M I E N

Model Interaction Environment for Neuroscience

Version 1.0

October 2008

Contents

What Is MIEN?

MIEN (Model Interaction Environment for Neuroscience) is a suite of programs written in Python to assist with the analysis of neuroanatomical and time series data.

MIEN generates, edits, stores, and evaluates abstract mathematical models and realistic compartmental neural models. The core of the system is an XML dialect for specifying the models, parameters, experiments, data, connections, etc., and a set of python classes for providing these descriptions with useful functions.

Two of the components of MIEN are **DataViewer** and **CellViewer**.

DataViewer is a graphical front-end for viewing and processing time series data. A variety of visualization modes are supported. The GUI is designed for easy addition of user-written data processing functions. Extensions are already included for signal processing, information theory, and neuron spike detection. The system can read and write files in Matlab 6, text, DataMAX, simple binary, Datastreamer, and its own internal format.

CellViewer is a GUI for visualizing morphological models of neurons. It can also be used to edit and mathematically transform morphological data. It supports the Neurolucida (.asc) format as well as BBT, XML, and Neuron Hoc (export only).

System Requirements

Operating System: OS Portable

If building from source, current versions of Python, Numerical Python (NumPy), OpenGL (PyOpenGL), and wx (wxPython) are required.

How do I use MIEN?

Starting MIEN

Open terminal window and run MIEN from the command line. To start the top-level GUI interface, type "mien" at the prompt.

However, most users may want to start by opening a specific component of MIEN, such as DataViewer or CellViewer. To do so, you will need to pass the -a option (for "application"), followed by an appropriate value. Example: "mien -a cell" will open CellViewer; "mien -a data" will open DataViewer.

Other options for running MIEN are explained below.

mien [-h|v|t|p|a app|r method|s depth|b fname|c type] [-f format] [-i] [file [file ...]]

-h Help. Display this list of options.

-v Visualize. In this mode, MIEN will attempt to automatically detect the type of the input file(s) and open the simplest GUI that will provide a visual image of the data. Type detection is based on the type of the first file, and all files in the argument list will need to be of the same type.

-t Text. Open the interactive text-based (command line) user interface

-p Permissive. Set the environment variable MIEN_NO_VERIFY_XML, which prevents the MIEN xml dialect parsers form raising exceptions if they are asked to read xml that doesn't obey the document type definition.

-a Application. Open a component application. Legitimate values include:
"**cell**" - the display GUI for anatomical data (CellViewer)
"**data**" - the display GUI for time series data (DataViewer)
"**image**"- the Display GUI for image data (ImageViewer)
"**wave**" - the stimulus and waveform synthesizer (Waveform)
"**dsp**" - the Dsp toolchain generator (Dsp)

Note: setting Application to "cell2d" will force execution of the (deprecated) 2D CellViewer. This viewer uses only the 2D library, rather than the OpenGL library. It is slower, less pretty, and less capable than the GL version, and is no longer supported, so you usually don't want to run it unless your system doesn't support PyOpenGL. In this case, using "-a cell" will fall back to the 2D viewer after the 3D viewer fails, so the only reason to use "cell2d" is for debugging the 2D application on a PyOpenGL capable system.

-r Run (method). Find and run a method of one of the objects in the first specified data file (this file must exist, and it must contain an nmpml model structure). There are several syntaxes for "method."

The most complete syntax is an nmpml path, joined by a "." to the name of a method (e.g. "/NmpmlDocument:MyDoc/Experiment:MyExp1.run)
If the path doesn't begin with a "/" but does contain a ":" the top-level document element is automatically matched, and the search begins below it. This syntax exactly specifies a particular element and method, which must exist or the command fails immediately.

Alternatively, the path may be only an nmpml tag, joined to a method by a '.' (eg "Experiment.run"). In this case MIEN will find the first instance of that tag and call the method.

The method name may be omitted, in which case it is assumed to be "run", so "-r Experiment" is equivalent to the other examples above, if called on a document that only defines one Experiment tag.

-s Scan (depth). Print a scan of the file to the indicated depth and exit.

-b Batch (fname). Call AbstractModels that take variable file inputs. fname is the URL of a file containing input data. MIEN will call the first AbstractModel in the document, passing it data from the file at fname. The output will be stored in a file of the same type and extension as fname with _mien_batch added to the name.

-c Convert (type). Convert the input files to the specified type (which should be a file name extension). No interface will open. Note that if '-c' is specified on time series data files, the file subset selection dialog (command line version) will still appear, which can be useful for cropping files. To over-ride this, use "-cf".

-f Format. Force MIEN to use the specified format. This can be important for xml files that are not in the nmpml dialect. By default, the specialized nmpml classes are used to represent corresponding nmpml tags. This offers advanced features (for editing, display, and simulation of models), but may cause errors when loading non-nmpml xml.

If you know the input file is xml, but not nmpml, use this switch with format "xml" if you want to be sure that the file is un-altered, or use format "tonmp" to force the document into nmpml (which will enable all the capabilities of the GUIs, but may cause the document to be altered if you save it (even if you don't make any modifications by hand).

Legal arguments for "format" are any key in the mien.fileIO.filetypes dictionary. You can get a list with "-f list", and you can also get a list of the extensions associated to each file type with "-f listext".

-i Interact. Prompt for selection of sub-components while loading files. This allows you to import branches of an xml tree, part of a binary file, etc. This can greatly speed loading times, and allow you to scan huge files, but requires extra user interaction.

--version Version information for MIEN and the python modules it uses.

File Structure

Mien stores two kinds of information: big chunks of binary data, like physiology data, images, sound, etc., and descriptive data, like anatomy, model descriptions, experimental protocols, and Dsp tool chain instructions. Pure descriptive data are natively stored in a dialect of xml called nmpml. Pure numerical data are natively stored in a "header plus packed binary" format called "mdat". Files with extension ".mien" are in the native Mien Hybrid format, meaning that they contain both types of data in the same document.

The "mien" command line command provides the ability to convert between formats, using the "-c" switch. You can convert a .mien file to mdat to get only the data part, or to nmpml to get only the (uncompressed) description part.

Using MIEN Extensions

MIEN is set up to load extension blocks from outside the MIEN project. Here's how the system works:

MIEN components find user add-ons using the environment variable MIEN_EXTENSION_DIR. You can set this to any value you want, but if you don't set it, mien/frontends/mien will attempt to set it for you. It looks for a directory named $HOME/mienblocks. If MIEN_EXTENSION_DIR isn't set manually, and it finds this directory, it sets that value in MIEN_EXTENSION_DIR.

All immediate subdirectories of MIEN_EXTENSION_DIR are added to the Python path when mien/frontends/mien runs. Note that these directories are added to the end of the path, so MIEN can't overwrite modules defined somewhere else. It's up to the user to choose unique module names.

