TrakEM2 0.4y User Manual

Last updated: 2007-12-19 19:09 Los Angeles time.
Download the latest manual in **PDF format** here.

When clueless: right-click is your friend!

Introduction

Setup

Create, save, open and export a project

- A database project
  - Start a new project
  - Open an existing project
  - Save a project: already done!
  - Exporting a project to XML
- An XML file-system project
  - Start a new project
  - Open an existing project
  - Save and Save As

Introduction to modeling

- Creating the abstract structure
- Creating basic data types
  - Filling a profile list with profiles
  - Adding a pipe object
  - Adding a ball object
- Preview 3D
- Exporting 3D

Basic usage:

- Layers and Displays: navigate, adjust dimensions, add and remove
  - Add a layer
  - Add a layer set
  - Adjust a layer’s Z coordinate and thickness
  - Reverse selected layers Z order in one shot
  - Adjust a layer set width and height
  - Delete a layer
  - Delete a layer set
  - Navigate a layer set in a display
  - Navigate a layer
  - Register entire layers one to another
  - Register two layers with landmarks
  - Register a sequence of layers by selected profile lists
- Images: importing, adjusting, transforming, reverting, stitching
  - Importing single images
  - Importing stacks of images
  - Importing an image grid or montage
  - Importing a sequence of images as a grid or montage
  - Import images as listed in a text file

Adding functionality with your own plugins

Using TrakEM2 as a framework

Appendix

Introduction

What is TrakEM2?

TrakEM2 is a program to **model** from images. To model *what* is up to you.
Two different kinds of modeling concur:

- **Three-dimensional modeling**: objects in 3D, defined by sequences of contours, or profiles, from which a skin, or mesh, can be constructed, and visualized in 3D in programs such as Blender , Autocad, Maya, etc.
- **Relational modeling**: the extraction of the map that describes links between objects. For example, which neuron contacts which other neurons through how many and which synapses.

To accomplish the above tasks TrakEM2 relies on the segmentation of the objects represented in the images. That is, the outlining of each section of an object as it appears in the images, and its orderly storage in the hierarchy of existing objects.
But there's more! TrakEM2 is built on top of ImageJ, the standard in the image analysis and research

community. In the graphical interface of TrakEM2, each image and its elements can be **analyzed**, **measured**, **quantified** and **transformed** with ImageJ internal functionality, and by its enormous collection of freely contributed plugins.

How is TrakEM2 different?

TrakEM2 has been designed to handle an arbitrarily large amount of images, as many as your hard drives can store.
TrakEM2 can manage large amounts of data by storing and retrieving it dynamically from a PostgreSQL database. As of version 0.3g, the program can run just as well on XML, dynamically loading and flushing images from the file system.

Why another program?

Because there is no such program out there that can:

- Handle, display, navigate, annotate, measure, transform and store a virtually infinite number of images.
- Model in 3D while generating a relational tree, the logical skeleton of the objects in a sample.

Our purpose is to **perform 3D modeling in a meaningful way**, not just to generate pretty pictures. We are targeting the extraction of the relational structure for the purpose of running simulations in computers, using the 3D models as a nice visualization tool for humans. We also want to compare structures to themselves and to one another across multiple experiments on similar samples.
In addition, we plan to enable **distributed** annotation, modeling and analysis of a large sample. The database helps in keeping data centralized and will enable, in the future, the compartmentalization of privileges in the 3D space of the sample, for multiple users to work on the same dataset without colliding with each other.

Setup

Minimal system requirements

Minimal requirements:

- Java 1.5.0 or above (1.6.0 STRONGLY recommended).
- ImageJ 1.37g or above.
- Bene Schmid's ImageJ 3D Viewer (place the jar file in ImageJ's plugins folder) (local copy.)
- edu_mines_jtk.jar (place the jar file in ImageJ's plugins folder) (local copy.)
- Jama-1.0.2.jar (place the jar file in ImageJ's plugins folder) (local copy.)

The local copies are **strongly** recommended, for they have been tested and are know to work properly.
For **database projects**, you'll need in addition to the above:

- A PostgreSQL installation, 7.4.12 or above (8.1.4 recommended).
- The JDBC postgreSQL driver.

There is a detailed TrakEM2 installation how-to.
In the real world, for handling a 300 Gigabyte project with 150,000 images, this is the computer I use:

- Intel core 2 extreme (4 cores) 2.40 Ghz
- 4 Gb DDR3 800 Mhz RAM
- 4 HD 500 Gb each, 7200 RPM
- Ubuntu Linux 7.04 64-bit
- java 1.6.0 64-bit

which I assembled from parts for about $3,600 in January 2007 together with Wayne Pereanu.

The database (optional)

Install PostgreSQL

Ready-made packages

The PostgreSQL database system can be installed most trivially from packages via the apt-get system (debian GNU/Linux derivatives), the ports collections (*BSD) and from RPM (RedHat GNU/Linux derivatives). In Windows, you can install binaries as distributed in the postgresql.org website. For other systems, you need to install from source as described below.

Installing from source

Download the PostgreSQL sources and then open (in MacOSX) the Terminal.app from /Applications/Utilities/
Create a folder to hold your PostgreSQL installation, in this example the folder pgsql under /Users/albert/
You need to have installed the readline library, make and the GCC. In MacOSX, you can get the GCC from the Developer Tools. The readline and make easily via fink, by typing in a Terminal:

```
$ fink install readline make
```

Then unpack, configure and install:

```
$ pwd
/Users/albert/TrakEM2
$ ls
postgresql-8.1.3.tar.bz2
$ bzip2 -cd postgresql-8.1.3.tar.bz2 | tar xvf -
[ ... output skipped ...]
$ mkdir pgsql
$ ls
pgsql/
postgresql-8.1.3/
postgresql-8.1.3.tar.bz2
$ cd postgresql-8.1.3
$ ./configure --with-java --prefix=/Users/albert/TrakEM2/pgsql
$ make install clean
```

In MacOSX, you will need to specify the location of the fink-installed libraries and binaries, such as:

```
$ ./configure --with-java --prefix=/Users/albert/TrakEM2/pgsql --with-includes=/sw/include/ --with-libraries=/sw/lib
$ make install clean
```

Create the database: a one-time step

Your default PostgreSQL installation comes with a prompt from which the databases can be managed. First, though, a folder to contain the databases has to be created and initialized:

```
$ cd /Users/albert/TrakEM2
$ mkdir data
$ ./pgsql/bin/initdb -D data/
[ .. output skipped .. ]
```

Now launch the postmaster database daemon on our database folder:

```
$ ./pgsql/bin/pg_ctl start -D data
```

Now we can enter the default database named 'template1' and create our own, in this case I've named it 'shared':

```
$ ./pgsql/bin/psql template1
template1# create database shared;
template1# create user imagej with password 'anypassword';
template1# create group shared;
template1# alter group shared add user imagej;
template1# \q
$
```

Now shut down the database:

```
$ ./pgsql/bin/pg_ctl stop -D data
```

Several settings need to be adjusted to restrict the access to the privileged users and enable ImageJ to reach the database through network sockets. Changing the settings is only allowed when the database is shut down.
Open the file postgresql.conf contained in the database folder 'data' and uncomment this line below, to enable TCP/IP access to the database:

```
#listen_addresses = 'localhost'
```

So that now it reads like:

```
listen_addresses = '*'
```

With the asterisk, we are enabling any IP from all over the internet to connect to our database. For a more restrictive setup, enter a comma separated list of IP addresses, or just 'localhost' for the local computer. Further, if we are to insert hundreds of images at a time it will speed up things considerably to set more checkpoint segments, so that:

```
#checkpoint_segments = 3
```

Now reads like:

```
checkpoint_segments = 50
```

Another setting to consider is in the data/pg_hba.conf file. At the bottom of the file, the database can be setup to require password authentication. For example, to enable all users belonging to a group of users whose name is the same of the database (my preferred settings), comment the last lines and add, so that it reads:

```
# TYPE  DATABASE    USER        CIDR-ADDRESS          METHOD
#local  all     all                     trust
# "local" is for Unix domain socket connections only
#local   all         all                                trust
# IPv4 local connections:
#host    all         all         127.0.0.1/32          trust
# IPv6 local connections:
#host    all         all         ::1/128               trust

host    samegroup       all     0.0.0.0         0.0.0.0 md5
local   samegroup       all                              md5
```

The above settings enable any user that belongs to a database user group whose name is coincident with the database you want them to access, to access the database from any IP, but requiring password authentication via the md5 method.
With the above settings, our user ’imagej’ that belongs to the database group ’shared’ can now enter, query and edit the database ’shared’ by producing the appropriate user password.

Launching the database

Whenever you need the database up, enter a terminal and type:

```
$ cd /Users/albert/TrakEM2/
$ ./pgsql/bin/pg_ctl start -D data/
```

I simply leave it always on in my remote storage server, where the images and project data live.

Database performance tuning and maintenance

See the PostgreSQL maintenance manual page for lots of details.
The most relevant:

- If large image sets are going to be inserted over short periods of time and disk space is not such a problem, a considerable speed increase can be achieved by increasing the checkpoint segments from the default 3 to for example 50. Open the database folder’s postgresql.conf file and edit:

  ```
  checkpoint_segments = 50       # in logfile segments, min 1, 16MB each
  ```

- Provide more RAM for each internal database process. In my system with 4GB of DDR3 RAM, I use:

  ```
  work_mem = 128 MB              # min 64kB
  ```

  ... which boost image loading speed **nearly 20x !**

- The autovacuum system (versions 8.* and above) will also need more memory to run properly. For my 300 GB databases, I use:

```
maintenance_work_mem = 128MB   # min 1MB
```

- Manage tablespaces smartly to squeeze the maximum performance from your database, and/or to setup several hard drives to work together as a single database.
- Run VACUUM FULL daily in a cron job, preferably late at night when no users are expected to be quering the database. See the PostgreSQL manual on VACUUM for all details, including an AUTOVACUUM feature new in version 8.1. The AUTOVACUUM can be setup in the database's postgresql.conf file. Search for "autovacuum" in it and adjust to your needs.

