it interacts with existing annotation can be configured too. You can make it stops when it meets an existing spot, linking to it or not. You can make it connect to small tracks and resume tracking when it meets the track end. You can force it to only create links on already existing spots. These configuration options give rise to several use-cases we will also survey in this chapter. But the important message is that semi-automated tracking is a convenient means for dealing with difficult cases, when a fully automated approach fails, and when the data to track is large that doing it manually is inconvenient. Or when you only care for a subset of cells in a dense tissue.

5.1. Simple semi-automated tracking.

We will introduce semi-automated tracking on a movie with empty annotation. Open the dataset we have been using so far, and clear all annotations (for instance select all ctrl + A then delete selection ⇧ + ⌫). Then pick a cell in the top layer, and create a spot, centered on its brightest part, like for instance on Figure 37. Adjust its radius and select it. Open a TrackScheme window to visualize the tracking progress.

To start semi-automated tracking, press ctrl + T. A log window should open, and tracking should proceed. If the log does not complain about candidates being too far, you should end up with something resembling Figure 38. The tracking stopped after 10 frames, and the last spot added is now in the selection. Each spot is centered on the cell we started with, and it has the same radius that of the first spot we created. You can resume semi-automated tracking from the last spot created by just pressing ctrl + T again. If you do it one more time you should reach the end of the movie, with a new track following a single cell over the full movie duration. To get it, you just had to press ctrl + T

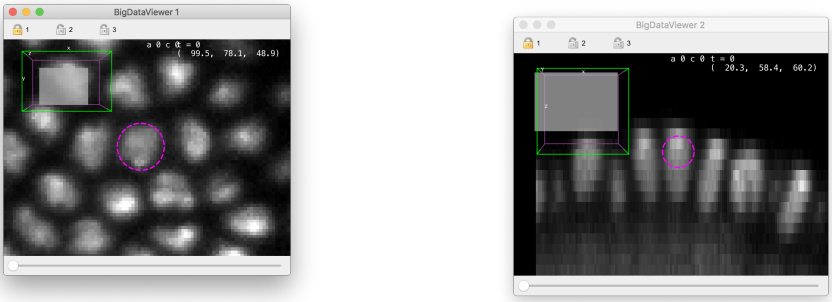


Figure 37.: Manually picking a cell for semi-automated tracking.

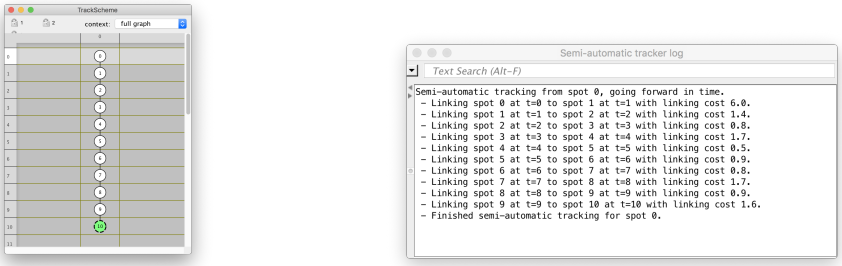


Figure 38.: Results of semi-automated tracking.

3 times. If you have more than one cell in the selection, they will be tracked one after another.

5.2. Configuring the semi-automated tracker.

The tracking does not always succeed. Depending on where you position the initial spot, it might fail, for instance stating in the log that if found a suitable spot, but outside the tolerance radius. (In this example movie, this happens often because the cells are very elongated along the Z direction, and the tracker tends to find candidates near the top, brightest part of the cell.) The parameters that control for instance the tracker search radius can be set in the semi-automated tracking configuration dialog, in the Plugins » Tracking » Configure semi-automatic tracker. The window show in Figure 39 should appear. This dialog is very similar to the one used to configure feature color modes, that we have seen in chap-

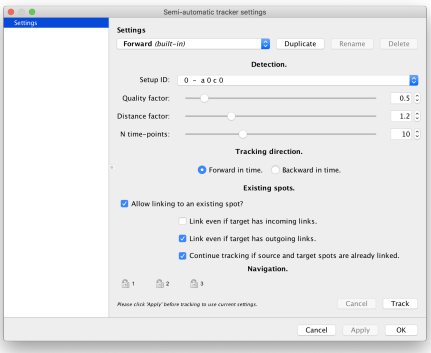


Figure 39.: The semi-automated tracking configuration panel.

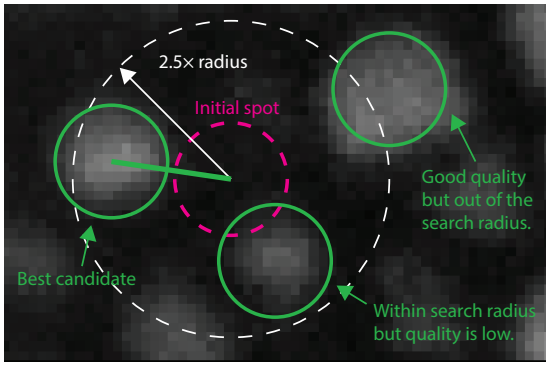


Figure 40.: Illustration of the semi-automated tracking process.

ter 4.2.2 page 40. It also works the same way: the top elements are made to manage several tracking configuration, that will be stored on disk and retrieved in your next Mastodon session. There are two default tracking configuration: the **Forward** one is the default that we just used. The **Backtracking** configuration tracks backward in time. The other parameters controls the tracker behavior for the configuration currently selected in the top drop-down list. To explain what they do we need first to describe how the semi-automated tracker works (Figure 40).

The semi-automated tracker works by processing only a small neighborhood around the initial spot, (called the *source* spot later). This neighborhood is centered on the spot center (in magenta in Figure 40), but taken in the next time-point (or previous one if you choose to go backward in time). It applies the DoG detector (described in chapter 1.4 page 11) on this neighborhood, which yields several detections (green circles). The detections that are found outside of a search radius (white, dashed circle) are not considered. Detections inside the search radius but with a low quality are discarded as well. The tracker therefore select the detection with a sufficiently large quality inside the search radius. If there is more than one suitable detection, it selects the one with the highest quality. A new spot is created at this location, with the same radius that of the source spot. If the source spot is not a sphere but an ellipsoid, the smallest radius of the ellipsoid is taken. The newly created spot is then linked to the source spot. This is then repeated for the next time-point (or previous one), using the new spot as initial spot in the same process. If no suitable detections are found within the search radius, the tracker stops, and the reason is printed in the tracker log window.

The configuration panel controls the parameters of this process.

- `Setup ID` specifies what channel (or setup in case you have a multi-view dataset) that will be used for the detection.
- `Quality factor` specifies the threshold on quality below which we reject detections. This threshold is expressed in fraction of the source spot quality. For instance if the source spot quality is 60 and the `Quality factor` is 0.5, detections with a quality lower than 30 will rejected. If the source spot has no quality value (it was added manually), this parameter is ignored and all quality values are accepted.
- `Distance factor` specifies the search radius, in units of the source spot radius. For instance, for a value of 2.5, only detections that are within 2.5 × the radius of the source spot will be considered. Again, if the source spot is not a sphere but an ellipsoid, the smallest radius of the

ellipsoid is taken.

- `N time-points` specifies the number of time-points after which to stop.
- `Tracking direction` lets you specify whether you want tracking to happen forward or backward in time.