Dsp looks for a directory MIEN_EXTENSION_DIR/dsp, and checks it for MIEN block modules. Files with the .py extension in this directory will be treated just like the files in mien/blocks. If they define any functions, these will automatically appear in the Dsp block selection GUI, and the DataViewer's "Dsp" menu.

In the DataViewer menu, these modules will be indicated with names that look like "user - myModuleName". For example, let's say you define an extension named "signal". This is legal, since the full name of your module is "signal", and the MIEN built-in module with the same name is properly named "mien.blocks.signal". In the Dsp block selection GUI you will see just that - one module named "signal" (yours) and one named "mien.blocks.signal". In the DataViewer menu, however, you will see one module named "signal" (the built-in one) and one named "user - signal" (yours).

DataViewer looks for a directory MIEN_EXTENSION_DIR/gui to find extensions

that define their own GUI. If there is such a directory, it looks through any modules in there to see if they define a class named "DVControlPanel". If there is a class with that name, a function will be added to DataViewer's "Extensions" menu named "Show myModuleName Extension GUI". This function will call the class constructor myModuleName.DVControlPanel and pass it one argument, which is a pointer to the DataViewer instance (so your extension class should expect this argument and do the right thing with it).

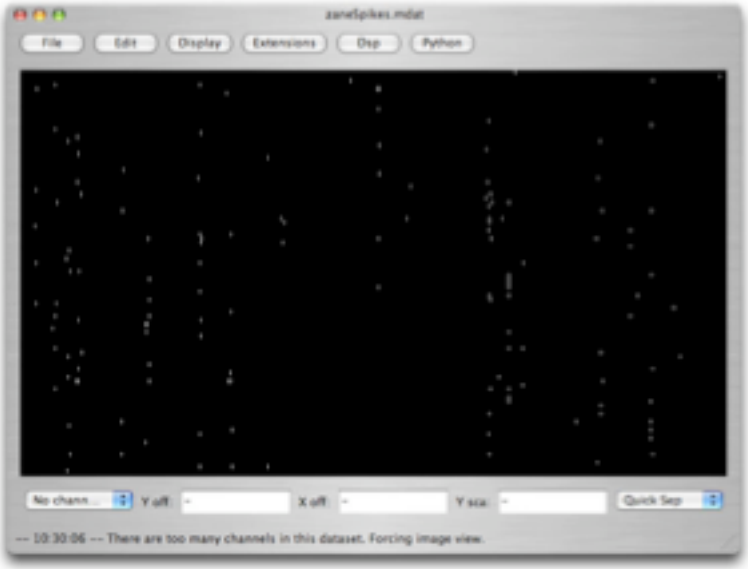
Step-by-Step Examples

DataViewer: Calculating spike clusters

1. Load a time series data file containing transposed spikes.

> mien –a data [filename]

You should see the viewer, showing a black field with some white flecks. You will see a console message that there are too many channels to display, forcing image view.

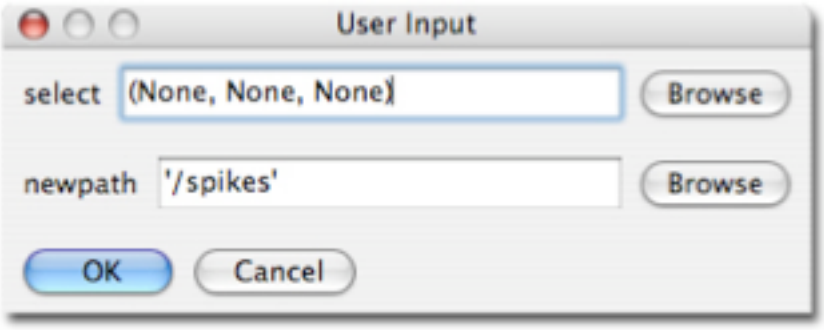


2. Threshold.

From the DataViewer menu options:

Select Dsp -> gicspikesort.spikes -> tsToEvents

select: (None, None, None) [default]
newpath:'/spikes' [default]
Click "OK."



You will see a console message like: "Completed gicspikesort.spikes.tsToEvents in 0.4766 sec" You won't see any change in your image view.

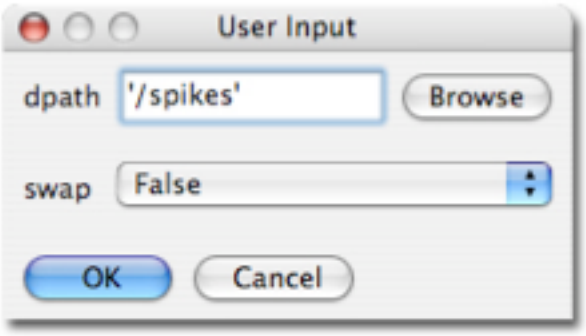
If you open the data editor (Edit -> Data Editor) you should find that you now have a hierarchal data tree with a sub-data element named "spikes", but you don't need to do this.

3. Get rid of the timeseries data. Use either one of these two methods:

Method 1. Using subdata operations.

Select Dsp -> subdata -> moveToTop

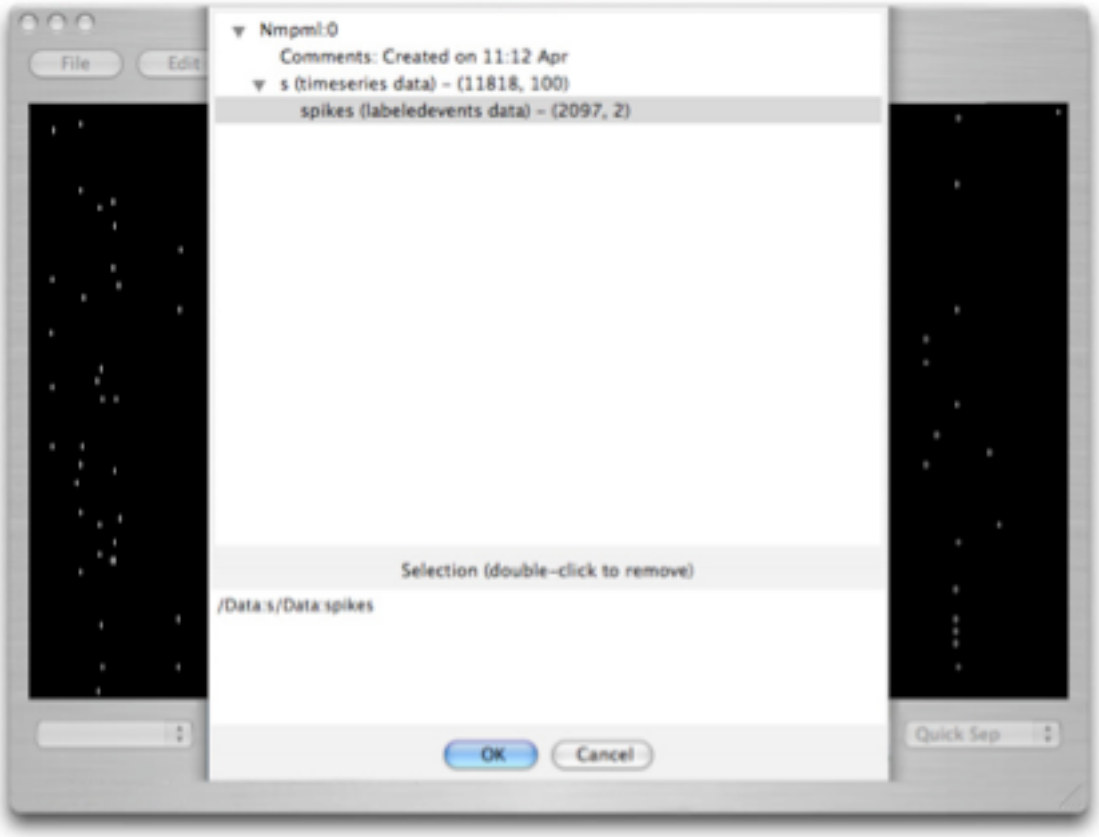
dpath:'/spikes'
swap:False [default]