Setup ImageJ classpath

The PostgreSQL JDBC driver is required for TrakEM2 to run **only if you are planning to use a database** backend. Otherwise you can **ignore it**.
To include the driver in ImageJ's classpath:

- In MacOSX(TM), the standard way of adding an extra, non-plugin jar file to ImageJ for plugins to compile and run fine is to drop it in the /Library/Java/Extensions folder and restart ImageJ.
  Alternatively, you can edit the ImageJ.app/Contents/Info.plist file to include the driver's jar in the classpath. To edit the Info.plist, right-click the ImageJ.app icon, select "Show Package Contents", and open the Contents folder.
  Assuming the postgresql-8.2-504.jdbc3.jar has been dropped in the ImageJ folder, edit the Info.plist with any text editor, so that the 'ClassPath' key reads like:

```
        ...
<key>ClassPath</key>
<string>$APP_PACKAGE/../ij.jar:$APP_PACKAGE/../postgresql-8.2-504.jdbc3.jar</string>
        ...
```

  If you drop the postgresql-8.2-504.jdbc3.jar in the ImageJ.app/Contents/Resources/Java/ (old way), then use $JAVAROOT/postgresql-8.2-504.jdbc3.jar. As a third alternative, launch ImageJ directly from the command line as in *nix systems (see below).
- In *nix operating systems (GNU/Linux, FreeBSD, MacOSX(TM), etc.), simply modify the startup script to include the JDBC driver jar:

```
java -Xmx1024m -Xincgc -classpath ij.jar:postgresql-8.2-504.jdbc3.jar ij.ImageJ
```

- In Windows(TM), the standard way of adding an extra, non-plugin jar file to ImageJ for plugins to compile and run fine is to drop it in the ImageJ\jre\lib\ext folder and restart ImageJ.
  Alternatively, just edit the ImageJ.cfg file to include the JDBC jar in the classpath. Separate the different jar files with a ';'. Assuming the postgresql-8.2-504.jdbc3.jar is in the same folder as the ij.jar, edit ImageJ.cfg so that it reads like (edit the maximum memory -Xmx512m to your convenience):

```
.
jre\bin\javaw.exe
-Xmx1024m -Xincgc -cp ij.jar;postgresql-8.2-504.jdbc3.jar ij.ImageJ
```

The project XML template

Deciding beforehand what to model

So what do you want to model? The TrakEM2 program offers 5 different basic object types, which can be combined with abstract types to construct an object hierarchy. In plain words, from images the user can extract the three-dimensional outline of objects and their interrelationships. How does this work?
When creating a new project via the ImageJ -> Plugins -> TrakEM2 -> New Project command, the very first that the user must enter is an XML file where the object structure to model is specified. For example, if one is to model neuronal tissue, a snippet of such a template may look like:

```
[...]
<nervous_tissue id="1" title="mouse neocortex">
    <neuronal_network id="300" title="region 1">
        <neuron id="301" title="neuron">
            <soma id="423">
                <nucleus id="424">
                    <profile_list>
                        <profile id="1001" index="1" />
                        <profile id="1012" index="2" />
                        <!-- many more -->
                    </profile_list>
                </nucleus>
                <junction id="545">
                    <profile_list>
                        <profile id="870" index="1" />
                        <profile id="871" index="2" />
                        <!-- many more -->
                    </profile_list>
                </junction>
                <!-- more junctions -->
            </soma>
            <dendrite_tree id="430">
                <dendrite_branch id="431">
                    <profile_list>
                        <profile id="432" index="1" />
                        <profile id="433" index="2" />
                        <!-- many more -->
                    </profile_list>
                    <spine_head id="512">
                        <profile_list>
                            <profile id="432" index="1" />
                            <profile id="433" index="2" />
                            <!-- many more -->
                        </profile_list>
                    </spine_head>
                    <dendrite_branch id="431">
                         <!-- a CHILD branch from the branch -->
                         <!-- ... --->
                    </dendrite_branch>
```

```
                </dendrite_branch>
            <dendrite_tree>
     ....... AND SO ON
[...]
</nervous_tissue>
```

What we can see above is a hierarchy of objects defined by opening and closing tags, which enclose objects within, which in turn enclose other objects, until we reach objects of type **profile_list** which only contain a list of **profile** objects. These profiles are user-drawn curves that define an outline, for example the outline of the nuclear membrane in a neuron.
So one can read, a neuronal network is composed of many neurons, each neuron composed of a soma and a dendritic and axonal trees, the soma defined by a list of profiles, and also containing several junctions with other nearby somas, and so on, to whatever level or resolution one wants to go, or the data allows one to go.
See example XML templates here:

- Drosophila's brain secondary lineages as DTD
- Drosophila's brain secondary lineages as XML
- Generic neuronal network (as it can be visualized in electron microscopical images)

Create and edit the template on the fly

As for version 0.3g, a project can be created without any preselected XML template. The empty template tree can be filled up as the need arises, and edited any time.
When creating a new project a dialog opens to choose a template .dtd or .xml file. If you cancel the dialog, the project is created with a root node labeled "anything". Use the right-click popup menu to rename it and add children to it.
If a template node is deleted, and there exist project objects that depend on the template, a dialog will prompt on whether to delete the project objects as well. Project objects cannot exist without a template.

Export and import a template

The basic abstract types: profile, pipe and ball

There are three basic types that can be wrapped in higher-order, abstract objects in the XML template for our sample. Briefly:

- Profile: a user-drawn outline, flat on a section of tissue.
- Pipe: a multi-point bézier curve that defines a tube, where each point is a joint that can have its independent radius, and further, can lay anywhere in the space (a pipe can cross several sections).
- Ball: a spherical object. Mostly used for small cells or nuclei at the optical level.

So **before** ever starting touching any image at all, the user must agree with himself and the research team on an ontology or object structure to use. In this way, the modeling flows within an enforced structure that leads to the extracting of meaningful relationships between the modeled objects.

Create, save, open and export a project

A database project

Start a new project

When the XML template is ready, TrakEM2 can be started. Ensure the database is up and running in your server (or local computer), and then open ImageJ and run the "TrakEM2 -> New Project" plugin.
The expected sequence of events is as follows:

1. Connect to the database: enter the database server IP, database name, user login and password.
2. Select an XML template. Without it, a project cannot be created. The TrakEM2 main window will open, displaying your project template as a tree.
   As of version 0.3g, if no .xml template file is selected the project will still be created, with an empty template tree. Go and edit the tree on the fly as the need arises.
3. Optionally, rename your project from 'Project' to a meaningful name. Double-click the tab or right-click the project node and select "Rename".
4. Now you have two options: in the Layer Tree, right-click the Top Level Layer Set and proceed to:
   - add a stack of images.
   - add a new layer.

If you add a stack, layers will be automatically created to accommodate one slice of the stack in each.
If you add a layer, then you can import images into the layer, for example an image grid defining a larger montage that represents an electron microscopy section.
Once you have imported images (you can import more, and create more layers, at any time) you are ready to start modeling.

Open an existing project

To continue working on a project, select the "TrakEM2 -> Open Project" plugin. Enter the database server IP, the database name, and your database user and password. A dialog will let you choose between existing projects in the database. If the database contains a single project, it will be opened directly.
You can record a macro to automate this process. Then, in future sessions you can open your project by selecting it from the ImageJ macros menu, or even assign a keyboard shortcut to it.

Save a project: already done!

There is no need to save a project, for every action is saved the moment it's done.
That's why there are always dialogs asking for confirmation any time a deletion is going to be performed.

Exporting a project to XML

From the TrakEM2 window, displaying the trees, right-click on the Project node (the root node of the tree in the middle), and select "Export XML...".
What will happen: an XML file (human-readable text) will be generated, containing all the object hierarchies in your project, including labels, profiles, etc. The images, though, will be exported to a folder in the same directory as the XML file, named like the XML file. Be sure to have enough storage space in the selected folder!

An XML file-system project

XML file-system based projects are **completely independent** of a database. Thus, you **don't** even need to have the driver installed to work with XML projects.

Start a new project

Select Plugins-> TrakEM2 -> New Project, and then optionally choose a proper template file, either as a .dtd or an .xml file (the DOCTYPE declaration will be read if explicit, not an URL). If the template selection file dialog is canceled, a project with an empty (but editable) template is created.
In an XML project, the original images are not saved whenever modified as in the database project. An image can always be duplicated with ImageJ's Image-> Duplicate... command (shift+d).
In an XML project you are in charge of housekeeping the images.

Open an existing project

Select Plugins-> TrakEM2 -> Open Project, and then select the XML file in which the project is contained.
The images will be loaded from the proper file path entries in the XML 'patch' tags.
When images cannot be found (because for example the CD in which they lived is not mounted), an empty image with a large 'x' in the center is drawn instead.

Save and Save As

Either push 's' on a open display to save, or:

- In the project tree, right-click and select "Save" to overwrite the project XML file.
- On a display, right-click and select "Project / Save".

As of version 0.3j, images will have been saved as copies whenever they where edited from within TrakEM2.
If you'd rather save the project under a different name, select "Save As" to create a new XML file. The images are not duplicated, but the new project file points to the same image files as the original did.

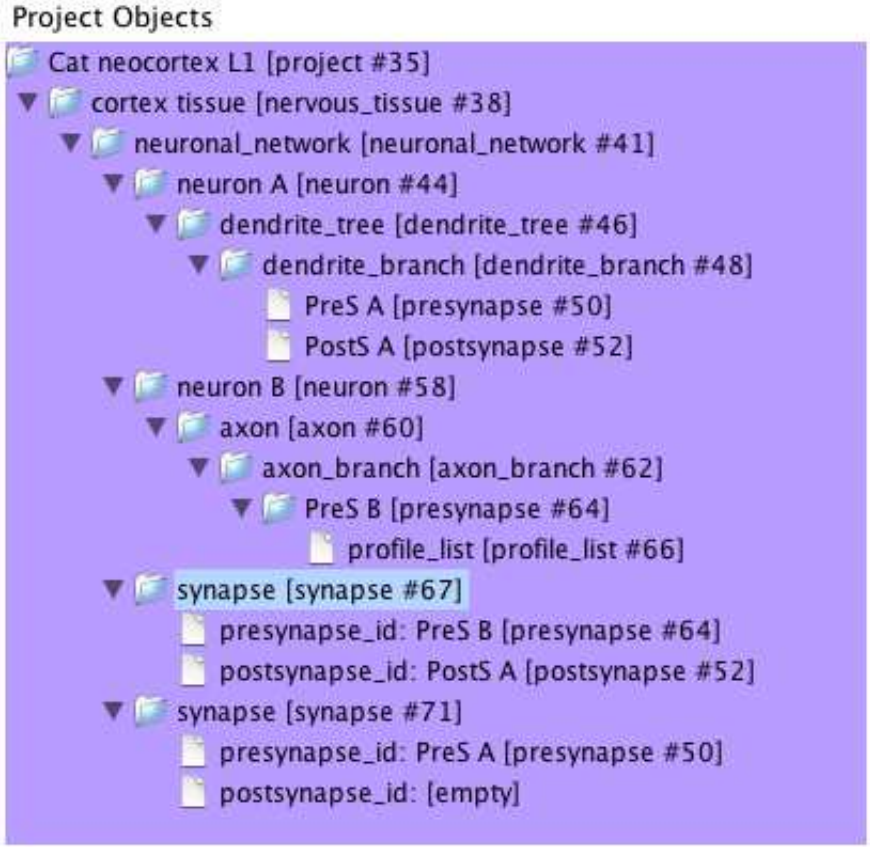
Introduction to modeling

Modeling consists of two parts: create the abstract structure, and create the basic data types wrapped in nodes of the abstract structure.

Creating the abstract structure

Creating the abstract structure is as simple as drag and drop: from the Template Tree, to the left, drag the nodes to the Project Tree, in the center. You will be prevented from dragging nodes where they do not correspond.
There are two types of relationships between abstract nodes: either as parent/child, or as attribute.
As a parent/child example, first create a "neuronal network" node under your Project Tree by dragging it from the Template Tree. Then drag subsequent child nodes such as "neuron", then a "soma" for the "neuron", and finally a "profile list" to define the three-dimensional outline of the "soma".