If you see that the tracker often stops with a message stating that it could not find a suitable candidate within search radius, try to either decrease the value of the `Quality factor` or increase the value of the `Distance factor`.

5.3. Tracker behavior with existing annotations.

The next parameters below the **Existing spots** category configure how the tracker deals with existing annotations. They change the behavior described in the previous section. Indeed, before running the DoG detector on the neighborhood, the tracker first searches for an existing spot within the search radius. If it finds one, it does not run the DoG detection, but links to the existing spot (called *target* spot later) or not, depending on the following parameters.

If the `Allow linking to an existing spot` checkbox is deselected, the tracker stops. If it is selected, the tracker will link to the target spot, provided it has no links already. However, the next 3 parameters allow to add exceptions:

- `Link even if target has incoming links`
- `Link even if target has outgoing links`
- `Continue tracking if source and target spots are already linked`

Their selection have important consequences when you are tracking along existing spots. They are best exemplified by several different use-cases.

5.4. Main use-cases for semi-automated tracking.

5.4.1. Tracking a subset of cells.

This is what we have been doing in the first section of this chapter. This does not require any special configuration and the default tracking configuration called **Forward** will do. Here is an example of when this use-case can be useful.

Arianne Bercowski Rama and Laurel Ann Rohde (Segmentation Timing and Dynamics Laboratory, EPFL) are studying the somatogenesis dynamics in the zebrafish embryo. They acquire long-term time-lapse movies of the development of an embryo, with cellular resolution. For this project they wanted to follow a few cells of interest (a few dozens per movie) and investigate the expression of a gene reporter as the cells moved along the zebrafish embryo. The nuclei are all stained with a nuclear marker, so this fluorescence channel could be used for automated detection, but there are thousands of cells. Instead, they relied on semi-automated tracking, selecting a the cells of interest in the first frames of the movie, and tracking them to the end thanks to the semi-automated tracker (Figure 41,

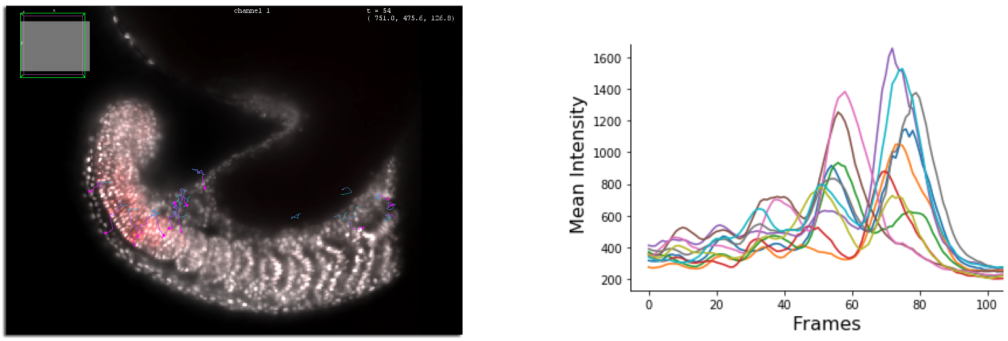


Figure 41.: Semi-automated tracking of a subset of cells during somatogenesis in the zebrafish embryo. **Left**: Image data overlaid with the tracks of the cells of interest. **Right**: Gene reporter fluorescence intensity for 10 of these cells as a function of time.

Figure 42.: Small discontinuous track segments following a single cell.

left). Once the tracking was done, they used feature analysis to extract fluorescence intensity over time for each cell in the gene reporter channel (Figure 41, right).

5.4.2. Stitching small track segments.

The semi-automated tracker can be used with the same configuration to stitch small track segments, which follow the same cell but are separated and spread in time. This might be caused linking with missing detections, and the situation could resemble what is pictured in Figure 43.

We can use the semi-automated tracker to connect them, and add the missing detections. We want the tracker to create spots for the cell in time-points when they are missing, and to connect to existing ones it will find by following the cell. To allow this, we must enable first the 'Link even if target has outgoing links' setting, so that the tracker does not stop when it meets a spot that is at the beginning of a track segment. Second, we must enable the 'Continue tracking if source and target spots are already linked' setting. If we do not, the tracker will stop after connecting the first track segment it meets. Finally, we have to increase the value of the 'N time-points' parameter, so that the tracker iterate through the full movie. After doing this and tracking for the first cell, we get a single track (Figure 42).

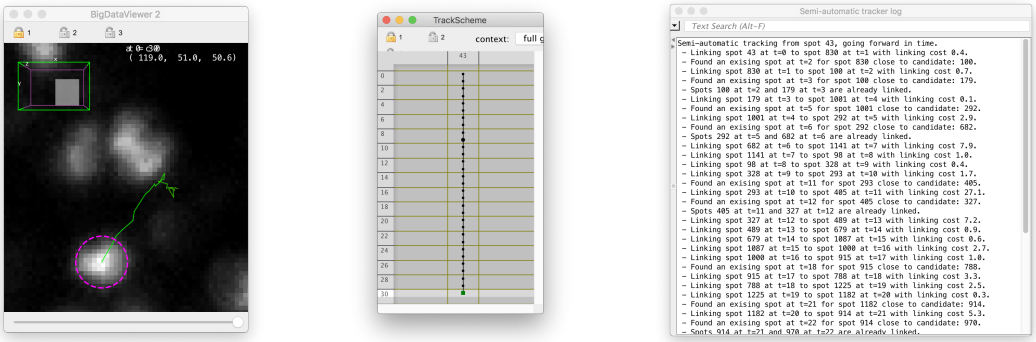


Figure 43.: Results of stitching the track segments with the semi-automated tracker. Notice the messages on the log window that state what the tracker did when it meets existing spots.

5.4.3. Backtracking, branching on cell divisions.

Backtracking is particularly useful when creating lineages of cells that divide often in the movie. Another application is in mapping differentiated cells at a late time-point, to the position of their progenitors at the beginning of the movie. For this we typically starts from an empty annotation, select a cell of interest in a late time-point, and backtrack it to its position in the beginning of the movie. The built-in configuration called **Backtracking** is made for this. If the cell we backtrack divided several times during the movie, might be creating tracks for sibling cells. We want these tracks to branch properly in case we meet the mother cell when it divides.

This is why we need to enable the '`Link even if target has outgoing links`' setting, but no other. Suppose we already have one track for a cell that divides, following the mother cell then one of the daughter cells. When we will backtrack from the other daughter cell, late in the movie, we will meet the division point of the mother cell. There is an existing spot there, just before the cell divides. It has already an outgoing link (to the first daughter we already tracked), and we want to connect to it. Hence we enable the '`Link even if target has outgoing links`' setting. Because we want to stop tracking there (the rest of the track is good already), we do not enable any of the two other settings.

5.4.4. Sparse linking over dense spots.

6. The selection creator.

The automated detection process we use often generates a lot of spurious detections. In TrackMate [5] we complemented it by adding *feature filters* just after the detection step. In TrackMate UI it takes the shape of filter windows, where the user can specify a feature and a threshold above or below which spots are rejected. The filters can be stacked to generate a more stringent filtering. This approach is like fishing with a small-hole net, then throwing back unwanted fishes to sea.

In Mastodon we take a somewhat different approach. We don't have a filter interface, but instead work with the selection tool. To remove spurious spots, they are added to the selection based on criteria you define, then the selection content is deleted. The tool to create a selection is called the selection creator and we describe it here. It works differently from the interactive selection we have been presenting before (see 2.7 page 27). Instead of manually clicking on spots or links, or drawing a selection rectangle in TrackScheme , you will enter an expression that will be parsed to generate a selection.