Method 2. Saving a new file.

Select File -> Save Subset

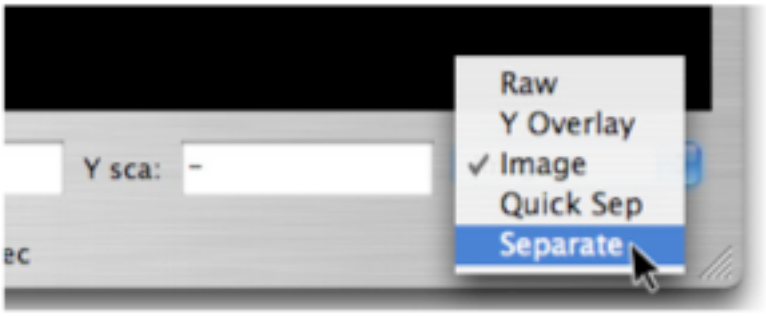
Use the selection browser to select /Data:s/Data:spikes only. If you select anything else, make sure you double click on it to remove it from the selection list. Click OK.



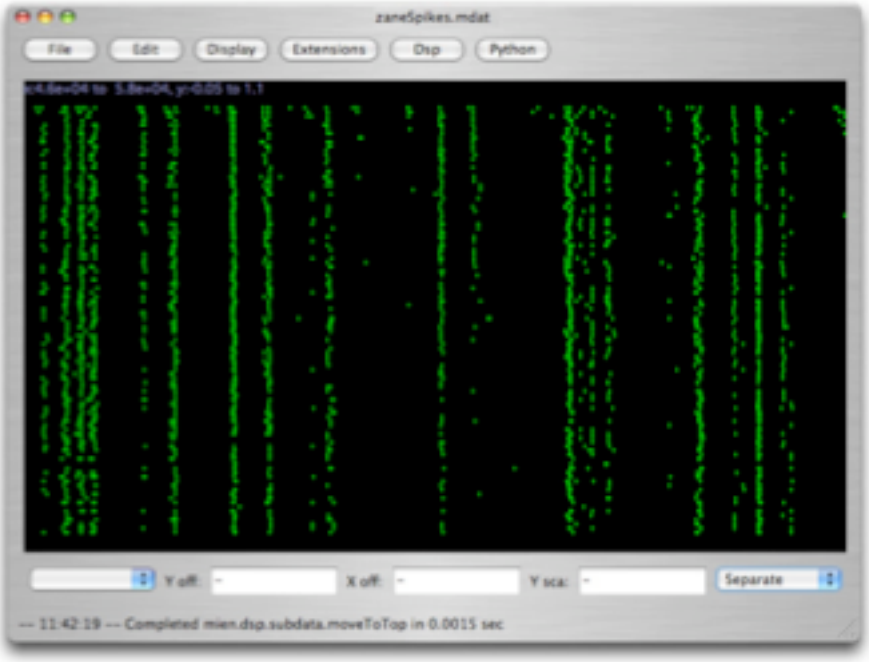
Now you will see a file browser. Create a file name. End the name with ".mdat" to save in mien native format.

Now select File -> Load and open the file you just saved.

In either case (Method 1 or 2), you will now have a strange grayscale image. Remember you were forced into image view. Use the pull down menu in the lower right to re-select "Quick Sep" or to select "Separate".



Now you will see a lot of green spikes. You can press "q" to see the full range of the data. If you save a subset, exit DataViewer, and restart it with the new file, you will automatically leave image mode.



NOTE: Method 2 is slower, but a bit safer and more memory efficient. In the future, Method 1 should be the "right" way to do this, but if you have trouble with it, switch to Method 2.

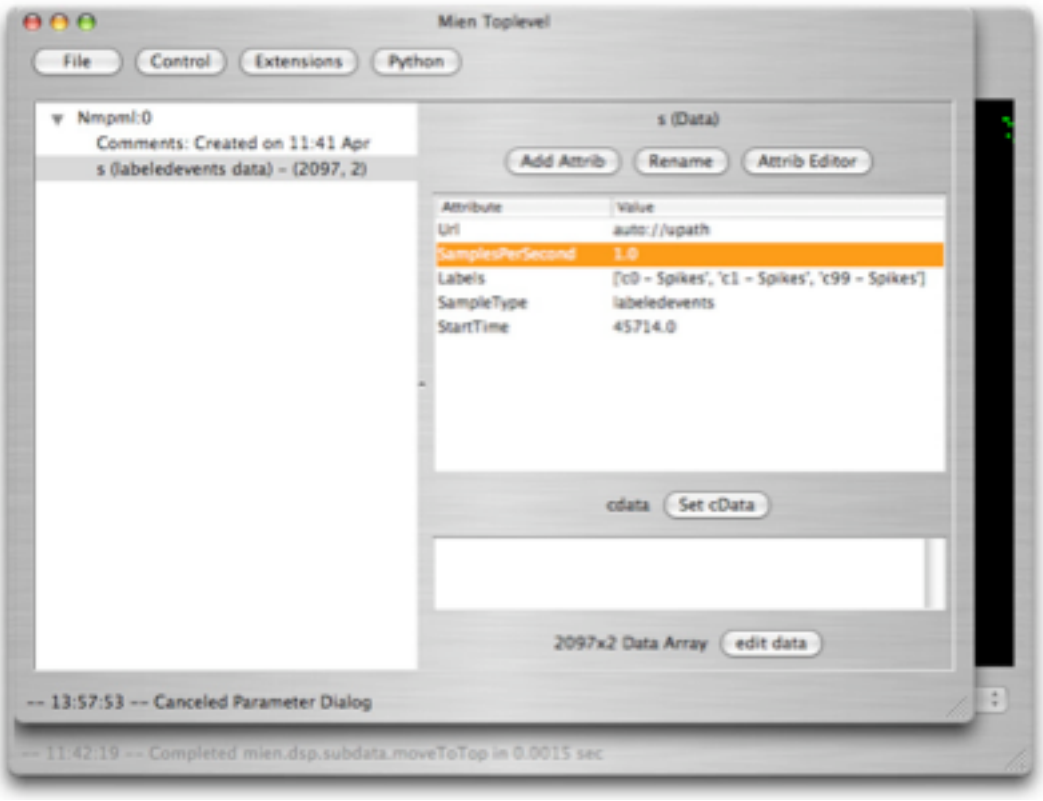
4. Change the sampling rate.