Now the next step would be to fill in the "profile list" with actual "profiles", which are basic data types. Attributes are defined in the XML template inside the tag of the object they represent. Any number of attributes can be used.
The id and title attributes are already shown in the name of the node, along with its type. Other attributes are listed immediately under the node.
Attributes can hold text and numeric data, but also can be used as pointers to other objects in the structure. When to use attributes as pointers: when the objects that compose a higher level object are owned by someone else. For example, a synapse is but a presynaptic and at least one post synaptic membrane together. Each membrane in the synapse belongs to a separate neuron. The synapse in itself is thus composed of objects that belong to other objects (the neurons). This idea can be represented as the pre- and postsynaptic membranes being attributes of the synapse.
Why this distinction: well, deleting a neuron will delete its branches and pre- and post-synaptic membranes. But the partners of these pre- and postsynaptic membranes will remain unaffected in the Project structure.
Attributes will be shown as either empty, or with the name of the object they point to. Observe the two "Synapse" objects in the Project Tree below. One has both its presynapse_id and postsynapse_id filled in, the second has the postsynapse_id empty:



In versions subsequent to 0.2, attributes will be directly editable. We have yet to explore the use of attributes to store measurements and other metadata.

Creating basic data types

Filling a profile list with profiles

When reaching a basic type such as a profile, you need to have an open Display, which shows the layer in which it will be added.
Double-click on a layer to open a display for it. You can open as many displays as you want, even several for the same layer.
Navigate the layer and locate the region you want to model. You can import more images or stacks by calling the popup menu (right-click on the display) if necessary.
Go back to the trees and drag the basic type to the object you want to model. The foremost display will be brought to front, the pen tool (right-most) from the ImageJ toolbar will be selected, and all you have to do is click and drag several times on the image to create the profile, adjusting the Bézier curve as needed.
Once the first profile has been added to a "profile list", the latter cannot accept anymore profiles to be dragged into it. Instead: when done with the profile in the current layer, right-click and select "duplicate, link and send to next layer". The display will now show the next layer (notice its Z coordinate in the window title), and you can modify the profile to adjust to the new section of the object you are modeling. Iterate until the object is no longer visible in the layers.
If the object you are modeling splits in two or more profiles, go back to the previous layer where the object was still a solid, single profile and do again a "Duplicate, link and send to next layer".
Now you'll have another profile to deform for the new branch.
PLEASE NOTE: as of version 0.2, the 3D models of such branched objects will fail. A proper 3D mesh extraction algorithm is on the works.
If you started modeling in a medial section, you can navigate the layers back (use the '<' and '>' keys) until reaching the first profile. Then, right-click and select "Duplicate, link and send to previous layer". Proceed as a above but in the opposite direction in the volume.
When a basic object is added, the images underneath that support it become linked to it. So now, if you drag the image with the select tool (black arrow), the linked profiles will be dragged as well. And vice versa.

Adding a pipe object

Make sure a display is open. Open a display by double-clicking a layer if necessary. If no displays are open it wouldn't know which layer set to use.
Drag a pipe node from the Template Tree to an appropriate abstract object in the Project Tree, for example an "axonal branch". A display will be brought to front and you start creating the Pipe.

Adding a ball object

Make sure a display is open. Open a display by double-clicking a layer if necessary. If no displays are open it wouldn't know which layer set to use.
Drag and drop a ball node from the Template Tree to an appropriate abstract object in your Project Tree. The front most display will be brought to front, the pen tool selected, and you can start adding ball objects by clicking on the display.

Preview 3D

In the Project Tree, right-click any node and select "Show in 3D". Meshes are generated and presented in the ImageJ 3D Viewer, which must be installed (place its jar file under ImageJ's plugins folder).
Work is on the way to add editing capabilities. For example, to select, measure, delete and join objects.

Exporting 3D

In the 3D Viewer, select File -> export ".obj" (Wavefront format) or export ".dxf" (digital exchange format).
In addition and for backward compatibility, the Project Tree has an "Export 3D". The node and all its children, downward, will be exported into a single file.
========== DEPRECATED ==========
Choose between two formats to export to:

- As of version 0.3, TrakEM2 exports to custom Scalar Vector Graphics files (.svg), which preserve the linking, grouping and coloring, and are 100% compatible with Inkscape and Adobe Illustrator. Work is on the way to provide a proper Blender importer plugin.
- As of version 0.2, TrakEM2 exports to a .shapes file, which can be imported into the OSS 3D suite Blender by means of the load_shapes_v1.4.4.py Blender plugin, which works much nicer with the CurveMorphing module present.
  Beware of current limitations in TrakEM2 0.2 .shapes format: profile lists must have only one profile per layer, or the rendering will fail (in Blender).

========== END ==========
In Blender, the three-dimensional objects can be edited and colored at will, and rendered by ray-tracing. Further, python-sql extension plugins will enable database queries from within Blender in the near future, removing the limitation of simply generating "pretty pictures", providing a 3D interactive navigator of your modeled data and images. Truly an amazing tool!

Basic usage

Layers and Displays: navigate, adjust dimensions, add and remove

Add a layer

A layer is a section with a given Z coordinate and thickness, both demanded at creation time.
A layer exists inside a layer set. A layer set defines the width and height of the layers it contains.
In the TrakEM2 main window you'll find three trees. The one on the right let's you manage the layers and layer sets.
There are two ways to add layers:

- By right-clicking on a layer set and choosing "new Layer".
- By importing a stack into a layer or a layer set. Layers are created automatically as needed.

Add a layer set

Besides the top level layer set, any layer can contain nested layer sets.
In the Layer Tree, right-click a layer and select add new layer set. The new layer set dimensions will be asked for.
Each layer set has its own x,y position in the layer, while at the same time defining the width and height of the layers it contains.
The ability to nest layer sets into layers enables, for example, optical microscopy images to contain sets of sequential, electron microscopy images that were obtained from parts of the optical microscopy sections.

Adjust a layer Z coordinate and thickness

At any time, right-click on a layer node in the Layer Tree to change its Z coordinate or thickness.
WARNING: as of version 0.2, this operation can mess up the construction of profile lists and the extraction of three-dimensional objects in general.
If multiple layers are selected (by shift+click on a second layer or control+click on any number of them), the right-click popup menu will show options to displace all selected layers in the Z axis.

Reverse selected layers Z order in one shot

Select more than one layer in the Layer Tree by shift+click on a second layer, or by control+click on any number of them.
Then right-click and select "Reverse layer Z coords"

Adjust a layer set width and height

At any time, right-click on a layer set node in the Layer Tree to adjust the width and height of the layers it contains. The resizing operation will demand an anchor (top left, top, top right, etc.), so that the objects contained in the layers of the layer set can be placed properly according to the new dimensions.

Adjust a layer set snapshots policy

Right-click on a layer set node in the Layer Tree and adjust the snapshot policy to bounding box or full snapshot. Bounding box mode is very useful to quickly navigate gigantic data sets across it’s layers (Z axis).

Delete a layer

In the Layer Tree, right-click and select "Delete".
The deletion will not be allowed if any of the objects in the layer are linked to objects in other layers.
All the objects contained in the layer will be deleted along with it.

Delete a layer set

Absolutely all objects contained in the layers of the layer set, and the layers themselves, will be deleted.
The top level layer set cannot be deleted.

Navigate a layer set in a display

A display can show you any of the layers of a given layer set. When double-clicking a layer in the Layer Tree, the display that opens will be able to browse through any of its sibling layers.
To scroll through the layers in an open display, use:

- ’<’ and ’>’ keys (or their equivalent ’,’ and ’.’ keys).
- The scroll wheel.

Navigate a layer

Layers can be both zoomed and panned.
Zooming:

- With the glass tool from the ImageJ toolbar.
- With the ’-’ and ’+’ keys (or equivalent ’_’ and ’=’ keys in English keyboards)
- With control + scroll wheel

Panning:

- With the hand tool from the ImageJ toolbar, directly on the images.
- With the mouse, on the navigator window (lower left). A red rectangle shows the area currently displayed.
- With the middle mouse button, click-drag over the images directly - just like in Inkscape and many other graphics applications.

Register entire layers one to another

Select the Align tool (if not obvious, place the mouse over the tools to read their description), and right-click on a display to bring up the popup menu. Select "Register layers (layer-based)", and choose the range of layers that you want to register one to another.
The current layer will be left untouched; the registration will proceed towards any choosen layers before the current layers, and towards any choosen layers after the current layer.
For other layers not included in the range, you can transmit to them the registration applied to the nearest rgistered layer by ticking in "propagate". This will preserve any previous layer registrations you may have done.

Register two layers with landmarks

Using the alignment tool (if not obvious, place the mouse over the tools to read their description), click to add landmarks on any two layers.
There must be the same number of landmarks in both layers.
Only objects under landmarks and their linked ones will be registered.
Push ’ESC’ to cancel, and ’ENTER’ to register the affected objects in the second layer (landmarks in magenta) relative to the first (landmarks in yellow).
The order in which landmarks are added matters: the algorithm will minimize the square of the distance between every pair of landmarks (which are numbered).

Register a sequence of layers by selected profile lists

First open a display and select as many profiles as you want to use, each potentially belonging to its own profile list. Use shift+click with the black arrow tool selected to select more than one, or just shift+click over the left panel on the profiles' tab.
Then select the alignment tool, and right click on the display and select "Align using profiles". The entire contents of all layers will be affected.
All layers where the involved profile lists have profiles will be aligned using consecutive profile pairs as landmarks. When no profiles exist for a layer, the proper transformation will apply relative to transformations in other layers.
You can always undo the transformations
.

Create a flat image

On any display, right-click and select "Display -> Flat image". Options will be provided to create an image of the visible area or the entire layer. A regular ImageJ image window will open.
The image defaults to RGB to paint properly in colors. The pixel accuracy from 16- and 32-bit images will be thus lost.
The dialog lets you select any range of layers to take a snapshot from, and also directly to an image sequence if desired. If the resulting stack is too large to fit in memory, a dialog will prompt to save the stack as an image sequence, or abort the operation.

Images: importing, adjusting, transforming, reverting, stitching

Grayscale and RGB images can coexist in the same layer. 8-bit, 8-bit color, 16-bit, 32-bit and RGB can all coexist.
Each image, when selected, behaves like any other ImageJ image. Plugins and commands can be run on them.
At any point, an image can be reverted to its original state from the contextual menu.
Also, any ROI tool will work as well. Select the ROI tool from the ImageJ toolbar and create the ROI on the display. The ROI will apply to the currently selected image.
As of version 0.2 there are some limitations in ROI usage. Calling measuring and other commands such as copy/cut/paste works for some ROI types but not others. Full support is on the works.

Importing single images

Open a display by double-clicking a layer in the Layer Tree, and then right-click the display canvas.
From the contextual menu, select "Import -> Image".
As of version 0.3b, these key bindings are available:

- shift+alt+i: import an image
- alt+i: import the next image in the folder, or a new image if none have been opened yet.
- by drag and drop directly into an open display. The image top left corner will be placed where the mouse was released.