We thought this approach would be more convenient and powerful. First going through the selection allows to use the selection creator for other ends than filtering. You can create a selection and assign a tag to it for instance (see 4.1.2 page 36). Or use a selection as an input to the linking wizard (see 1.5.1 page 16). Or have a BDV view that only shows the selection you just created (see 3.2 page 33).

Part II.

Mastodon interoperability.

My mistress' eyes are nothing like the sun;
Coral is far more red than her lips' red;
If snow be white, why then her breasts are dun;
If hairs be wires, black wires grow on her head;
I have seen roses damasked, red and white,
But no such roses see I in her cheeks;
And in some pérfumes is there more delight
Than in the breath that from my mistress reeks.
I love to hear her speak, yet well I know
That music hath a far more pleasing sound.
I grant I never saw a goddess go;
My mistress, when she walks, treads on the ground.
And yet, by heaven, I think my love as rare
As any she belied with false compare.

William Shakespeare

Part III.
Extending Mastodon.

As palavras, senhor, estão por aí, no ar,
qualquer as pode aprender.

José Saramago

Part IV.
Technical information.

Frères humains qui après nous vivez
N'ayez les cœurs contre nous endurcis,
Car, se pitié de nous pauvres avez,
Dieu en aura plus tost de vous merciz.

Francois Villon

7. Mastodon numerical features.

This chapter describes how the feature values currently available in Mastodon are calculated. We also gives information about their dimension, units, *etc* .

7.1. Feature dimensions.

In Mastodon, feature values are expressed when possible in physical units. Each feature projection as a *dimension* (in the physics meaning) from which we compute the *units* of the values. For instance, if a feature value as the dimension `LENGTH` and the spatial units is µm, then the values of this feature will be in µm. Mastodon only has physical units for space. For time, the frame interval is always equal to the dummy `1 frame`. This is why you will find all units involving time expressed in `frames`. Each feature projection has a dimension, and the features report what are the dimension of their projections in the feature computation dialog (Figure 44). The Table 6 lists all the feature dimensions currently supported in Mastodon, and give examples of the derived units when the spatial units are µm.

7.2. Spot features.

7.2.1. Spot gaussian-filtered intensity.

This feature has two projections per channel, **Mean** and **Std**. The values are floating point numbers, with the dimension `Intensity`.

The **Mean** projections give the average intensity at the center of the spot. The average is calculated by taking the mean intensity inside the spot, weighted by a gaussian centered on the spot. The size of the gaussian is adapted to fit into the smallest radius of the spot. The Figure 45 illustrates the Gaussian weights in the average for several spots.

The **Std** projections give the standard deviation of these means. The standard deviation is also weighted by the Gaussian.

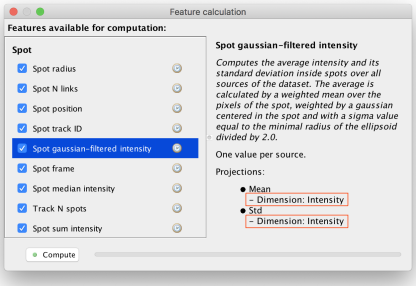


Figure 44.: The feature dimensions are listed in the feature computation dialog, in the description panel that appears on the right when you click on a feature name.

Dimension	Name	Example units (when spatial units are µm)	Description
NONE	None	ø	Used for dimensionless quantities, such as frame position, number of things, *etc*
LENGTH	Length	µm	For quantities about the length of objects. For instance radius or distance between objects.
POSITION	Position	µm	Dimension for feature that report a position. Different from LENGTH so that for objects with small lengths at large positions, quantities are plotted separately.
TIME	Time	frame	For quantities that report a delay, a duration or the timing of an event. Because Mastodon does not deal with physical units for time, quantities formed with the time dimension always use the frame unit.
VELOCITY	Velocity	µm / frame	For quantities that report a speed or a velocity.
RATE	Rate	/ frame	For quantities that report a change per units of time.
ANGLE	Angle	Radians	Measures of angles. In Mastodon, all angles are in **radians**.
STRING	NA	ø	For non-numeric features.
INTENSITY	Intensity	Counts	For quantities based on pixel values. For instance the mean intensity within a spot.
INTENSITY_SQUARED	Intensity²	Counts²	For quantities based on pixel intensity squared. For instance the variance of the mean within a spot.
QUALITY	Quality	ø	This dimension is used by spot detectors. There is a special feature called **Detection quality**, that stores for each spot they detect automatically a measure of quality or confidence in their detection. See section 1.4.
COST	Cost	ø	This dimension is used by spot linking algorithms. There is a special feature called **Link cost** used in the estimation phase. It stores for each link the cost that the linker computes for it in the estimation phase. These costs are then used in the association phase to retrieve the best set of links.

Table 6.: Feature dimensions in Mastodon.

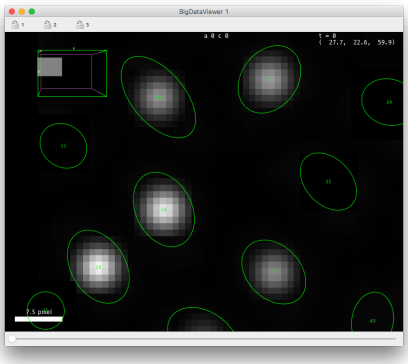


Figure 45.: Illustration of the Gaussian weights used in the **Spot gaussian-filtered intensity** feature.

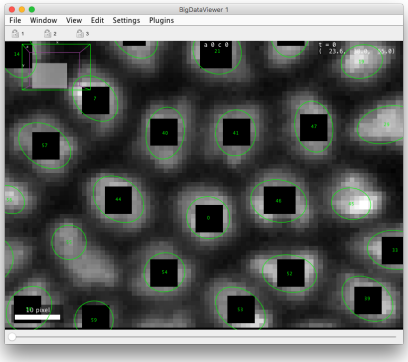


Figure 46.: Illustration of the box in which the **Spot median intensity** feature is calculated.

7.2.2. Spot median intensity.

There is one projection of this feature per channel. It reports the median intensity in the center of the spot for each channel. The median is calculated inside a box that fits in the smallest radius of the ellipsoid. The Figure 46 illustrates what this box looks like for several spots.

7.2.3. Other spot features.

Feature name	Projections	Description
Spot frame		The spot frame.
Spot N links		The total number of links, incoming and outgoing, of the spot.
Spot position	X, Y, Z	The spot center position, in physical units.
Spot radius		The spot radius in physical units. For spots that are ellipsoids, returns a radius using the geometric mean of the spot ellipsoid radiuses. This approximation is such that the sphere with the reported radius and the spot ellipsoid have the same volume.
Spot sum intensity	One value per channel	The total spot intensity for all the pixels inside the spot ellipsoid.
Spot track ID		The ID of the track the spot belongs to. Track IDs are positive integer numbers starting from 0.

7.3. Link features.

Feature name	Projections	Description
Link target IDs	Source spot id Target spot id	Stores the IDs of the two spots the link connects to. In Mastodon, the links are oriented: the source and target are not equivalent. By convention in Mastodon, the source spot is always the first in time, and the target the last in time.
Link displacement		The distance between the source and target spots of the links, in physical units.
Link velocity		The velocity at the time of the link. It is calculated as the link displacement divided by the frame interval between the source and target spots (in `frame` units).