   Since MATLAB didn't store a SamplesPerSecond attribute for these spikes, DataViewer thinks the sampling rate is 1.0 Hz. This means you can't use parameters in seconds or milliseconds until you fix it.

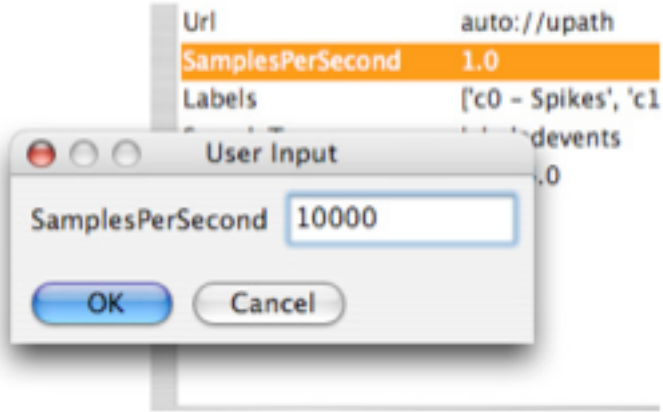
   Select Edit -> Data Editor

   You should see the mien top-level xml editor with a data element named "spikes". Double-click on that.

   In the right hand pane, you should now see a list of attributes including "SamplesPerSecond" : 1.0



Double-click on the line specifying that attribute. You should get an edit box. Type in your real sampling rate (10000) and click OK. Close the Data Editor.

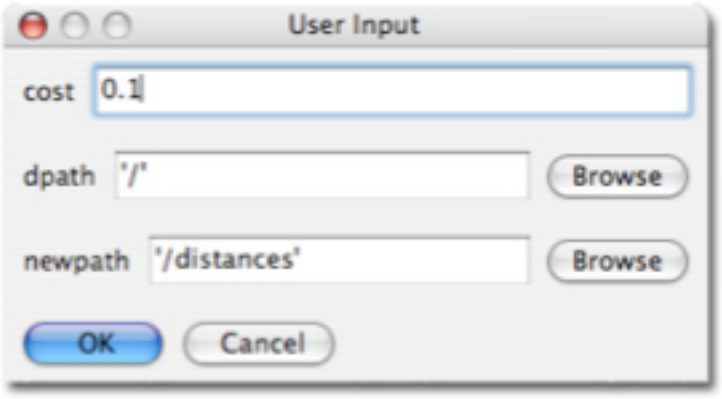


The spikes in viewer have disappeared! That's OK. They are all shifted to the left in time, since the sampling rate increased. Press "q" to re-center the view. The spikes should be back now.

5. Calculate a matrix of distances between the spike trains.

Select Dsp -> gicmext.comparators -> multiDistance

cost: 0.1 (or something you like. 0.1 means the max range of movement is 20ms, and the max probable range is 10)
dpath:'/' [default]
newpath:'/distances' [default]



This will take 10 to 20 seconds. Then you should see a console message like "Completed gicmext.comparators.multiDistance in 13.9163 sec". If you see a ton of messages like "Inf cost" or "0 cost", check your sampling rate.

Once again, you can now find a sub-element "distances" under "spikes" in the data editor if you want to check (If you still have the Data Editor open from step 5, you may need to right click on "spikes" and select "refresh tree" to get an updated display).

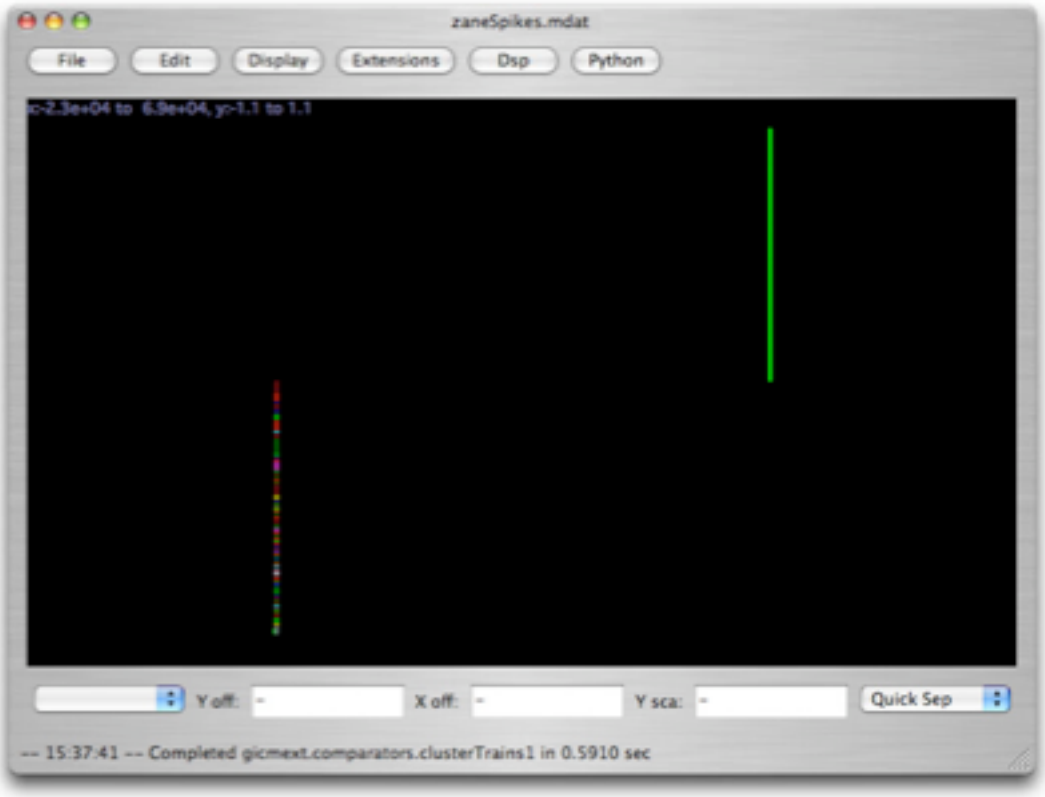
6. Calculate spike clusters.