If the image is an stack, you'll be asked whether to import it, creating new layers as it needs them.

Importing stacks of images

A stack is a set of images of sections consecutive in the z axis, and which have the same width and height, and usually properties (8-bit or color).
There are three ways to import a stack of images:

- In the Layer Tree, right-click a layer set that contains no layers and select "import stack".
- In the Layer Tree, right-click an existing layer, and select "import stack".
- Right-click a display and select "Import -> Stack".

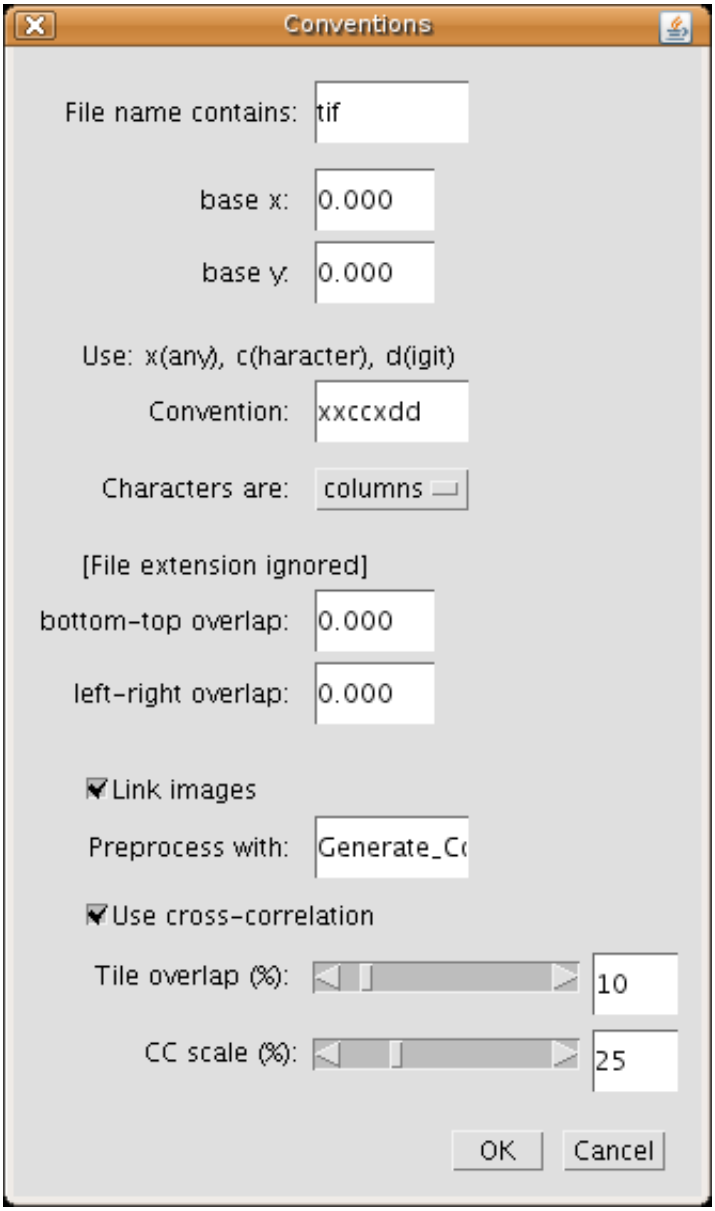
New layers are created on the fly to accommodate the sections of the stack, if necessary.
Whenever possible, existing layers are used.
As of version 0.3j, drag and drop image stacks directly into the canvas, or folders containing image sequences.

Importing an image grid or montage

An image grid or montage is a collection of images, where each image shows a part of the same section of the sample.
There are three ways to import an image grid:

- In the Layer Tree, right-click on a layer and select "Import grid...". This option has the advantage that it can be performed without a display showing, i.e. when finished, there is no delay related to repainting.
- In an open display, right-click and select "Import -> Import grid"
- Drag and drop a folder of images directly into the canvas.

The layer set width and height will be enlarged to accommodate the montage, whenever necessary.
After selecting an image from a folder, a dialog will popup asking for parameters:



Options are:

- Filter file names in the folder by a sequence of common characters (optional).
- The top left coordinate of the grid (or montage) of images to import, i.e. the base x,y.
- The convention used to order the list of images in a grid: for each character in the image name, type in an ’x’ for any character, a ’c’ for character and a ’d’ for digit.
- Decide whether characters specify columns or rows in the grid: for example in an image named tissue_A_002.tif, enter: ’xxxxxxxcxddd’.
- The overlap of the images on the horizontal and vertical sides. For example, if 1024 x 1024 images where taken with a 10% overlap in all directions, enter 102.4 for bottom-top and 102.4 for left-right.
- Whether images should be linked to their top/bottom/left/right neighboring images, if any (so they

can be dragged as one group by dragging any of its members).

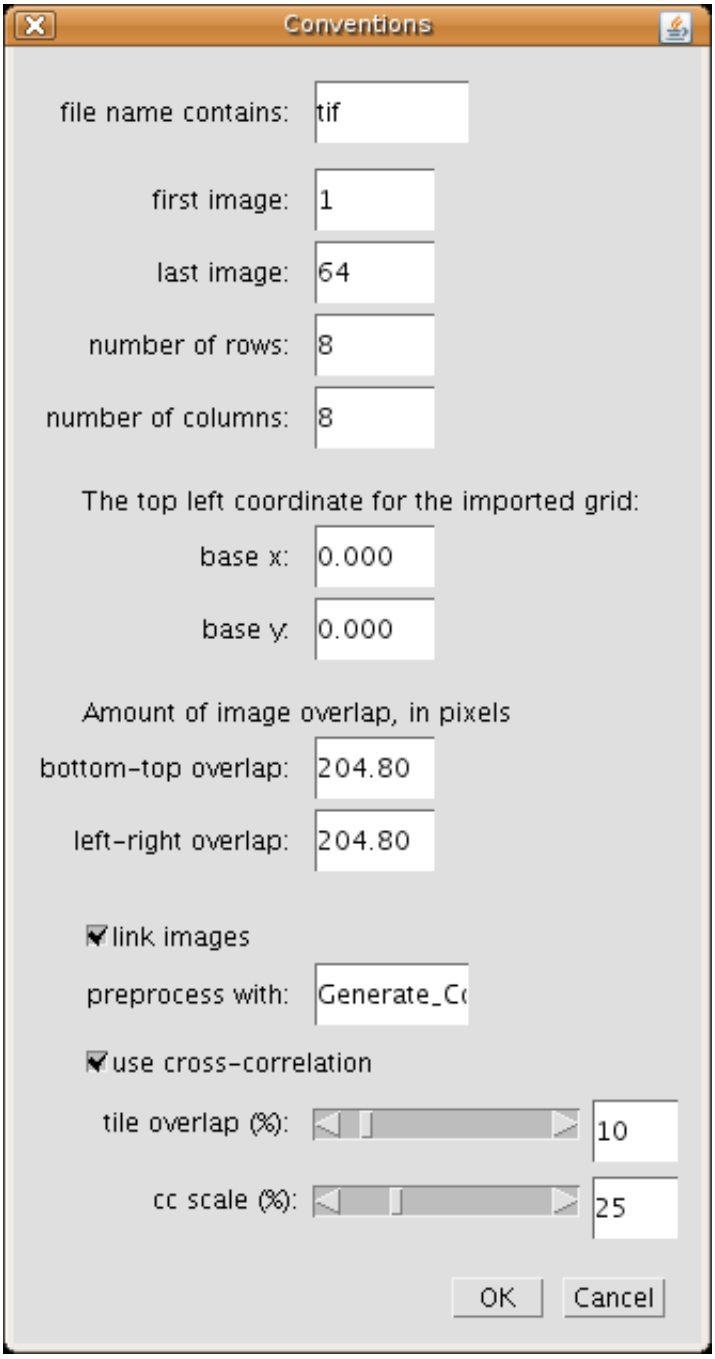
- The name of a PlugInFilter to run on every image prior to insertion. Single white space is tolerated. Any ImageJ commands or plugins that implement the PlugInFilter interface are accepted.
  Note for developers: be sure NOT to multithread the plugin, for the program won't wait on it. Also, the plugin should be resistant to null images in its setup(String arg, ImagePlus imp) method, since the program tests for the existence of the plugin by invoking it with a null image. Images are passed to the plugin by simply setting the image in question as the active image in ImageJ's WindowManager (see WindowManager.setTempCurrentImage(ImagePlus) method).
- Apply cross-correlation between neighboring tiles (left and top only) to correct improper overlaps. The two sliders define (1) the maximum amount of tolerated overlap, and (2)the scaling factor to apply to temporary images when cross-correlating, for speeding-up purposes.

Importing a sequence of images as a grid or montage

A sequence of images defined as image_001.tif, image_002.tif, etc., can be imported as a montage, by defining the number of columns and rows.
As in importing grids, there are three ways to add them:

- In the Layer Tree, right-click on a layer and select "Import sequence as grid...". This option has the advantage that it can be performed without a display showing, i.e. when finished, there is no delay related to repainting.
- In an open display, right-click and select "Import -> Import sequence as grid".
- Drag and drop a folder of images directly into the canvas.

After selecting an image from a folder, a dialog will appear asking for details.
All images will be imported assuming they have the same size as the selected image.



Options are:

- Filter file names in the folder by the sequence of common characters (optional).
- The first and last image to use for the grid. So that for example the first half can be imported, or the last half or third, etc.
- The number of rows and columns. In the example, images 1 to 22 will be in the first row, then images 23 to 44 in the second row, etc. The number of images to import has to be exactly the product of the

number of rows and columns.

- The top left coordinate of the grid (or montage) of images to import, i.e. the base x,y.
- The overlap of the images on the horizontal and vertical sides. For example, if 1024 x 1024 images where taken with a 10% overlap in all directions, enter 102.4 for bottom-top and 102.4 for left-right.
- Whether images should be linked to their top/bottom/left/right neighboring images, if any.
- The name of a PlugInFilter to run on every image prior to insertion. Single white space is tolerated. Any ImageJ commands or plugins that implement the PlugInFilter interface are accepted.
  Note for developers: be sure NOT to multithread the plugin, for the program won't wait on it. Also, the plugin should be resistant to null images in its setup(String arg, ImagePlus imp) method, since the program tests for the existence of the plugin by invoking it with a null image. Images are passed to the plugin by simply setting the image in question as the active image in ImageJ's WindowManager (see WindowManager.setTempCurrentImage(ImagePlus) method).
- Apply cross-correlation between neighboring tiles (left and top only) to correct improper overlaps. The two sliders define (1) the maximum amount of tolerated overlap, and (2) the scaling factor to apply to temporary images when cross-correlating, for speeding-up purposes.

Import images as listed in a text file

As simple as that: to import any number of images, list their names and X,Y,Z coordinates in a text file. The columns may be separated by space, tab or commas. Layers will be created automatically for any Z coordinate that doesn't have one already.

```
Image1.tif 0 0 30
Image2.tif 2048 0 30
Image3.tif 4096 0 30
...
```

The source folder where the images reside will be asked for.
Several parameters will be asked for, such as whether coordinates should be transformed by a defined factor; if histograms of all the images in each touched layer (as defined by the Z coordinate) must be histogram-adjusted; the thickness of the layers if they need be created; and whether all layers should be aligned when finished importing.