7.4. Track features.

'Track' is the vocable we use in Mastodon for the weakly connected components of the graph. A track is made of all the links and spots that can be reached by jumping across links in any direction. In a lineage, a track corresponds to a single cell and all its daughters, grand-daughters, *etc* . Track feature are value that are defined for a whole track. An example would be the number of spots in a track. In Mastodon, there is no special place to store track feature values. Track feature values are stored in the spots of the tracks, and listed in spot features. By convention, their name starts with **Track** and spot features starts with **Spot**.

Feature name	Projections	Description
Track N spots		The number of spots in a track.

8. The graph data structure of Mastodon.

The *mastodon-graph* Java package can be used to implement directed graphs with small memory footprint. Vertices and edges are not stored as individual objects. Instead vertex and edge data is laid out in a primitive `byte[]` array and accessed via proxy objects. This chapter describes some internals of the *trackmate-graph* package.

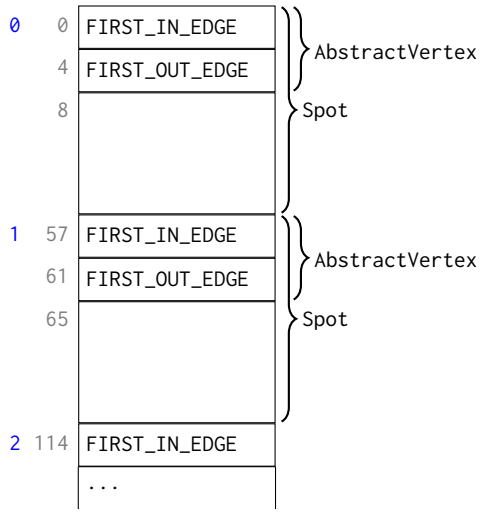
8.1. Memory layout.

Each vertex and each edge maps to a contiguous portion of a `byte[]` array. The size of a portion (*elements*) is fixed for a particular `AbstractVertex` or `AbstractEdge` subclass. Each *element* starts with a fixed part that represents the graph structure and then additional payload used by the subclass to describe some vertex attributes, etc.

There is one `byte[]` array that stores all vertices, and one `byte[]` array that stores all edges. References between these arrays are in the form of *element indices*. Whether these are indices refer to elements in the vertex or in the edge memory array is clear from the context.

8.1.1. Vertex layout.

The following diagram illustrates the layout of vertices in a `byte[]` array:



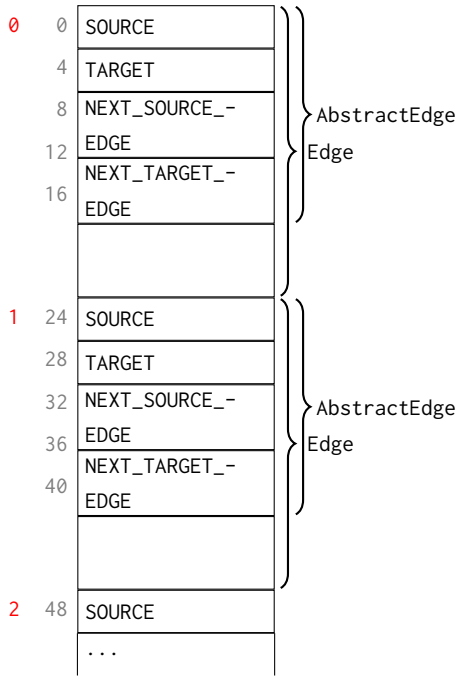
In the left-most column, *element indices* are shown (in blue), followed by byte indices (in grey). In the example, each element of the `AbstractVertex` subclass `Spot` requires 57 bytes to store. The data for the ith `Spot` starts at byte $i \cdot 57$. The fixed `AbstractVertex` part comprises element indices `FIRST_IN_EDGE` and `FIRST_OUT_EDGE`, occupying 4 bytes each. The remaining 49 bytes are `Spot` attributes.

`FIRST_IN_EDGE` is the element index (in the edge memory array) of the first *incoming* edge, i.e., an edge pointing to this vertex. The remaining incoming edges of the same vertex are stored as a linked list in the edge memory as described below. If this vertex does not have any incoming edges `FIRST_IN_EDGE` is -1.

Similarly, FIRST_OUT_EDGE is the element index of the first *outgoing* edge, i.e., an edge starting from this vertex. The remaining outgoing edges of the same vertex are stored as a linked list in the edge memory as described below. If this vertex does not have any outgoing edges FIRST_OUT_EDGE is -1.

8.1.2. Edge layout.

The following diagram illustrates the layout of edges in a byte[] array:

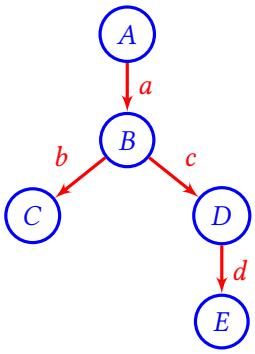


In the left-most column, *element indices* are shown (in red), followed by byte indices (in grey). In the example, each element of the AbstractEdge subclass Edge requires 24 bytes to store. The data for the ith Edge starts at byte $i \cdot 24$. The fixed AbstractEdge part comprises element indices SOURCE, TARGET, NEXT_SOURCE_EDGE, and NEXT_TARGET_EDGE, occupying 4 bytes each. The remaining 8 bytes are Edge attributes.

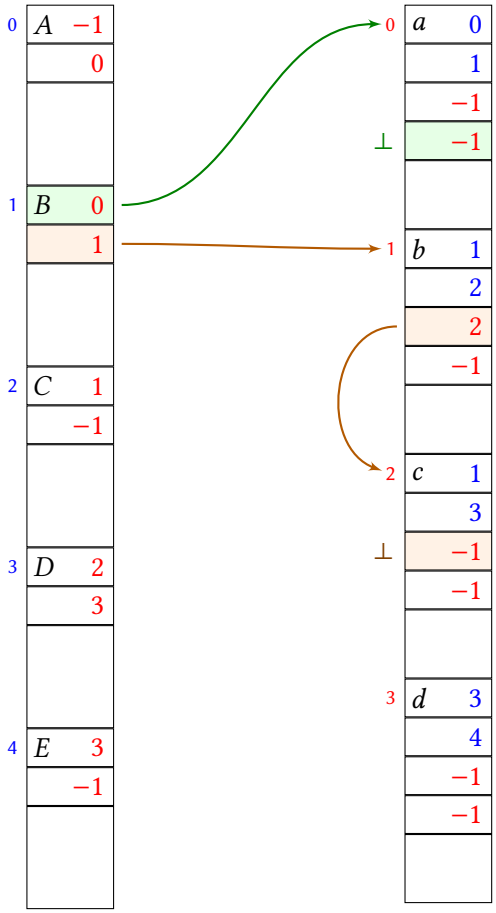
- SOURCE is the element index (in the vertex memory array) of the vertex from which this edge starts.
- TARGET is the element index (in the vertex memory array) of the vertex to which this edge points.
- NEXT_SOURCE_EDGE is the element index (in the edge memory array) of the next outgoing edge of the source vertex, *i.e.* , the next edge that has the same SOURCE. If there is no such edge then NEXT_SOURCE_EDGE is -1.
- NEXT_TARGET_EDGE is the element index (in the edge memory array) of the next incoming edge of the target vertex, *i.e.* , the next edge that has the same TARGET. If there is no such edge then NEXT_TARGET_EDGE is -1.