Select Dsp -> gicmext.comparators -> clusterTrains1

dpath:'/distances' [default]
newpath:'/cluster' [default]
mode:'cluster' [default]
Click "Ok"

This will take a few seconds, and print a lot of output in the terminal window that looks like (15, 76) 11 23 (this means "mapping spike train 11 to spike train 23"). It will eventually produce a message like "Completed gicmext.comparators.clusterTrains1 in 0.5689 sec". Although the cluster algorithm runs in <1 sec, it will take at least 4 sec to refresh the display, because the current algorithm that draws clustered spike trains is slow.

Press "q" to display all plots, and then "a" to zoom out.

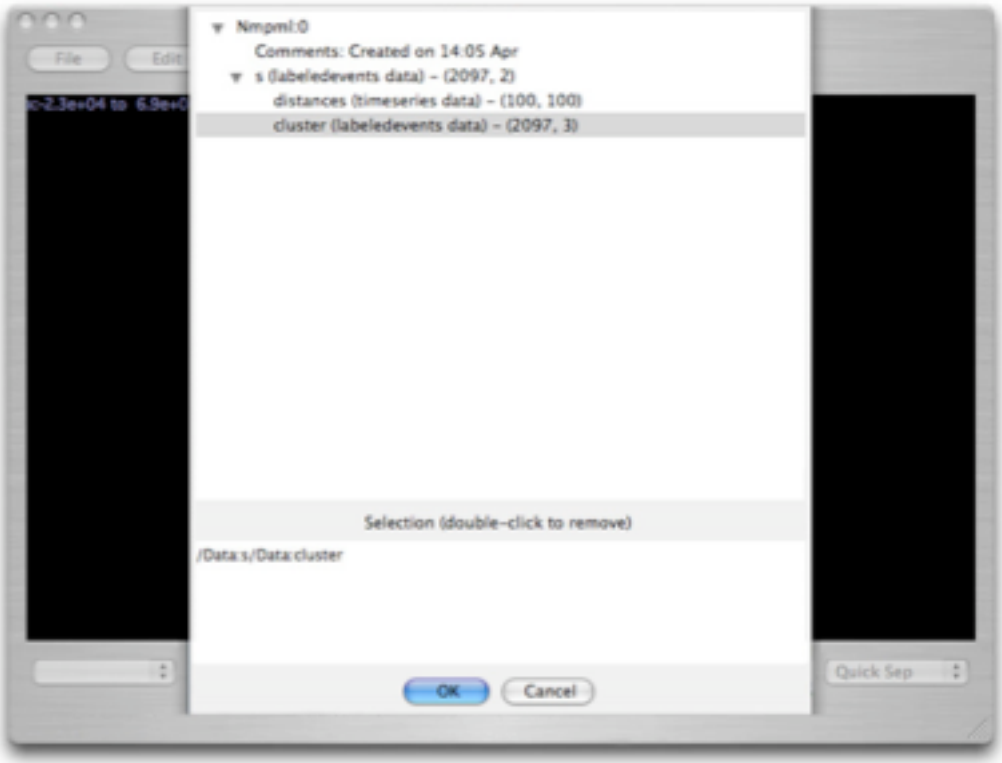


You will now see two dotted lines, one green, on top, and one multicolored on the bottom. This is pretty much a bug in the display algorithm for datasets that contain more than one labeled event raster. It will be fixed, but you needn't worry about it, since you don't need to see both rasters. After step 7 things will look prettier.

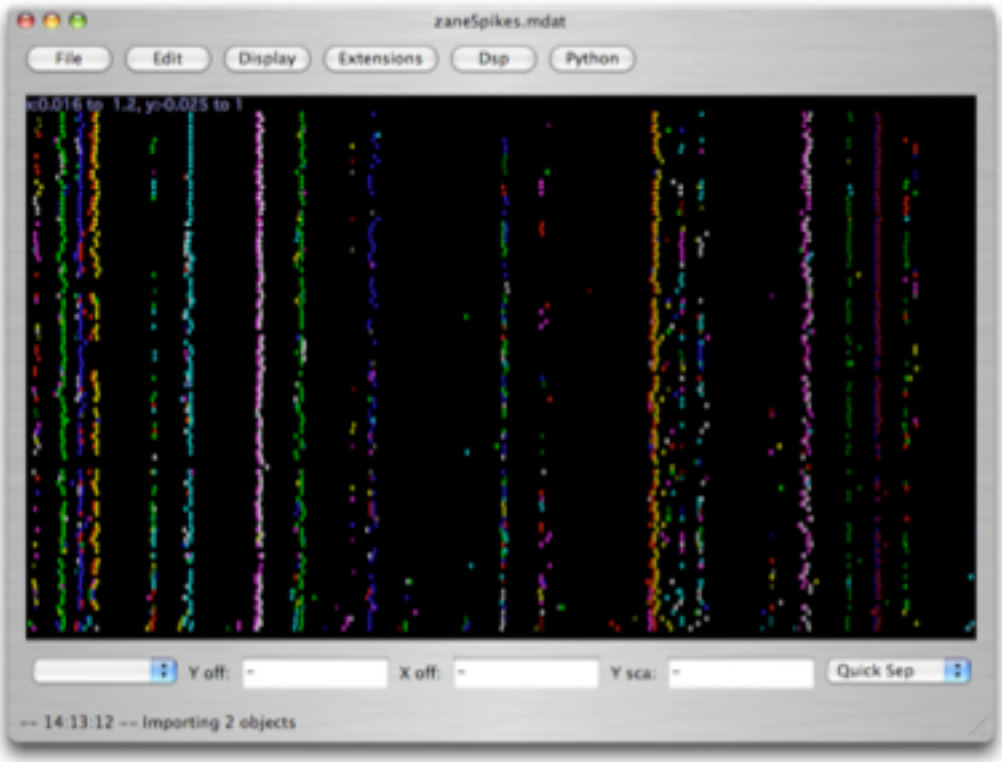
7. Get rid of the old un-clustered spike trains.

Just like in Step 3, you should be able to do this with Dsp -> subdata -> moveToTop, or with File -> Save Subset. There seems to be a bug in parsers.matfile that makes writing to .mat in Step 8 fail only some of the time, unless you do Save Subset. For now, to be on the safe side, do:

File -> Save Subset
    Select the path "/Data:s/Data:cluster"
    Save as a ".mdat"
File -> Load
(select that file)



Again, expect some delay to draw the clustered spike train.
Now you should see a bunch of multicolored spikes.



By default, these spikes are colored "hotter" if they have an event index that was assigned earlier.