Importing rare image formats such as Zeiss LSM

To import rare image formats such as Zeiss LSM, EMMENU .dat files, etc., download ImageJ's plugin HandleExtraFileTypes to your plugins folder. Any image format is readable by ImageJ as long as a plugin is written for it, and the HandleExtraFileTypes is adjusted to call such plugin.
Then simply import images or stacks normally.

Semi-automated snapping: one image onto another

An image dragged onto another can be snapped to it by pressing alt+s while dragging, or right-click and select 'snap'.
A subset of the pixels will be analyzed for similarity on both images and then the dragged image is displaced for an optimal match.
When more than one image lay under the dragged image, the dragged image is aligned with the best matching image.

Adjusting images: ImageJ integration

Make sure the image is selected first. The panel in the Patches tab of the display will be highlighted in blue.
While selected, the image behaves like any other ImageJ image. You can run any command or plugin on it. For example, to adjust the brightness and contrast, just go to the menu "Image -> Adjust -> Brightness/Contrast..."
Any ImageJ key bindings will work as well. For example, to open the brightness and contrast dialog, push shift+c (or shift+control+c in some keyboards).
If the image is part of a stack, you will be given the option to adjust the entire stack at once, or just the selected image.

Transforming images

Make sure the image is selected and then push ’t’ to start the transformation.
Drag the handles from the bounding box to deform and rotate it.
Push ESCAPE to cancel, or ENTER to finish the transformation.
Rotation preserves pixel quality completely, by not rotating the data but only it’s screen representation.

Reverting images

Make sure the image is selected and then right-click and "Revert".
The original image is fetched from the database, and the modified copy is deleted.
For stacks, you are given the option to revert the entire stack at once.
Bear in mind that reverting an image restores its pixels, but not it’s screen representation. If you have transformed the image, you can restore it’s original width, height, rotation and position values values by selecting it and then right-click to adjust its "Properties..."

Color channel opacity

For RGB images, the "Opacity" tab in the display enables the control of the opacity of each individual color channel.
Simply select the channel by clicking its panel, and then drag the slider.
To completely switch off a channel, select/unselect its check box.
See a sequence of snapshots.

Profiles: create, adjust and transform

Create a profile

Make the layer you want to draw the profile into be displayed in the front most display. You can either double-click the layer from the Layer Tree to open a new display for it, or navigate your current display until finding it.
Drag a "profile" node from the Template Tree to a "profile list" node in the Project Tree. If the "profile list" already contains one profile, you must continue from that profile by selecting it and then right-click and select "Duplicate, link and send to previous/next layer".
The first point of the profile defines the layer in which it will exist.
Click and drag on the display, anywhere over the edge of the structure to outline. Drag towards where you are going to set the next point.

Now click and drag towards the previous point. The curve is drawn, and while dragging towards the first point the curvature is adjusted.
Proceed clicking and dragging towards the previous point.
Finally, click on the first point and drag towards the last, to close the curve while adjusting the curvature of the last segment.
All points can be dragged together by alt+drag on the area enclosed by the curve.

Adjust individual points in the Bézier curve

Select the pen tool from the ImageJ toolbar, if not already selected.
At any time, click and drag over the curve to insert a point between two existing points.
Shift+click any point to toggle the open/closed state of the curve.
At any time, click and drag the handles of any point to adjust the curvature.
Also, alt+drag a point to reset its handles.
To drag a point and its handles, simply drag the point.
To delete a point, shift+control+click it, or push 'x' to delete all points in the profile.

Adjust the Bézier curve by freehand drawing

New in version 0.3g
Select the pencil tool (the one on the left of the pen tool) in ImageJ's toolbar
. Click and drag on a point to draw freehand. Release the mouse to finish. The curve will close on the nearest point.
The slower the movement of the freehand the more bézier points will end up making the profile.
A profile can be drawn from scratch using this technique, no points need to exist beforehand.
[This functionality contributed by Mathias Buehlmann]
While the profile is unlinked (either it has not been deselected yet, or it has been actively unlinked), you can drag the whole curve with the black arrow tool.

Transform a profile

Make sure the profile is selected. When selected, the profile will show its points and control points besides the curve itself.
With the pen or the black arrow tool selected, double-click the profile. The bounding box will be drawn, with handles to deform it.
Pull the handles with the mouse to deform the profile. When done, double-click it to end the transformation.
Cancel the transformation anytime by pushing the ESC key.

Pipes: create, adjust and transform

Create a pipe

Make the layer you want to start drawing the pipe into be displayed in the front most display. You can either double-click the layer from the Layer Tree to open a new display for it, or navigate your current display until finding it.
Drag a "pipe" node from the Template Tree to an abstract node in the Project Tree that can contain it. You can proceed to add points by click-drag, adjusting the bézier curve, and then shift-drag, to adjust the radius

of the pipe at that point.
Navigate the layer set back and forth following the structure (an axon bundle, a dendrite) that you are modeling with a pipe. Add points to the same pipe in any layer.
The segments of the pipe resident in the layer will be linked to the images underneath, over which the pipe was drawn.

Adjust a pipe

Pipes exist in the entire layer set, not just in the current layer. Thus, only those pipe sections showing in the current layer are editable.
The pipe is a Bézier curve, and thus pipe points and control points can be adjusted like those of a profile, three differences withstanding.
First, a pipe cannot be closed. Second, the points of a pipe can exist in different layers. The segments of a pipe that exist in previous layers are painted in red, while those existing in deeper layers are painted in blue (see snapshot).
The points of a pipe that exist in the current layer are painted in the pipe's color.
Third, pipe points have a radius: by shift-drag on a point, you can increase and decrease the radius of the pipe at that point.
The radius of a new point is determined by the radius of the point that precedes it, unless manually edited.
Points in the current layer can be added either by insertion (click-drag near the medial line of the pipe) or by clicking anywhere, in which case they are appended at the end.

Transform a pipe

Just double-click the pipe and drag the handles of the bounding box. Double-click again to finish.

Balls: create, adjust and transform

Create a group of balls

A ball object is composed of many individual balls. The ball object is thus a group of many balls.
Drag and drop a "ball" node from the Template Tree to a node in the Project Tree that can contain it. A display needs to be open just as with profiles and pipes.
Each ball is represented by a triple x,y,z coordinate plus a radius. In the display, click anywhere to add a ball, or shift-click to add while adjusting its radius. Any subsequent balls will have the radius of the previous.
Navigate the layer set and add individual balls in any layer. Just like the coloring hints in pipes, balls existing in the immediately previous layer are colored in red, and those in the immediately next layer in blue.
This hints help in avoiding the repetition of balls that show partially in two consecutive layers.

Adjust individual balls

At any time, while a ball object is selected (will be labeled orange in the Project Tree, and with a blue background in the "Z space" tab), you can use the pen tool to add more individual balls.
Any ball in the current layer can have its radius be edited by shift-drag.
Delete an individual ball by control+shift+click, same way you would delete points for pipes and profiles.

Transform a ball

At any time, the entire ball object can be double-clicked with the black arrow tool and transformed by dragging the bounding box handles. The radius of the balls will be scaled along.

Labels: create and edit

Create a new label

Labels are independent of the Project Tree. You can create a label anytime anywhere.
Labels will also become linked to the underlying images when deselected, and they can be unlinked to be dragged elsewhere.
To add a label, select the Text tool from the ImageJ tool bar and click anywhere on the display. Enter as much text as you want in the window that opens.
When done, close the window. Only the first 10 characters or the first word will be painted on the screen.
While selected, the label background will be highlighted in a contrasting, semi-transparent color.
Before adding the label, the text font, size and style can be selected from the dialog opened by double-clicking on ImageJ's text tool.

Edit/Read a label

Labels can be read in full, and edited, by double-clicking them while selected. A text window will open, with the title set as that of the label. You can have as many open as you need.
Clicking an existing label with the Text tool will also open its text window for editing.
To edit the font family, size and style, right-click and select "Properties". See options at the bottom of the dialog.

Adjust/move/colorize a label

When unlinked, or not yet deselected for the first time, labels can be dragged with the black arrow tool just as any other floating object.
While selected, clicking on the color picker window will change the color of the text.
To adjust the font, size and style, right-click a selected label and choose "Properties".

AreaLists: create, adjust and transform

What is an AreaList

An AreaList consists of a number of painted areas anywhere on the entire layer set.
An AreaList exists in the entire layer set, not just in the current layer. So feel free to paint anywhere, while scrolling through the layers.

Create a new AreaList

First create a template object that contains an AreaList. As easy as right-clicking on any node in the Template Tree (leftmost tree) and selecting "new / area_list".
Once a template exists, create as many instances of it as you need in the Project Tree. Simply by drag and drop, or right-click on the appropriate node in the Project Tree and select "new / area_list".
The PEN tool will be selected, the display will come to the front and the new AreaList will be selected

(blue background) under the Z-space tab.
Like any other objects, the AreaList will become linked to any underlying images both when deselected and when scrolling through layers.

Adjust a new AreaList

Be sure to have the PEN tool selected in ImageJ's toolbar (mouse over a tool will tell you its name), then:
With the left mouse button:

- Drag to paint.
- Alt+drag to erase.
- Shift+click to fill a hole.
- Shift+alt+click to erase a filled area.

The diameter of the brush is adjusted by either of:

- Shift+scrollwheel.
- Double-click on the oval/brush ROI tool, and adjust the brush value.

In addition, right-click and choose "Properties" from the popup menu to adjust several values, including an option to paint the AreaList as outlines instead of as filled areas.

Transform an AreaList

Make sure the AreaList is selected and then push 't' or right-click and select "Transform" from the popup menu.
Any linked objects will be transformed along with the AreaList.

Merge two or more AreaList

Select as many AreaList objects as you want, and right-click and choose "Merge" from the popup menu. The active object in the selection will be kept as a base onto which all others are merged. Superfluous, empty parent nodes in the Project Tree will be removed.

Dissectors: count objects only once

What is a Dissector

A 'Dissector' is a technique for counting only once objects that appear in multiple sections of a three-dimensional tissue. There are numerous variants. The Double Dissector technique, along with detailed statistical analysis detailing when to stop counting, how to sample a large dataset properly, etc., is explained in this paper:
Geinisman, Y., Gundersen, H.J.G., van der Zee, E. and West, M.J. 1996. Unbiased stereological estimation of the total number of synapses in a brain region. Journal of Neurocytology 25:805-19.