8.1.3. Example

Consider the following example graph comprising vertices A, B, C, D, E and edges a, b, c, d.



This is laid out in memory as follows

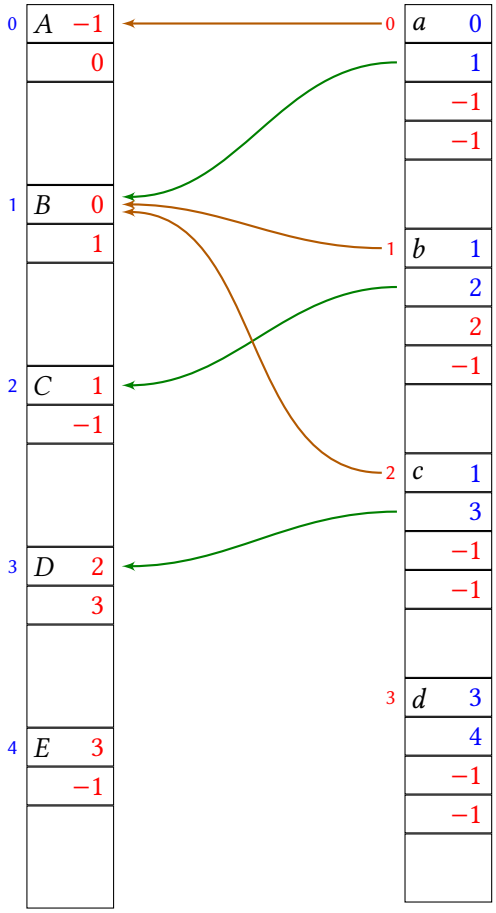


The links between vertex B and it's adjacent edges a, b, c have been highlighted. Let's look at that in more detail: Vertex B is stored at element index 1 in the vertex memory array. B has one incoming edge a. The edge a is stored at element index 0 in the edge memory array. Therefore the `FIRST_IN_EDGE` field of B is 0. Apart from a, the vertex B has no further incoming edges. Therefore, the `NEXT_TARGET_EDGE` field of a is -1, i.e., the list of edges entering B terminates here.

B has two outgoing edges b, c. The edge b is stored at element index 1 in the edge memory array. Therefore the `FIRST_OUT_EDGE` field of B is 1. The next outgoing edge of B is c which is stored at

element index 2. Therefore, the NEXT_SOURCE_EDGE field of b is 2. After c, the vertex B has no further outgoing edges. Therefore, the NEXT_SOURCE_EDGE field of c is -1, i.e., the list of edges leaving B terminates here.

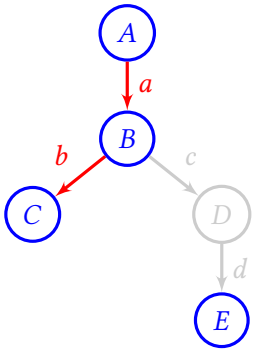
Below is the same memory layout again, this time highlighting the references from edges a, b, c back to the vertex memory array.



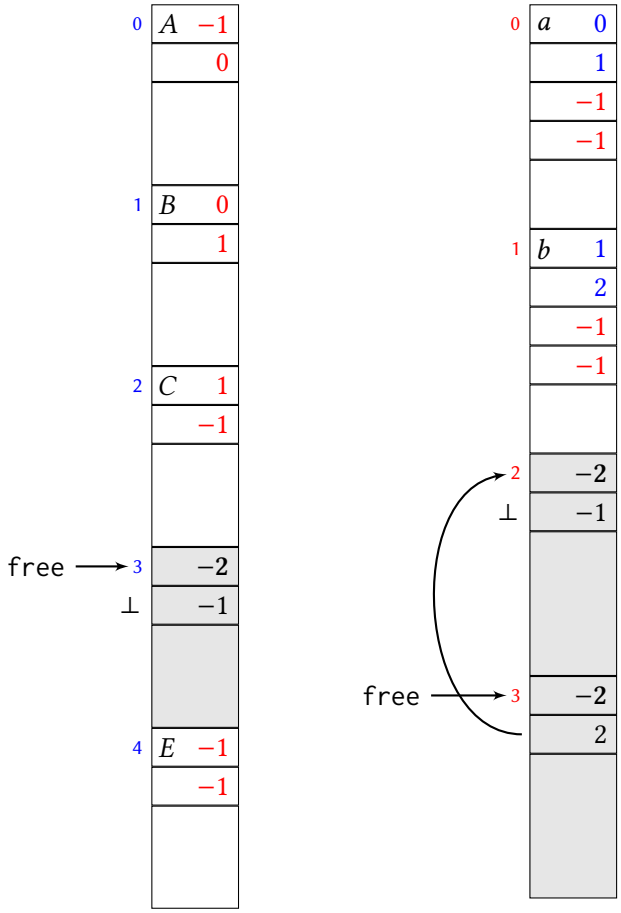
For example, edge c is leaving vertex B (index 1) and entering vertex D (index 3). Therefore the SOURCE field of c is 1, and the TARGET field of c is 3.

8.2. Free-list of unallocated elements.

The vertex and edge memory arrays can only ever grow. When elements are released, they are simply marked as free for re-use. Assume that in the above example vertex D is deleted, as well as it's adjacent edges c, d, leaving this:



After removing c, d, D the memory layout looks like this:

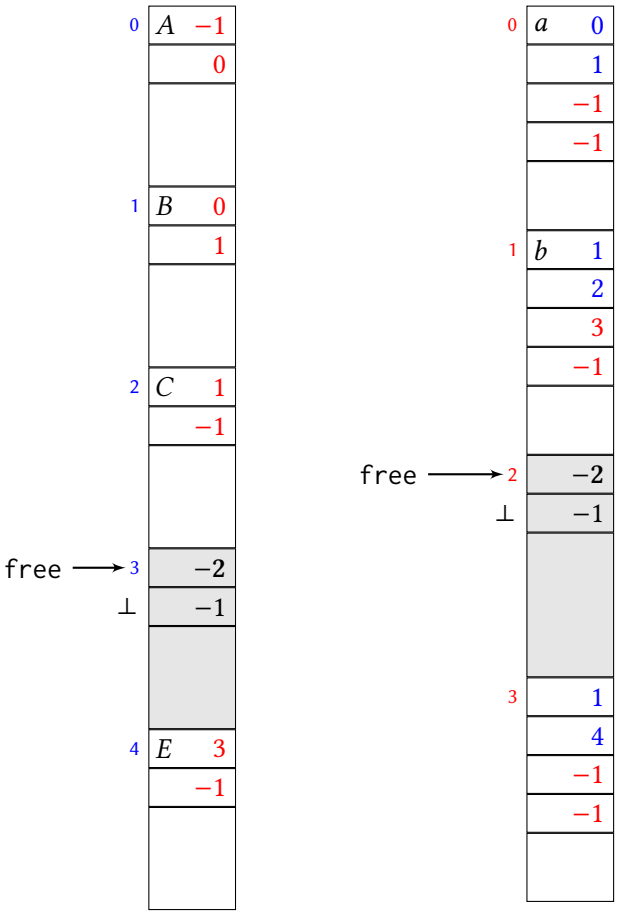


The element 3 in the vertex memory array as well as elements 2 and 3 in the edge memory array have been marked as free. This is done by putting the magic number "−2" into the first 4 bytes of the element.[10] The next 4 bytes of a freed element are the index of the next freed element in the (same) memory array, or −1 if there is no next freed element. Each memory array remembers the index `free` of the first freed element.