8. Save the data.

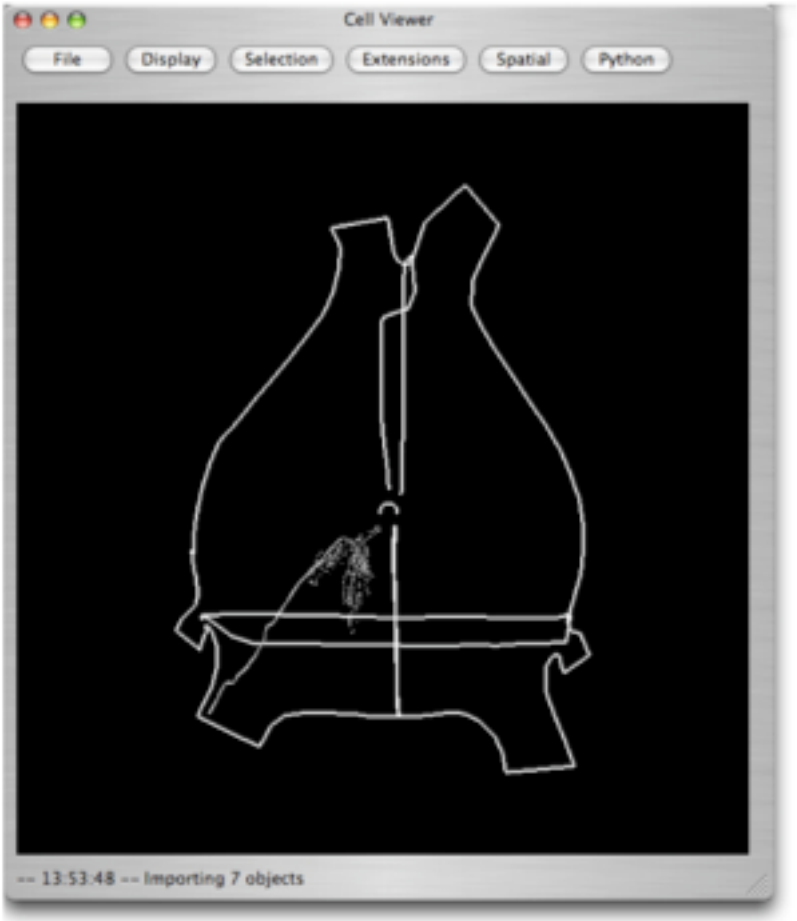
Remember that anything DataViewer can do, the Dsp toolchain editor can assemble into an xml instruction file and save to disk. MIEN can then execute those instructions in batch mode.

CellViewer: Locate points on an axon

1. Load morphological data file.

    Locate the data file you want to view, either by clicking File -> Load and navigating to it, or by dragging and dropping it into the middle of the CellViewer window.

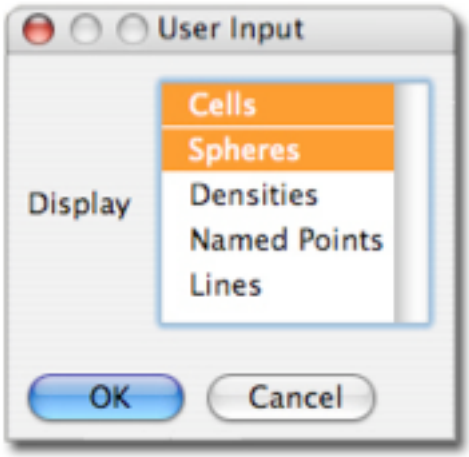
    An afferent data file has been loaded in the example shown. CellViewer displays the outline of the terminal ganglion, the cells and the varicosities. Each of these components exists as a different filter.



2. Choose to look only at the cell and varicosities.

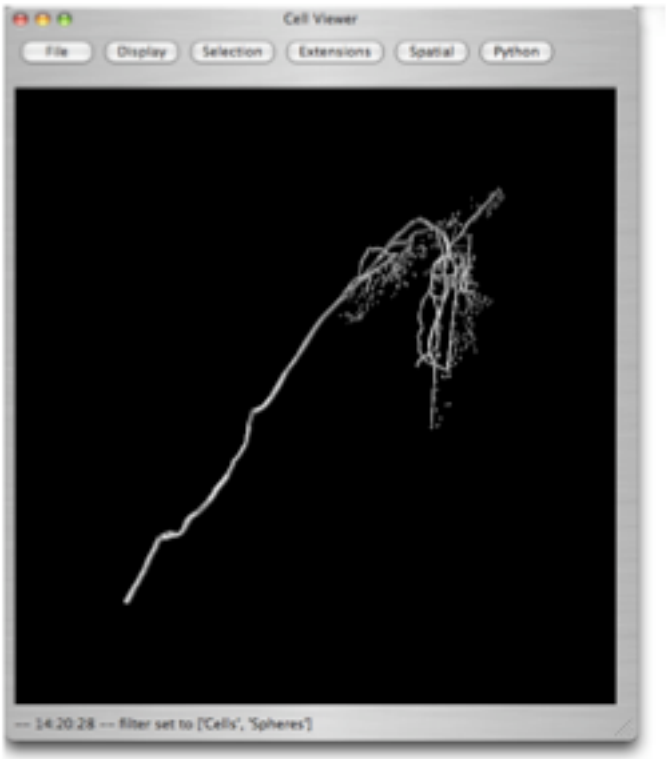
Select Display -> Filter

In the dialog box that appears, select "Cells" and "Spheres." You can select multiple items by holding down the command key (Ctrl on a PC) while clicking.



In order to return to the original view, showing all the filters, you need to go back to Display -> Filter and select all of the filters.

The display now shows only the axon and varicosities.



3. Adjust image.

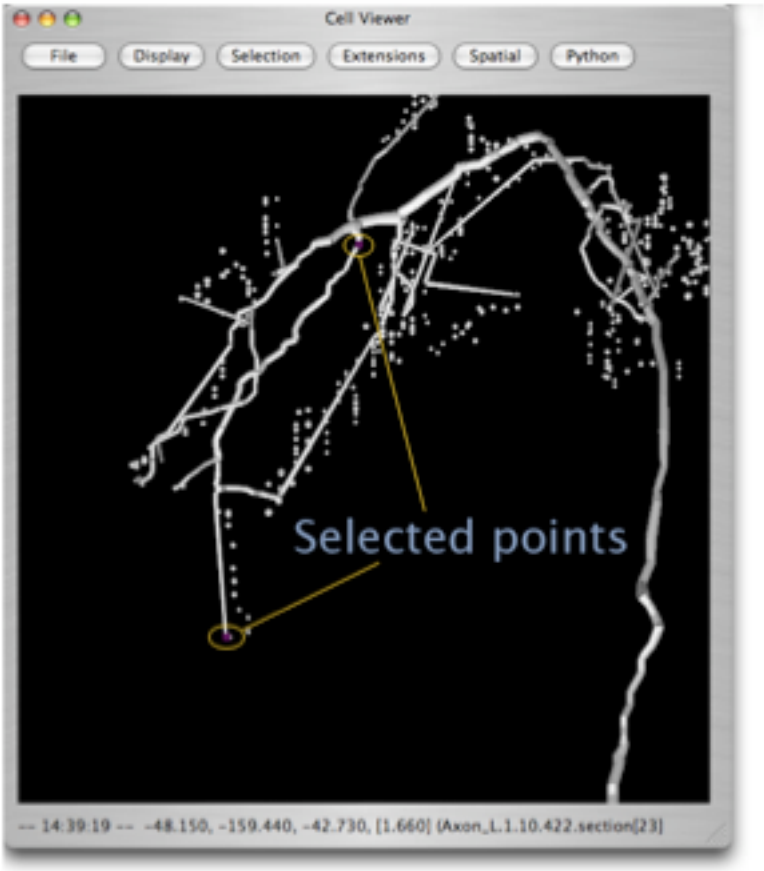
Press '=' to zoom in and '-' to zoom out.
Orbit around the image by pressing 'a' 'w' 's' and 'd'.