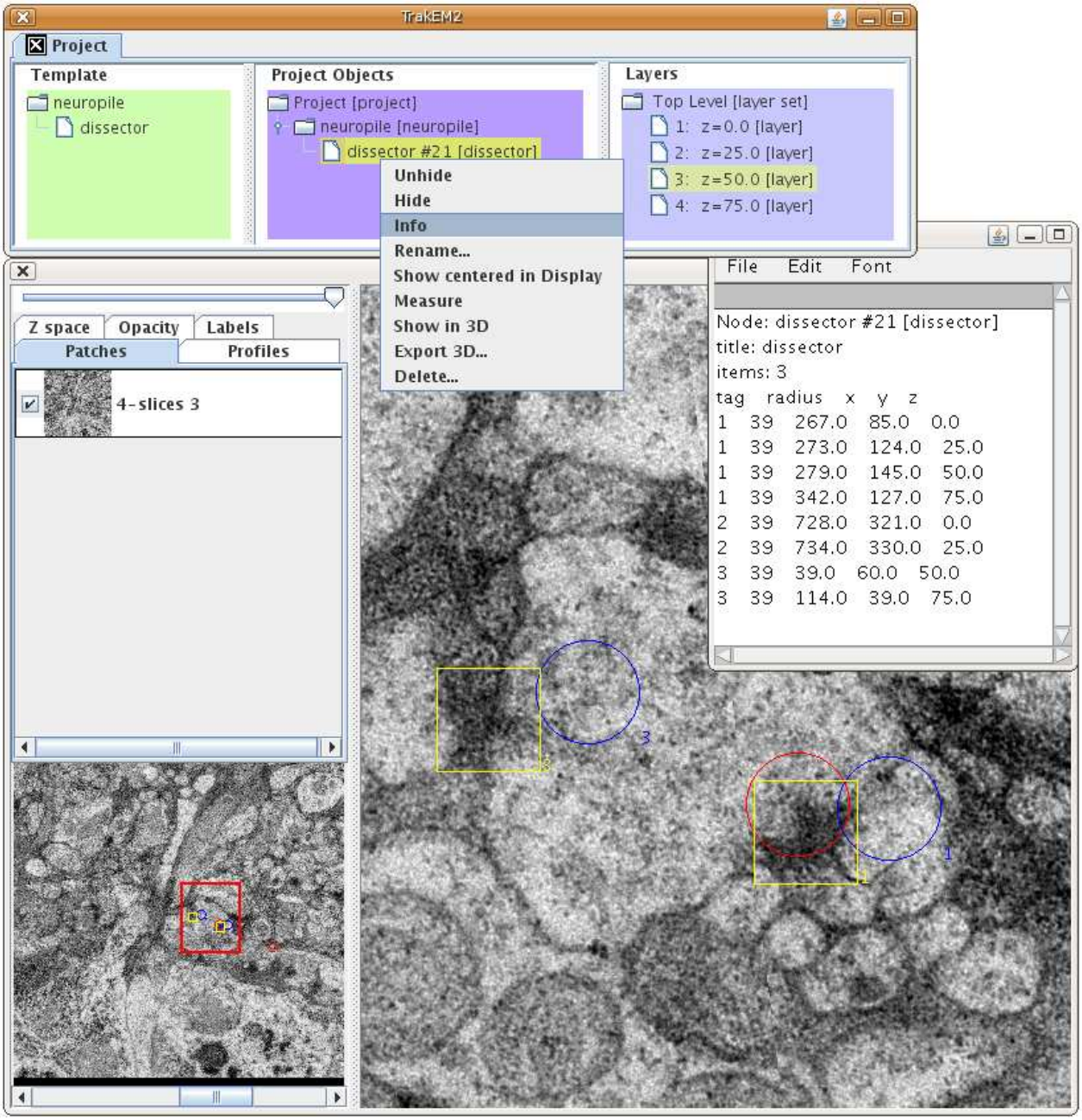
Create a new Dissector

First create a template object that contains a Dissector type. As easy as right-clicking on any node in the Template Tree (leftmost tree) and selecting "new / dissector".
Once a template exists, create as many instances of it as you need in the Project Tree. Simply by drag and drop, or right-click on the appropriate node in the Project Tree and select "new / area_list".
The PEN tool will be selected, the display will come to the front and the new dissector will be selected (blue background) under the Z-space tab.
Like any other objects, the Dissector object will become linked to any underlying images both when deselected and when scrolling through layers.

Adjust/edit/construct a dissector

A dissector object consists of a group of items to count, each item consisting of a set of marks in consecutive layers, at a reason of a mark per layer.
When scrolling through layers, marks in the previous layer appear in red, and marks in the next layer appear in blue. Such color hints enable you to know where have you added already a mark in the previous and/or next layer, and thus whether a new item should be added, or just a mark for an existing item.
Be sure to have the PEN tool selected in ImageJ's toolbar (mouse over a tool will tell you its name), then:
With the left mouse button:

- Click outside any existing marks to add a new mark, starting a new item to count.
- Click on the red or blue ghost of a mark to add a new mark for that item.
- Drag to move a mark.
- Shift+drag a mark to resize all marks for an item.
- Shift+control+click to erase a mark.



[click to enlarge]

Analyze a dissector

All information regarding a dissector is obtained by right-click on it's project node and selecting 'Info'. A text window will open containing the number of items counted, and then a detailed list of all marks for all items. The data list is spread in columns for easy processing in a spreadsheet.
To measure the area of a dissector, use the Rectangular ROI tool (leftmost tool in ImageJ's toolbar) and then push 'M' or Analyze -> Measure. If you had calibrated the project first (with Analyze -> Set Scale), then the measured area will be given in microns, meters, or any units you entered.

Eventually statistical tools will be built in.

Object manipulation: select, move, link, unlink, transform

Selecting

Select a profile, image, etc. by clicking it with the black arrow, or selecting it from the list under the proper tab to the left of the display window.
Alternatively, you can shift+click on a profile, image, etc. and make a multiple selection.

1. Click to select
2. Shift+click to add to a selection
3. Shift+click a selected object to make it the active object in the selection
4. Shift+click the active object in a selection to deselect it

If more than one object exists under the clicked point, a popup will list all the possible selectable objects from which you can choose the profile.
When selected, profiles, pipes and balls will be highlighted in the Project Tree.
Any selected object will be shown in the proper display tab, with its background in blue (if it's the active object) or in pink.

Deselecting

Push the ESC key to deselect all.

Move above or below one another

Floating objects within the same layer can overlay one another. The order in which they overlay each other can be changed.
The contextual menu either over the selected floating object or over its snapshot panel provides entries to move up, down, top and bottom.
As of version 0.3b, these key bindings are available:

- Home: move to top
- Page up: move up
- Page down: move down
- End: move to bottom

Dragging

Once and object is selected, you can drag it with the black arrow tool.
If the object is linked to images, and thus maybe indirectly to other objects, the entire group of linked elements will be dragged together.
When working with stacks, the linked group **includes** objects in other layers. That is, the other images in the stack and any other object linked to them.
Non-image objects can be unlinked and then dragged independently.

Linking

Linking is automatic.
Linking is the binding of a floating object to an image, or an image to another image. For example, the moment a profile is unselected it becomes automatically linked to whatever images lay under it, in that layer. The same speaks for labels, pipes and balls.
What linking means is: the floating object is supported by the image to which it is linked.
That is why linking occurs: if it didn't, the moment an image or a profile was dragged from its position, it would become meaningless.
Links are also used to preserve pre-knowledge on the arrangement of images:

- When importing a stack, all images are chain-linked, that is, every slice is linked to the next.
- When importing a grid or montage of images, options are given to link each image with its immediate neighbors.

In any case images within the same layer can be indirectly linked to each other if a floating object, for example a profile, is draw over both.

Unlinking

All floating elements in a Layer can be unlinked. The images, if linked to other images because they are part of a stack, cannot be unlinked (would be meaningless and would introduce artefactual relationships between data).
To unlink, simply select the object and right-click. From the contextual menu select "Unlink". The object will remain unlinked until it becomes deselected (another object is selected, or ESC is pushed, or an empty area of the layer is clicked).

Transforming

Any floating object, including images, can be selected and then transformed by pushing 't' or selecting "Transform" from the right-click menu, with the black arrow tool selected.
While transforming, selected objects cannot be transformed in any other display.
While transforming, selected objects are drawn with their compound bounding box and a set of handles that are used to stretch and deform it.
Push the ESCAPE key to cancel a transformation.
Push ENTER to end the transformation.
Alternatively, use the contextual menu (right-click) to apply or cancel a transformation.
Transformations are most powerfully used in combination with the undo.
Rotation preserves pixel quality completely, by not rotating the data but only it's screen representation.
Abstract objects such as profiles, pipes and balls, are rotated by rotating their data with double floating-point precision.
The radius of pipes and balls is preserved relative to the most dramatically changing axis when the scaling is not proportional.

Adjusting transparency and visibility

Any floating object can be set transparent, or totally invisible, whenever convenient.
When the object is selected, drag the slider to adjust its transparency.
In its proper tab, set the check box selected or unselected to make the object visible or invisible, respectively.
The transparency is preserved when the visibility is toggled with the check box.
As a convenient shortcut for manual montaging, select an image and push the space bar to set its transparency to 50%. Push again to reset to 100%.

Hide / Unhide all of a class

Any time right-click and select the proper entry in the "Hide/Unhide" menu item.
Images cannot be hidden massively, but only individually by toggling their check boxes or setting their transparency to zero.

Lock

Locking means to prevent an object from being displaced.
Right-click an object and select "Lock". All objects linked to the locked object will be locked as well.
To unlock, select the locked object, or any in its linking group, and right-click and select "Unlock".

Object properties

Right-click a selected object, or its panel in the tabs, and select "Properties". Width, height, position, visibility and locking state are all adjustable here, including specific options for each kind of object such as fonts for labels.

Deleting profiles, pipes and ball objects

Basic data types such as profiles, pipes and balls cannot be deleted on the display.
The currently selected profile, pipe or ball object in the front display will be shown selected in orange in the Project Tree.
Go to the Project Tree and right-click the node to delete, and select "Delete...".

Undo/Redo transformations

At any given time, push shift+z to undo the last transformation (up to 40 steps are recorded).
If no translation/rotation/scaling has been applied after undoing one or more steps, push alt+z to redo all the way to the very last transformation.
If after undoing one or more steps a new transformation is done, the undo history is truncated at that particular point.

Search

Both on any open display or on a layer set node in the layer tree, right-click and select "Search...". A dialog will come up with a text search box. Any text contained in the project objects (image file names, labels, etc.) is searchable.
The search can be constrained to only labels, or only images, etc., by selecting the appropriate type from

the pulldown menu.
The search is case insensitive and supports regular expressions. For example, to search for all labels containing the words "dorsal" and "lobe" in this order, type:

```
dorsal.*lobe
```

The dot "." stands for "any character" and the "*" acts as a modifier and means "any number of times". See the regular expression syntax documentation for all details.
A successful search will create a list of found objects. Double-click any row to center the object in an opened display.
Subsequent searches will coexist in new tabs as long as the Search window is open, and will be updated when objects are deleted, but not when new objects potentially matching the search are added to the project.

Adding functionality with your own plugins

How to run plugins on a selected image

Open a display, select an image, and simply run any ImageJ plugin on it. If the pixels are modified, the modified image will be saved along with the original image.
From a plugin, just call the active image from ImageJ's WindowManager:

```
import ij.WindowManager;
import ij.ImagePlus;
/* ... */
    ImagePlus imp = WindowManager.getCurrentImage();
    // process the image ...
```

The above works because TrakEM2 has set the selected image as the active image in the WindowManager, via the setTempCurrentImage(ImagePlus) method.