Newly freed elements are enqueued at `free`, that is, at the head of the free-list. So in the above example, edge element 2 was freed first, followed by edge element 3.

[10] In vertices and edges the first 4 bytes are always occupied by a (positive) index or a −1 index list terminator. Therefore, occupied and free blocks can not be confused.

The next edge element will be allocated at head of the free-list and the `free` index move to the next element. For example, assume that a new edge is created from B to E:



If `free` is -1, then no more freed elements are available and the underlying memory array has to grow to fit newly added elements.

9. Containment in Convex Polytopes using k-D trees.

Several important features of Mastodon rely on the fast retrieval of data items (spots and links) close to specific 3D positions. For instance we need to do so when you move the mouse close to the drawing of a spot in a view, to retrieve the spot in question. Or to determine what spots must to be painted in a BDV view, depending on the zoom level, position, rotation and field of view. There exists several well established algorithms and techniques to do that, in the case where the bounding-box in which we need to retrieve data items is a 3D rectangle aligned with the X, Y, Z axes of the dataset.

In our case it is not. In BDV you can rotate the view around an arbitrary angle. The XY view plane does not match an orthogonal plane of the dataset. Also the planes that make the bounds of the field of view are not aligned with these axes. In our case, the field of view is a 'Convex Polytope'. It is a portion of 3D space delimited by a set of planes[11]. Think of an ideal diamond. Each facet of the diamond would be one of the planes. The interior of the diamond would be the convex polytope. Our goal is to know what are the points that are inside this volume so that we can *e.g.* paint them without losing time painting the ones not in the field of view.

At the time of the development of Mastodon, there was no published algorithm for the fast retrieval of points in a convex polytope. Tobias derived such an algorithm in 2016, and it is detailed in this chapter. To the best of our knowledge this is unpublished.

9.1. Introduction.

The problem we want to solve is the following: Given a set of points $\mathbf{x} \in \mathbb{R}^k$ and a convex polytope in $\mathbb{R}^k$, partition the set of points into those inside and outside the polytope.

We assume that the points are stored in a k-D tree (formally defined below). We start by deriving an algorithm that, given a hyperplane, partitions the set of points into points in the positive and negative half-space of the hyperplane, respectively. We then give an algorithm that, given a convex polytope (a set of hyperplanes), partitions the set of points into points that are inside and outside the polytope, respectively.

9.2. k-D trees.

We assume that the points are stored in a k-D tree which can be defined as follows.

Definition 1 (binary point tree) *We define the set of binary trees with points $\mathbf{x} \in \mathbb{R}^k$ stored in the nodes as*

$$\mathcal{T}_k = \{\bot\} \cup \left\{(\mathbf{x}, s, L, R) \mid \mathbf{x} \in \mathbb{R}^k,\ 1 \le s \le k,\ L, R \in \mathcal{T}_k\right\}$$

where $\bot$ denotes the empty tree and s is called the splitting dimension.

[11] https://en.wikipedia.org/wiki/Convex_polytope

Definition 2 (min and max coordinate of a tree) *Let $T \in \mathcal{T}_k$ and $1 \leq s \leq k$. We define the min coordinate in dimension d as*

$$\min_d(T) = \begin{cases} +\infty & \text{if } T = \bot \\ \min\{x_d, \min_d(L), \min_d(R)\} & \text{if } T = (\mathbf{x}, s, L, R). \end{cases}$$

We define the max coordinate in dimension d as

$$\max_d(T) = \begin{cases} -\infty & \text{if } T = \bot \\ \max\{x_d, \max_d(L), \max_d(R)\} & \text{if } T = (\mathbf{x}, s, L, R), \end{cases}$$

where x_i denotes the ith component of vector $\mathbf{x}$.

Definition 3 (k-D tree) *We define the set of k-D trees as*

$$\mathcal{T}_{kD} = \{\bot\} \cup \{(\mathbf{x}, s, L, R) \in \mathcal{T}_k \mid \max_s(L) \leq \mathbf{x}_s \leq \min_s(R),\ L, R \in \mathcal{T}_{kD}\}.$$

9.3. Sub-tree bounding boxes.

The algorithms presented in the following are all based on recursively visiting the nodes of a k-D tree in a depth-first search. While doing this, we maintain the bounding box of all coordinates in the sub-tree rooted in the visited node. The recursion is given in Algorithm 1. We use $\mathbf{x}[i \mapsto y]$ to denote the vector $\mathbf{x}$ with the ith component replaced by y.

Algorithm 1: Sub-tree bounding boxes.

Procedure *visit* $\left((\mathbf{x}, s, L, R), \mathbf{x}^{\min}, \mathbf{x}^{\max}\right)$**:**

```
if L ≠ ⊥ then
    visit (L, x^min, x^max[s ↦ x_s])

if R ≠ ⊥ then
    visit (R, x^min[s ↦ x_s], x^max)
```

It is easy to show that $\forall d, 1 \leq d \leq k \ :\ \max_d(T) \leq x_d^{\max} \wedge \min_d(T) \geq x_d^{\min}$ is an invariant of the recursion in *visit* $\left(T \in \mathcal{T}_{kD}, \mathbf{x}^{\min}, \mathbf{x}^{\max}[s \mapsto x_s]\right)$.

9.4. Splitting k-D tree by a hyperplane.

Let $P = (\mathbf{n}, m)$ with $\mathbf{n} \in \mathbb{R}^k, m \in \mathbb{R}$ denote a k-dimensional hyperplane. Point $\mathbf{x} \in \mathbb{R}^k$ is *on* the plane iff $\mathbf{x} \cdot \mathbf{n} = m$; it is *above* the plane iff $\mathbf{x} \cdot \mathbf{n} \geq m$; it is *below* the plane iff $\mathbf{x} \cdot \mathbf{n} < m$.

Consider a set X of points $\mathbf{x} \in \mathbb{R}^k$. Let a bounding box of X be given by $\left(\mathbf{x}^{\min}, \mathbf{x}^{\max}\right)$ such that

$$\forall \mathbf{x} \in X,\ \forall d, 1 \leq d \leq k \ :\ x_d^{\min} \leq x_d \leq x_d^{\max}.$$

To determine whether all points in X lie above or below a hyperplane $(\mathbf{n}, m)$ respectively, it is sufficient to check the bounding box corner that is furthest along the negative or positive direction of the normal $\mathbf{n}$. This is formalized in functions *allAbove* and *allBelow* in Algorithm 2.

Algorithm 2: Bounding box above or below plane.