Note: a more comprehensive list of key commands and definitions is given below (see pages 21-24).

Adjust the image until you have a clear view of the axon section you want to work on.

4. Select two points to locate.

In order to select point on the axon itself, **hold down the shift key** while clicking. Otherwise you will be clicking on points in a flat plane, as if on the surface of the cube containing the axon.



After selecting two points on the axon, look at the terminal window to find their coordinates.

You should see two sets of numbers that look like this:

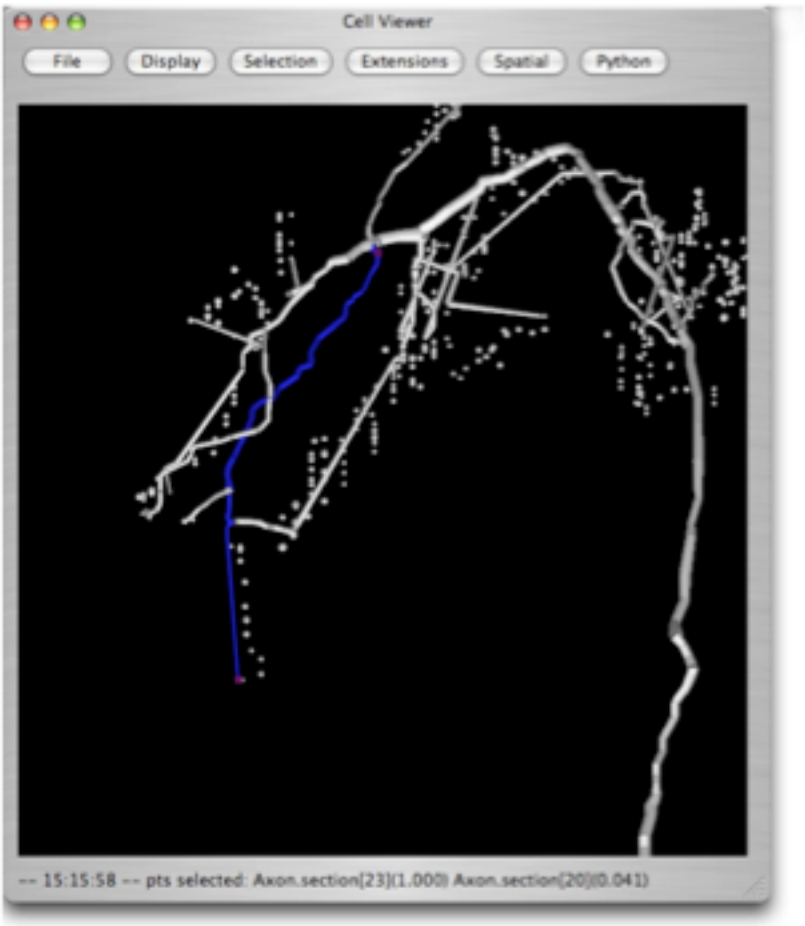
-40.500, -67.120, -11.780, [2.760] (Axon_L.1.10.422.section[20](0.072))
-48.150, -159.440, -42.730, [1.660] (Axon_L.1.10.422.section[23](1.00))

The first three numbers are the point's coordinates in xyz space. The next number, in brackets, is the diameter of the axon at that point (in microns). The last piece of data gives the name and location of the section containing the point you clicked on.

If desired, for viewing or image-creating purposes, you can select the path connecting the points you just clicked on:

Select Selection -> Select Path

The path will turn blue, as shown.



Commands List

DataViewer

click	Add X (vertical) Marker
shift-click	Add Y (horizontal) Marker
middle click	Center view
right click	Context Menu

---
a xZoomOut: Double the range of x coordinates in the viewable area.

A yZoomOut: Double the range of y coordinates in the viewable area.
---
k Remove all markers from both axes
---
q Set the limits of the viewable area to display all plots
---
r Redraw the screen
---
x Delete the most recently placed x marker

X Delete the most recently placed y marker
---
z Zoom the viewable region to the region between the last two markers. Operates in both x and y, but will not zoom an axis if there are fewer than two markers present on that axis.

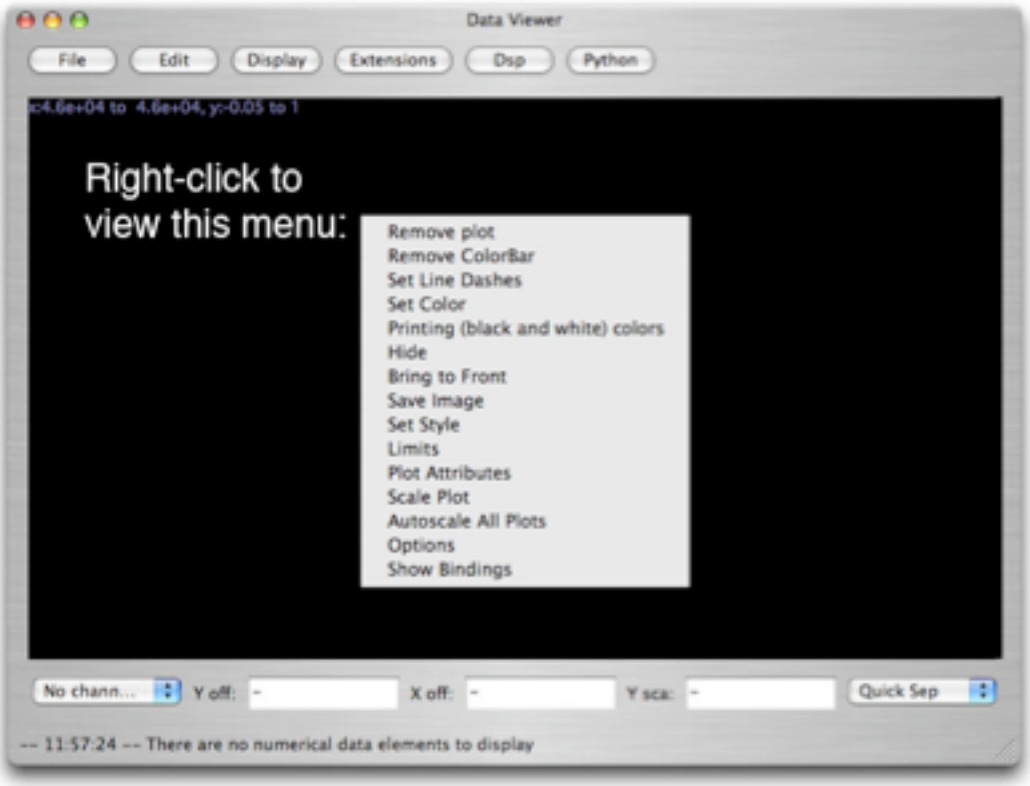
⇧ Pan viewable area up

⇨ Pan viewable area right

⇩ Pan viewable area down

⇦ Pan viewable area left

These commands can be displayed by right-clicking on the DataViewer window and selecting "Show Bindings" from the pop-up menu. (see image next page) They will then be shown in the terminal window.