How to run plugins on many objects in an open display

To retrieve all selected objects in a Display, and modify them from an ImageJ plugin, follow this example:

```
import ini.trakem2.display.*;
import java.awt.Rectangle;
import java.awt.geom.AffineTransform;
import java.util.*;
import ij.plugin.PlugIn;
import ij.ImagePlus;

public class My_Plugin implements PlugIn {

    public void run(String arg) {
        Display front = Display.getFront();
        if (null == front) return; // no open displays
        Selection selection = front.getSelection();
        if (0 == selection.getNSelected()) return; // no selected objects
        // store the area enclosing all selected objects and their links
        Rectangle box = selection.getLinkedBox();
        // get the Transform for each selected object
```

```
        Hashtable ht = selection.getTransformationsCopy();
        // add undo step
        front.getLayer().getParent().addUndoStep(ht);
        // modify each Displayable
        for (Iterator it = ht.keySet().iterator(); it.hasNext(); ) {
            Displayable d = (Displayable)it.next();
            /* Do whatever to the Displayable object, for example */
            /* displace it 40 pixels down and 100 pixels left,
            * not propagating to linked displayables */
            d.translate(-40, 100, false);
            /* Rotate by 45 degrees relative to pivot point 200, 200,
            * not propagating to linked displayables */
            d.rotate(45, 200, 200, false);
            /* Scale  by 2.0 in the X axis and 1.5 in Y axis,
            * relative to pivot point 200, 200
            * and not propagating to linked displayables. */
            d.scale(2.0, 1.5, 200, 200, false);
            /* For more complex transformations,
            * just grab the AffineTransform directly: */
            AffineTransform at = d.getAffineTransform();
            /* ... process AffineTransform ... */
            /* and don't forget to save it: */
            d.updateInDatabase("transform");
            //
            /* Do whatever to the image, if the Displayable is an image */
            if (d.getClass().equals(Patch.class)) {
                ImagePlus imp = d.getProject().getLoader().fetchImagePlus(d);
                imp.setProcessor(imp.getProcessor().invert());
            }
        }
        // update selection internals in all open Displays
        Display.updateSelection(front);
        // repaint all displays showing the same Layer as the front Display
        box.add(selection.getLinkedBox());
        Display.repaint(front.getLayer(), box, 0);
    }
}
```

To modify all objects in a Layer, do something like:

```
Display front = Display.getFront();
Layer layer = front.getLayer();
ArrayList al = layer.getDisplayables();
for (Iterator it = al.iterator(); it.hasNext(); ) {
    Displayable d = (Displayable)it.next();
    // do whatever to the Displayable, such as displacing it
    d.translate(-40, 100, true);
}
// update selection internals in all other open Displays and repaint them
Display.updateSelection(front);
// repaint (in this case, the whole area, but could be optimized to repaint less)
Display.update(layer);
```

To modify all objects in a LayerSet, call first each layer, and then all its objects:

```
Display front = Display.getFront();
LayerSet set = front.getLayer().getParent();
for (Iterator it = set.getLayers().iterator(); it.hasNext(); ) {
    Layer layer = (Layer)it.next();
    for (Iterator it2 = layer.getDisplayables().iterator(); it2.hasNext(); ) {
        Displayable d = (Displayable)it2.next();
        /* ... */
    }
}
// update selection internals in all open Displays
Display.updateSelection(front);
// repaint all Displays showing a Layer of the edited LayerSet
Display.update(set);
```

How to create an undo step

To add an undo step before running any of the above, create one for a LayerSet like this:

```
Hashtable undo = new Hashtable();
LayerSet set = front.getLayer().getParent();
for (Iterator it = set.getLayers().iterator(); it.hasNext(); ) {
    Layer layer = (Layer)it.next();
    for (Iterator it2 = layer.getDisplayables().iterator(); it2.hasNext(); ) {
        Displayable d = (Displayable)it2.next();
        undo.put(d, d.getTransform()); // getTransform() returns a copy
    }
}
set.addUndoStep(undo);
```

Of course, the above can be created for any subset of Displayable objects that belong to the given LayerSet, such as all Displayable objects of a Layer.
If you are interested in modifying the selected objects only, then there is a shortcut:

```
        Selection selection = Display.getFront().getSelection();
        // get the Transform for each selected object
        Hashtable ht = selection.getTransformationsCopy();
        // add undo step
        front.getLayer().getParent().addUndoStep(ht);
        // proceed to modify the Transform of selected objects
        // ...
```

How to retrieve objects when no displays are open

In this case, call the project directly and retrieve its root LayerSet:

```
import ini.trakem2.Project;
import ini.trakem2.ControlWindow;
import java.util.Set;
import java.util.Iterator;
/* ... */
Set projects = ControlWindow.getProjects();
if (null == projects) return; // no open projects
for (Iterator it = projects.iterator(); it.hasNext(); ) {
```

```
    Project pr = (Project)it.next();
    LayerSet root_set = pr.getRootLayerSet();
    /* ... */
}
```

How to create your own custom image file importer, and then stitch all tiles and register all layers

The code below scans a folder for files whose name starts with ’r’ and ends with ’tif’, and then classifies files according to a predefined naming convention as explained in the header. Then a new project is created programmatically.
There are 4 tiles per grid, and each grid results then in a montage being stitched and inserted into a Layer. Finally, all layers are registered one to the next.
The code below can be run as a regular ImageJ plugin.
If you need headless support, just add ControlWindow.setGUIEnabled(false); somewhere at the beginning, and of course provide the source folder and the layer thickness directly. In addition, you can programmatically save the project with project.getLoader().save(project); which would popup a file saver dialog unless you previously provide the path directly with Macro.setOptions("path=’/some/path/my_project.xml’");
Creating snapshots of the whole layer is just as easy. See second code box below.

```
import ini.trakem2.*;
import ini.trakem2.persistence.*;
import ini.trakem2.display.*;
import ini.trakem2.imaging.Registration;
import ini.trakem2.utils.*;

import ij.*;
import ij.gui.GenericDialog;
import ij.io.*;
import ij.plugin.PlugIn;

import java.io.File;
import java.io.FilenameFilter;
import java.util.*;

/* Import image files with the following naming convention:

   r2-2 is grid row 2 col 2 and abcd are four images that cover NW NE SE SW

   r2-2 13500x a.tif  r3-2 13500x c.tif  r4-3 13500x a.tif  r5-4 13500x c.tif
   r2-2 13500x b.tif  r3-2 13500x d.tif  r4-3 13500x b.tif  r5-4 13500x d.tif
   r2-2 13500x c.tif  r3-3 13500x a.tif  r4-3 13500x c.tif  r5-5 13500x a.tif
   r2-2 13500x d.tif  r3-3 13500x b.tif  r4-3 13500x d.tif  r5-5 13500x b.tif
   r2-3 13500x a.tif  r3-3 13500x c.tif  r4-4 13500x a.tif  r5-5 13500x c.tif
   r2-3 13500x b.tif  r3-3 13500x d.tif  r4-4 13500x b.tif  r5-5 13500x d.tif
   r2-3 13500x c.tif  r3-4 13500x a.tif  r4-4 13500x c.tif  r5-6 13500x a.tif
   r2-3 13500x d.tif  r3-4 13500x b.tif  r4-4 13500x d.tif  r5-6 13500x b.tif
   r2-4 13500x a.tif  r3-4 13500x c.tif  r5-1 13500x a.tif  r5-6 13500x c.tif
   r2-4 13500x b.tif  r3-4 13500x d.tif  r5-1 13500x b.tif  r5-6 13500x d.tif
   r2-4 13500x c.tif  r4-1 13500x a.tif  r5-1 13500x c.tif  r5-7 13500x a.tif
   r2-4 13500x d.tif  r4-1 13500x b.tif  r5-1 13500x d.tif  r5-7 13500x b.tif
   r3-1 13500x a.tif  r4-1 13500x c.tif  r5-3 13500x a.tif  r5-7 13500x c.tif
   r3-1 13500x b.tif  r4-1 13500x d.tif  r5-3 13500x b.tif  r5-7 13500x d.tif
   r3-1 13500x c.tif  r4-2 13500x a.tif  r5-3 13500x c.tif
   r3-1 13500x d.tif  r4-2 13500x c.tif  r5-3 13500x d.tif
```

```
   r3-2 13500x a.tif  r4-2 13500x d.tif  r5-4 13500x a.tif
   r3-2 13500x b.tif  r4-2 13500x b.tif  r5-4 13500x b.tif

   Tiles order in the grid is:
   d c
   b a

   Put each a,b,c,d set in a layer
   Stitch tiles together
   Register layers

 *
 */
public class T2_import_custom_data implements PlugIn {


        public void run(String arg) {

                try {

                // 1 - Select source folder
                final DirectoryChooser dc = new DirectoryChooser("Choose source folder");
                final String source_dir = dc.getDirectory();
                if (null == source_dir) return; // user canceled the dialog

                // 2 - Ask for layer thickness
                final GenericDialog gd = new GenericDialog("Properties");
                gd.addNumericField("section thickness (pixels): ", 25, 2);
                gd.showDialog();
                if (gd.wasCanceled()) return;
                final double thickness = gd.getNextNumber();

                // 2 - Obtain source images
                final String[] tifs = new File(source_dir).list(new FilenameFilter() {
                        public boolean accept(File dir, String name) {
                                name = name.toLowerCase();
                                if (name.startsWith("r") && name.endsWith(".tif")) {
                                        return true;
                                }
                                return false;
                        }
                });
                if (0 == tifs.length) {
                        IJ.showMessage("No images found.");
                        return;
                }

                // 3 - Sort tiles by grid
                final Hashtable ht = new Hashtable();
                for (int i=0; i<tifs.length; i++) {
                        String grid_tag = tifs[i].substring(0, 4);
                        Object tiles = ht.get(grid_tag);
                        if (null == tiles) {
                                tiles = new ArrayList();
                                ht.put(grid_tag, tiles);
                        }
                        ((ArrayList)tiles).add(tifs[i]);
                }

                // 4 - check data integrity for assumptions below to work
                for (Iterator it = ht.entrySet().iterator(); it.hasNext(); ) {
                        Map.Entry entry = (Map.Entry)it.next();
```

```
        String grid_tag = (String)entry.getKey();
        ArrayList tiles = (ArrayList)entry.getValue();
        if (4 != tiles.size()) {
                YesNoDialog yn = new YesNoDialog("WARNING",
                "Grid " + grid_tag + " has " + tiles.size()
                + " tiles instead of the expected 4.\nProceed anyway?");
                if (!yn.yesPressed()) return;
        }
}

// 5 - sort grid tags
final String[] grid_tags = new String[ht.size()];
ht.keySet().toArray(grid_tags);
Arrays.sort(grid_tags);

// enable Macro.getOptions()
Thread.currentThread().setName("Run$_Custom_Import");

// 6 - insert each grid into a layer, and stitch its images
Macro.setOptions("path='/home/albert/projects/'");
final Project project = Project.newFSProject("blank");
try {
        Thread.sleep(500); // waiting for Swing's asynchronous calls
} catch (InterruptedException ie) {
        ie.printStackTrace();
        return;
}
final LayerSet layer_set = project.getRootLayerSet();
// fixing first layer, so it'll be found equal and thus returned
layer_set.getLayer(0).setThickness(thickness);
for (int i=0; i<grid_tags.length; i++) {
        double z = i * thickness;
        // creates a new Layer if not there yet
        Layer layer = layer_set.getLayer(z, thickness, true);
        // name the Layer
        project.findLayerThing(layer).setTitle(grid_tags[i]);
        ArrayList tiles = (ArrayList)ht.get(grid_tags[i]);
        String[] tile_names = new String[tiles.size()];
        tiles.toArray(tile_names);
        Arrays.sort(tile_names);
        // reverse order, so that we end up with 'd c b a'
        for (int k=0; k<tile_names.length/2; k++) {
                String tmp = tile_names[k];
                tile_names[k] = tile_names[tile_names.length -k -1];
                tile_names[tile_names.length -k -1] = tmp;
        }
        // prevent GenericDialog by filling in all options as macro keys
        Macro.setOptions("file_name_matches='" + grid_tags[i]
        + ".*' first_image=1 last_image=" + tile_names.length
        + " number_of_rows=2 number_of_columns=2 base_x=0 base_y=0"
        + " bottom-top=256 left-right=256 link preprocess=''"
        + " use_cross-correlation homogenize_contrast"
        + " tile_overlap=10 cc_scale=25");
        Utils.log2(Macro.getOptions());
        final Thread t = project.getLoader().importSequenceAsGrid(layer,
                                                source_dir, tile_names);
        if (null == t) return;
        try {
                t.join();
        } catch (InterruptedException ie) {}
        // just in case
        project.getLoader().releaseMemory();
```

```
                }

                // just in case the laptop has little RAM
                project.getLoader().releaseMemory();
                Loader.runGC();

                // 7 - register layers (rather lower stringency settings)
                // prevent GenericDialog by filling in all options as macro keys
                Macro.setOptions("scale=50 steps_per_scale_octave=3"
                + " initial_gaussian_blur=2.0 feature_descriptor_size=8"
                + " feature_descriptor_orientation_bins=8 minimum_image_size=64"
                + " maximum_image_size=2000 minimal_alignment_error=2.0"
                + " maximal_alignment_error=200 inlier_ratio=0.05");
                final Thread b = Registration.registerLayers(layer_set, 0, 0,
                                                        layer_set.size() -1, false);
                if (null == b) return;
                try {
                        b.join();
                } catch (InterruptedException ie) {}

                // reset
                Macro.setOptions(null);

                System.out.println("Done!");
                IJ.log("Done!");

                } catch (Exception e) {
                        e.printStackTrace();
                        IJ.showMessage("Some error occurred. Please see the terminal output.");
                }
        }
}
```