Function *allAbove* $\left(\mathbf{x}^{\min}, \mathbf{x}^{\max}, (\mathbf{n}, m)\right) \ : \ b \in \mathbb{B}$

$\mathbf{x} := (x_1, \ldots, x_n), \ x_d = \begin{cases} x_d^{\min} & \text{if } n_d \geq 0 \\ x_d^{\max} & \text{if } n_d < 0 \end{cases}$

return $\mathbf{x} \cdot \mathbf{n} \geq m$

Function *allBelow* $\left(\mathbf{x}^{\min}, \mathbf{x}^{\max}, (\mathbf{n}, m)\right) \ : \ b \in \mathbb{B}$

$\mathbf{x} := (x_1, \ldots, x_n), \ x_d = \begin{cases} x_d^{\min} & \text{if } n_d < 0 \\ x_d^{\max} & \text{if } n_d \geq 0 \end{cases}$

return $\mathbf{x} \cdot \mathbf{n} < m$

It is easy to show that

- if *allAbove* $\left(\mathbf{x}^{\min}, \mathbf{x}^{\max}, (\mathbf{n}, m)\right) = \mathit{true}$ then all points in the bounding box $\left(\mathbf{x}^{\min}, \mathbf{x}^{\max}\right)$ are above the plane, and
- if *allBelow* $\left(\mathbf{x}^{\min}, \mathbf{x}^{\max}, (\mathbf{n}, m)\right) = \mathit{true}$ then all points in the bounding box $\left(\mathbf{x}^{\min}, \mathbf{x}^{\max}\right)$ are below the plane.

Given these functions we can devise an algorithm that partitions points in a k-D tree $T = (\mathbf{x}, s, L, R) \in \mathcal{T}_{kD}$ into sets A (points above the hyperplane) and B (points below the hyperplane) as follows: Check whether $\mathbf{x}$ is above or below the hyperplane and add it to A or B accordingly. Determine bounding boxes for L and R as in Algorithm 1 and test whether these are *allAbove* or *allBelow* the hyperplane. If so, add all points in sub-trees L and R to A or B, respectively. Otherwise recursively descent into L and R.

This computation can be made more efficient by eliminating certain checks for L and R. For example, assume that $\mathbf{x}$ is *above* the hyperplane. Further assume that $n_s \geq 0$. Because we recursively descended into T, we already know that the bounding box of T is not *allAbove* the hyperplane. This means that the bounding box corner furthest to the negative normal direction is not *above* the hyperplane. Now, the bounding box for L only differs from the bounding box of T in that $x_s^{\max} = x_s$. Because $n_s \geq 0$, the bounding box corner of L furthest to the negative normal direction will have sth component equal to $x_s^{\min}$. This means that the bounding box corner furthest to the negative normal direction for L has the same coordinates as that for T. Therefore, we already know that L is not *allAbove* the hyperplane. Consequently we can eliminate the *aboveAll* check for L and recursively descent immediately. Similar considerations can be made for other combinations of sign of n_s and L or R.

The resulting algorithm is given in Algorithm 3, where we use $\mathit{all}\,(T)$ to denote the set of all points in the sub-tree T.

Algorithm 3: Split k-D tree points on hyperplane. Given a k-D tree and a hyperplane, the function *split* computes a partition of the points in the tree into sets A and B of point *above* and *below* the hyperplane, respectively.

Function *split* $\left((\mathbf{x}, s, L, R), \mathbf{x}^{\min}, \mathbf{x}^{\max}, (\mathbf{n}, m)\right) \ : \ (A, B) \in \mathcal{P}\left(\mathbb{R}^k\right) \times \mathcal{P}\left(\mathbb{R}^k\right)$

- $p := \mathbf{x} \cdot \mathbf{n} \geq m$ `// set` p `if` $\mathbf{x}$ `is above hyperplane`
- $q^L := n_s < 0$ `// set` q^L `if` $\mathbf{n}$ `points towards left child`
- $q^R := n_s \geq 0$ `// set` q^R `if` $\mathbf{n}$ `points towards right child`

`// handle` $\mathbf{x}$

if p **then**
  - $(A, B) := (\{\mathbf{x}\}, \varnothing)$

else
  - $(A, B) := (\varnothing, \{\mathbf{x}\})$

`// handle left child`

$(A, B) := (A, B) \cup \mathit{splitSubtree}\left(L, \mathbf{x}^{\min}, \mathbf{x}^{\max}[s \mapsto x_s], (\mathbf{n}, m), p, q^L\right)$

`// handle right child`

$(A, B) := (A, B) \cup \mathit{splitSubtree}\left(R, \mathbf{x}^{\min}[s \mapsto x_s], \mathbf{x}^{\max}, (\mathbf{n}, m), p, q^R\right)$

return (A, B)

Function *splitSubtree* $\left(T, \mathbf{x}^{\min}, \mathbf{x}^{\max}, P, p, q\right) \ : \ (A, B) \in \mathcal{P}\left(\mathbb{R}^k\right) \times \mathcal{P}\left(\mathbb{R}^k\right)$

if $p \wedge q \wedge \mathit{allAbove}\left(\mathbf{x}^{\min}, \mathbf{x}^{\max}, P\right)$ **then**
  - **return** $(\mathit{all}(T), \varnothing)$

else if $\neg p \wedge \neg q \wedge \mathit{allBelow}\left(\mathbf{x}^{\min}, \mathbf{x}^{\max}, P\right)$ **then**
  - **return** $(\varnothing, \mathit{all}(T))$

else
  - **return** $\mathit{split}\left(T, \mathbf{x}^{\min}, \mathbf{x}^{\max}, P\right)$

9.5. Splitting k-D tree into Inside and Outside of a Convex Polytope.

Now assume that we are given a convex polytope $C = \{P_1, \ldots P_h\}$ defined by hyperplanes $P_i = \left(\mathbf{n}^i, m^i\right)$ such that points $\mathbf{x} \in \mathbb{R}^k$ are *inside* C if they are above all hyperplanes P_i and *outside* C otherwise. We want to partition points in a k-D tree $T = (\mathbf{x}, s, L, R) \in \mathcal{T}_{kD}$ into sets A and B of points inside and outside the polytope, respectively.

Using the same reasoning as in Section 9.4 we can devise an algorithm that partitions points in a k-D tree $T = (\mathbf{x}, s, L, R) \in \mathcal{T}_{kD}$ into sets A (points inside the polytope) and B (points outside the polytope) as follows: Check whether $\mathbf{x}$ is above all hyperplanes P_i. If so, add $\mathbf{x}$ to A, otherwise add it to B. Determine bounding boxes for L and R as in Algorithm 1, and test whether these are *allAbove* and *allBelow* all hyperplanes P_i. If the bounding box for L (or R) is above *all* of the hyperplanes P_i, add all points in the sub-tree L (or R) to set A. If the bounding box for L (or R) is below *a single one* of the hyperplanes P_i, add all points in the sub-tree L (or R) to set B. Otherwise recursively descent into L and R.

We can make the following considerations to make the computation more efficient:

- Certain checks for individual hyperplanes can be eliminated by the same reasoning as in Section 9.4.
- If a sub-tree is *allBelow* a single hyperplane, we can stop checking further hyperplanes. The recursion can be terminated and the whole sub-tree can be added to the *outside* set B.
- If a sub-tree is *allAbove* a given hyperplane, all sub-trees further down the recursion will be *allAbove* this hyperplane as well. Consequently, that hyperplane can be removed from the set of hyperplanes to consider for this branch of the recursion. If in this process the set of hyperplanes becomes empty, recursion can be terminated and the whole sub-tree can be added to the *inside* set A.

The resulting algorithm is given in Algorithm 4, where $P_i = \left(\mathbf{n}^i, m^i\right)$ and we use $all\,(T)$ to denote

the set of all points in the sub-tree T.