CellViewer

click	select point on cube surface
shift-click	select point on model
right click	re-center image around point clicked

F1	Display context menu
F2	Add a view preset
1	Standard view ('z')
2	Standard view ('x')
3	Standard view ('y')
4	Redraw the graph
5	Select a view preset to use

-	Zoom out (increase the cross-section of the bounding box)
=	Zoom in
_	Reduce depth of field
+	Increase depth of field
w	Orbit front edge of image up
o, s	Orbit down
a	Orbit left
e, d	Orbit right
⇧	Rotate camera (not image) up
⇨	Rotate camera right
⇩	Rotate camera down
⇦	Rotate camera left
p, r	Move forward on z axis
u, f	Move backward on z axis
', q	Roll image counterclockwise
.	Roll image clockwise
<, W	Pan camera up
O, S	Pan camera down
A	Pan camera left
E, D	Pan camera right

(explanation continues on next page)

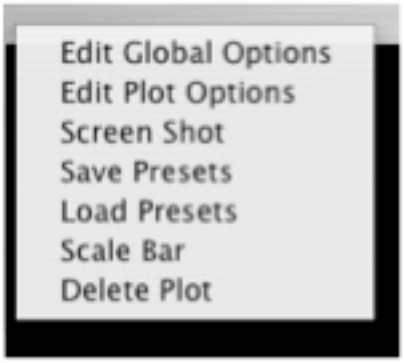
Standard view produces a view that is automatically sized (and set to large depth of field) to show the whole image, and has the camera looking along the indicated axis. The default view is Standard view ('z'). This shows a typical X=Horizontal (positive to the right), Y=Vertical (positive up) cross-section.

Orbiting is a complex camera movement in which the camera traverses a circular orbit while continuing to point at the same point in the center of the viewing volume. The effect is to cause the image to appear to rotate around its center. Because orbiting is an "image centric" action, the controls are reversed. Pressing 'a' (orbit left) will cause the front edge of the image to appear to rotate to the left. This means the camera is actually orbiting to the right.

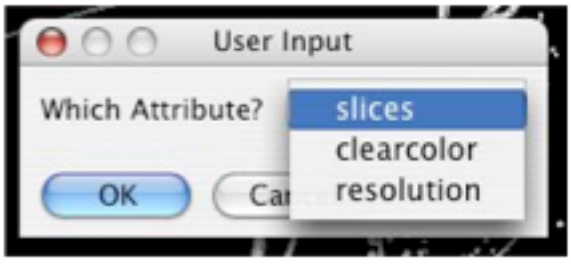
XY panning is a typical camera control, so pressing 'A' (pan left) moves the camera to the left, without changing its direction of view, and causes the image to appear to move right.

Note: you almost never need horizontal panning, since you can use right mouse click to see any visible point as the center of the view in the display pane.

F1 retrieves the following menu:



Under "**Edit Global Options**" you will find:



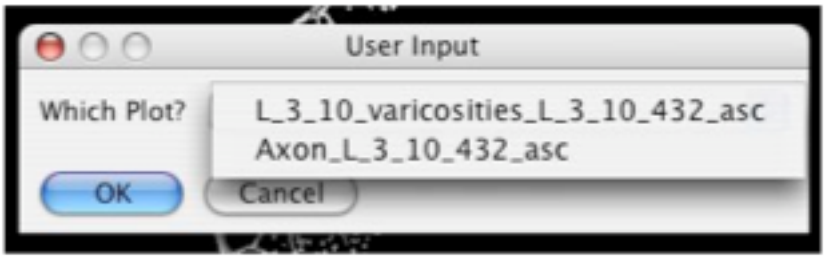
Slices refers to the degree of precision with which the image is drawn. It is

set to a default value of 6, but for the best quality image it should be increased. Increasing the number of slices will slow the program down, so for most purposes the default value is fine.

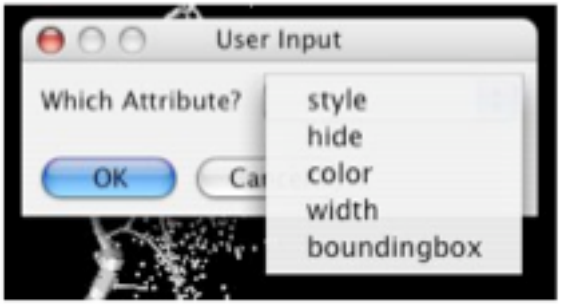
Clearcolor refers to the color of the background. It is set to black by default, and the color is defined by numerical values for red, green, and blue. Black is (0, 0, 0) and white is (1, 1, 1).

Resolution refers to the amount, in percent, that you turn when you press a, s, d, or w. By default it is set to 10%, but you may want to change it to 5% if you're going to be working closely zoomed in on an image.

Back to the **F1** menu. Under "**Edit Plot Options**," if you have more than one plot open, you will find a list of the plots, and you will need to select the one you want to edit:



After selecting a plot you will be presented with the following options:



Hide is a Boolean value, and by default it is "False," so if you want to hide a plot you need to type "True." It is case sensitive. To get the plot back, you'll need to go back in and type "False."

Color for plots is set to white by default. The color is defined by numerical values for red, green, and blue. White is (1, 1, 1). Increasing the red value to 2, for example, (2, 1, 1) changes the color to light pink.

Screen Shot saves a bitmap file (called foo.bmp) to your home directory.

Known Issues

Some of the GUIs appear sized wrong. Notably, the sash window in the xml editor always seems to appear all the way to the left, so you can't see the tree browser. You can drag the sash to reposition it, and then use the editor normally, so this is more an annoyance than a failure of function. It seems to be a wxGTK issue, not an ia64 issue, so it will occur on any Linux machine that runs MIEN. It will get fixed eventually but, since it's platform specific, non-fatal, and non-trivial to debug it may take a while.

Links

http://sourceforge.net/projects/mien
http://cns.montana.edu/~gic/programming.html

License

GNU General Public License (GPL)