How to make image snapshots of layers

The idea is to not include parts of the layer that are empty. So I create a Selection and add to it all Displayable objects of a Layer, then retrieve the bounding box and with it call the Loader to make a flat image scaled to 1/8th (that is, 0.125 magnification).

```
        Layer layer = ...
        // determine bounding box
        final Selection sel = new Selection(null);
        for (Iterator it = layer.getDisplayables().iterator(); it.hasNext(); ) {
                sel.add((Displayable)it.next());
        }
        final Rectangle box = sel.getBox();
        // of course the box above could be created much easily, including images only:
        //  Rectangle box = layer.getMinimalBoundingBox(Patch.class);
        // make snapshot and save it separately
        Utils.log("Going to paint flat image.");
        final ImagePlus flat_section = project.getLoader().getFlatImage(layer,
                box, 0.125, 0xffffffff, ImagePlus.GRAY8, Patch.class, null, true);
        new ij.io.FileSaver(flat_section).saveAsTiff(snapshots_dir
                                        + grid_tag + "_" + sec_tag + ".tif");
```

Using TrakEM2 as a framework

How to create your own automatic segmentation program

The two main elements are:

- How to retrieve the pixels for any arbitrary region of interest of a particular layer.
- How to store segmentations using TrakEM2's AreaList native type.

First you need a Project and its root LayerSet either from an open project:

```
Project project = ControlWindow.getActive();
LayerSet layer_set = project.getRootLayerSet();
```

or created new from scratch:

```
ControlWindow.setGUIEnabled(false); // headless mode, optional
Thread.currentThread().setName("Run$_do_it"); // enable Macro.getOptions()
Macro.setOptions("path='/home/albert/projects/'"); // provide storage folder
Project project = Project.newFSProject("blank");
LayerSet layer_set = project.getRootLayerSet();
```

Now, to retrieve arbitrary pixel rasters, call:

```
// getting the third layer in the set (other methods available, such as
// by id, from an iterator on the LayerSet.getLayers(), or by Z coordinate):
Layer layer = layer_set.getLayer(3);

// parameters:
Rectangle box = ... // the region you want, in offscreen world coordinates
                    // (i.e. relative to the LayerSet system of coordinates)
double scale = 1.0; // the desired magnification
int channels = 1; // the desired color channels: bit-shifted 0-255 values
//                                               for Blue-Green-Red
int type = ImagePlus.GRAY8; // GRAY8 or COLOR_RGB only
Class clazz = Patch.class; // specify images only (Patch class encapsulates an image)

ImagePlus imp = project.getLoader().getFlatImage(layer, box, scale, channels,
                                                 type, clazz, true);

// the ImagePlus is a native ImageJ type: you can process it with standard plugins.
// or just get its processor and its pixels:
ImageProcessor ip = imp.getProcessor();
if (imp.getType() == ImagePlus.GRAY8) {
        byte[] pixels = (byte[])ip.getPixels();
}
// see ImageJ API for all supported image types,
//     with their associated pixel array types.
```

Now, after processing all desired image regions and generating segmentation profiles, it's time to store them as AreaList. An AreaList is a wrapper for a list of java.awt.geom.Area (at a reason of one per Layer), and thus any class implementing the java.awt.Shape interface (such as Polygon, Area, Rectangle ...) can be added to it. Or any ImageJ ROI (by first making a ShapeRoi and then extracting the GeneralPath from it with reflection). When directly adding an Area, you must tell AreaList which layer it goes to:

```
Hashtable<Layer,Shape> profiles = ...; // your own data

AreaList ali = new AreaList(project, "the name", 0, 0);

for (Iterator it = profiles.entrySet().iterator(); it.hasNext(); ) {
        Map.Entry entry = (Map.Entry)it.next();
        Layer layer = (Layer)entry.getKey();
        Shape profile = (Shape)entry.getValue();
        //
        ali.setArea(layer.getId(), new Area(profile));
}

// be sure to make the data local, for best compactability
ali.calculateBoundingBox();

// don't forget to actually add it to the LayerSet!
layer_set.add(ali);

// and also to the ProjectTree:
ArrayList<AreaList> all_arealists = ... ; // your data
project.getProjectTree().insertSegmentations(project, all_arealists);
```

Above, we are storing the AreaList the lazy way, into a single node. If you'd like to store them in a tree structure, then you have to programmatically create template nodes and project nodes.
Finally, you can save the project programmatically if you want:

```
// enable Macro.getOptions() to avoid dialogs:
Thread.currentThread().setName("Run$_custom_segmentator");
Macro.setOptions("path='/some/path/my_project.xml'");
project.getLoader().save(project);
```

Appendix

Recording an ImageJ macro: recording the opening of a project and assigning a keyboard shortcut

In the ImageJ menu, select Plugins -> Macros -> Record... to open the recorder. Then perform all the actions you want to record, in sequential order. Finally, push the "Create" button and save the macro in the ImageJ/macros directory.
To record the opening of a particular project from a particular database, simply open the recorder prior to opening the project. Then, you'll see a command recorded, similar to:

```
run("Open DB Project", "host=164.43.04.243 port=5432 database=www user=anonymous
                            password=mypassword project=[Drosophila engrailed]");
```

Now simply:

- Write in a meaningful name, other than 'Macro'. For example the name of the project, "Drosophila_engrailed". Be sure to include an underscore in the name.
- Push the "Create" button.
- In the text window that opens, save the macro in the ImageJ/plugins folder, under the folder of your choice (such as "My Projects").

Done! Next time ImageJ is restarted, the project will be listed under the menu "Plugins -> My Projects -> Drosophila engrailed".
Alternatively, the macro can be installed immediately by selecting "Plugins -> Macros -> Install..." and it will appear under the "Plugins -> Macros" menu.
To assign a keyboard shortcut, just select "Plugins -> Shortcuts -> Create shortcut" and select the "Drosophila engrailed" from the pulldown list.
If the key is already selected, check the "Plugins -> Utilities -> List shortcuts" for available shortcuts.
Most shift+<key> are available, and also the F1, F2, etc. keys.

Opening the color picker window

Simply double-click the color dropper tool in the ImageJ toolbar.
Clicking on a color on the color picker window will edit the drawing color of the active floating object in the front most display.
So the color can be customized for text in labels, the curve in profiles, pipes and balls, and the transformation box of images.

Open XML projects by drag and drop to the ImageJ toolbar

Install the ToggleOpenUsingPlugins macro and push the 'F1' key to activate it, or, alternatively, paste its essential contents into your StartupMacros.txt in ImageJ's macros folder. If the file doesn't exist within your macros folder, just create it with any text editor, then write:

```
// Enable drag and drop of XML files to be opened by TrakEM2
setOption("OpenUsingPlugins", true);
```

In any case you will need a modified HandleExtraFileTypes.java plugin, compiled, in your ImageJ plugins folder. The modification is as follows:

```
// ****************** MODIFY HERE ******************

// try to see if it's a TrakEM2 file
if (name.endsWith(".xml")) {
        if (-1 != new String(buf).toLowerCase().indexOf("trakem2")) {
                try {
                        ini.trakem2.Project.openFSProject(directory + "/" + name);
                        width = IMAGE_OPENED; // signal all is ok
                } catch (Exception te) {}
                return null;
        }
}
```

For the above to work, the size of the read buffer should be increased a bit; a safe size is 136 bytes in length (see the modified HandleExtraFileTypes.java, line 69).
Beware that ImageJ 1.38f or higher is required.
As a side effect, you will now be able to open TrakEM2 XML files from the standard File/Open menu as well.

How to find out which TrakEM2 version you are running

Select Help / About Plugins / About TrakEM2 from the menus, and the version number will be printed in a dialog.
If you cannot see ImageJ's "Help" menu, enlarge ImageJ's window by pulling it's bottom right corner.

Open .dat (EMMENU) and .mrc (Leginon) files with ImageJ

- Leginon-generated raw .mrc files (32-bit Real) importer (compiled version)
- EMMENU-generated raw .dat files (16-bit Unsigned) importer (compiled version)

If you place this HandleExtraFileTypes.java (compiled version) file in your ImageJ plugins folder, TrakEM2 will be able to read such files seamlessly (from "Import image", "Import stack" and by drag and drop into a display). And, of course, ImageJ will be able to read them from the "File / Open" menu, and by drag and drop into the tool bar.