Algorithm 4: Partition k-D tree points into interior and exterior of a polytope. Given a k-D tree and a convex polytope, the function *clip* computes a partition of the points in the tree into sets A and B of point *inside* and *outside* the polytope, respectively.

Function $clip\left((\mathbf{x}, s, L, R), \mathbf{x}^{\min}, \mathbf{x}^{\max}, \{P_1, \ldots, P_h\}\right) \ : \ (A, B) \in \mathcal{P}\left(\mathbb{R}^k\right) \times \mathcal{P}\left(\mathbb{R}^k\right)$

foreach $1 \leq i \leq h$ **do**

$p_i := \mathbf{x} \cdot \mathbf{n}^i \geq m^i$ // set p_i if $\mathbf{x}$ is above hyperplane P_i

$q_i^L := n_s^i < 0$ // set q_i^L if $\mathbf{n}^i$ points towards left child

$q_i^R := n_s^i \geq 0$ // set q_i^R if $\mathbf{n}^i$ points towards right child

$\mathbf{p} := (p_1, \ldots, p_h)$

$\mathbf{q}^L := \left(q_1^L, \ldots, q_h^L\right)$

$\mathbf{q}^R := \left(q_1^R, \ldots, q_h^R\right)$

// handle $\mathbf{x}$

if $\bigwedge_i p_i$ **then**

$(A, B) := (\{\mathbf{x}\}, \varnothing)$

else

$(A, B) := (\varnothing, \{\mathbf{x}\})$

// handle left child

$(A, B) := (A, B) \cup$

$clipSubtree\left(L, \mathbf{x}^{\min}, \mathbf{x}^{\max}[s \mapsto x_s], \mathbf{p}, \mathbf{q}^L, \{P_1, \ldots, P_h\}\right)$

// handle right child

$(A, B) := (A, B) \cup$

$clipSubtree\left(R, \mathbf{x}^{\min}[s \mapsto x_s], \mathbf{x}^{\max}, \mathbf{p}, \mathbf{q}^R, \{P_1, \ldots, P_h\}\right)$

return (A, B)

Function $clipSubtree\left(T, \mathbf{x}^{\min}, \mathbf{x}^{\max}, \mathbf{p}, \mathbf{q}, \{P_1, \ldots, P_h\}\right) \ : \ (A, B) \in \mathcal{P}\left(\mathbb{R}^k\right) \times \mathcal{P}\left(\mathbb{R}^k\right)$

$\mathcal{P} := \{P_1, \ldots, P_h\}$

foreach $1 \leq i \leq h$ **do**

if $p_i \wedge q_i \wedge allAbove\left(\mathbf{x}^{\min}, \mathbf{x}^{\max}, P_i\right)$ **then**

$\mathcal{P} := \mathcal{P} \setminus \{P_i\}$

else if $\neg p_i \wedge \neg q_i \wedge allBelow\left(\mathbf{x}^{\min}, \mathbf{x}^{\max}, P_i\right)$ **then**

return $(\varnothing, all(T))$

if $\mathcal{P} = \varnothing$ **then**

return $(all(T), \varnothing)$

else

return $clip\left(T, \mathbf{x}^{\min}, \mathbf{x}^{\max}, \mathcal{P}\right)$

9.6. Source code availability.

Implementations of the algorithms discussed above are provided in ImgLib2 [10]. The *split* algorithm for splitting a k-D tree by a hyperplane is implemented in `SplitHyperPlaneKDTree`[12] in the `kdtree` package. The *clip* algorithm for splitting a k-D tree into inside and outside of a convex polytope is implemented in `ClipConvexPolytopeKDTree`[13] in the same package.

[12] net.imglib2.algorithm.kdtree.SplitHyperPlaneKDTree
[13] net.imglib2.algorithm.kdtree.ClipConvexPolytopeKDTree

References

[1] T. Pietzsch, S. Saalfeld, S. Preibisch, and P. Tomancak, "Bigdataviewer: visualization and processing for large image data sets," *Nature Methods*, vol. 12, p. 481, 2015.

[2] D. Hörl, F. Rojas Rusak, F. Preusser, P. Tillberg, N. Randel, R. K. Chhetri, A. Cardona, P. J. Keller, H. Harz, H. Leonhardt, M. Treier, and S. Preibisch, "Bigstitcher: reconstructing high-resolution image datasets of cleared and expanded samples," *Nature Methods*, vol. 16, no. 9, pp. 870–874, 2019.

[3] C. Wolff, J.-Y. Tinevez, T. Pietzsch, E. Stamataki, B. Harich, L. Guignard, S. Preibisch, S. Shorte, P. J. Keller, P. Tomancak, and A. Pavlopoulos, "Multi-view light-sheet imaging and tracking with the mamut software reveals the cell lineage of a direct developing arthropod limb," *eLife*, vol. 7, p. e34410, mar 2018.

[4] F. Amat, W. Lemon, D. P. Mossing, K. McDole, Y. Wan, K. Branson, E. W. Myers, and P. J. Keller, "Fast, accurate reconstruction of cell lineages from large-scale fluorescence microscopy data," *Nature Methods*, vol. 11, no. 9, pp. 951–958, 2014.

[5] J.-Y. Tinevez, N. Perry, J. Schindelin, G. M. Hoopes, G. D. Reynolds, E. Laplantine, S. Y. Bednarek, S. L. Shorte, and K. W. Eliceiri, "Trackmate: An open and extensible platform for single-particle tracking," *Methods*, vol. 115, pp. 80 – 90, 2017. Image Processing for Biologists.

[6] D. Sage, F. Neumann, F. Hediger, S. Gasser, and M. Unser, "Automatic tracking of individual fluorescence particles: Application to the study of chromosome dynamics," *IEEE Transactions on Image Processing*, vol. 14, pp. 1372–1383, September 2005.

[7] N. Chenouard, I. Smal, F. de Chaumont, M. Maška, I. F. Sbalzarini, Y. Gong, J. Cardinale, C. Carthel, S. Coraluppi, M. Winter, A. R. Cohen, W. J. Godinez, K. Rohr, Y. Kalaidzidis, L. Liang, J. Duncan, H. Shen, Y. Xu, K. E. G. Magnusson, J. Jaldén, H. M. Blau, P. Paul-Gilloteaux, P. Roudot, C. Kervrann, F. Waharte, J.-Y. Tinevez, S. L. Shorte, J. Willemse, K. Celler, G. P. van Wezel, H.-W. Dan, Y.-S. Tsai, C. O. de Solórzano, J.-C. Olivo-Marin, and E. Meijering, "Objective comparison of particle tracking methods," *Nature Methods*, vol. 11, p. 281, 2014.

[8] K. Jaqaman, D. Loerke, M. Mettlen, H. Kuwata, S. Grinstein, S. L. Schmid, and G. Danuser, "Robust single-particle tracking in live-cell time-lapse sequences," *Nature Methods*, vol. 5, p. 695, 2008.

[9] K. McDole, L. Guignard, F. Amat, A. Berger, G. Malandain, L. A. Royer, S. C. Turaga, K. Branson, and P. J. Keller, "In toto imaging and reconstruction of post-implantation mouse development at the single-cell level," *Cell*, vol. 175, no. 3, pp. 859–876.e33, 2018.

[10] T. Pietzsch, S. Preibisch, P. Tomančák, and S. Saalfeld, "ImgLib2—generic image processing in Java," *Bioinformatics*, vol. 28, pp. 3009–3011, 09 2012.