SBFSEM-tools

Sara Patterson

Neitz Lab, University of Washington

December 17, 2017

SBFSEM-tools

Goal: basic analysis support requiring minimal setup and background in computers, math, etc.

Current capabilities:

- Import and parse data into Matlab through Viking's OData service
- Parse data into high-level data structures (Neuron, etc)
- Count up synapses and condense synapses spanning multiple slices
- Summarize synapses with statistics and plots
- Arbitrary geometry rendering
- Basic network analysis

Install

First download or clone SBFSEM-tools.
Make sure SBSFEM-tools is added to your MATLAB path like so:

```
addpath(genpath('C:\...\sbfsem-tools'));
```

If you already have JSONLab installed, make sure

```
which loadjson
```

returns the version in sbfsem-tools. Otherwise, you might get some errors.

Neuron

The Neuron class is the basic representation of a cell exported from Viking. To create a Neuron:

```
% cellName = Neuron(cellID, 'source');
c207 = Neuron(207, 'temporal');
```

Note: Source can be 'inferior', 'temporal' or 'rc1'. Also, abbreviating to 'i', 't' and 'r' works as well.
Open the Neuron class in the user interface

```
NeuronApp(c207);
```

To update a neuron in the workspace:

```
c207.update();
```

Neuron

Properties

Neuron has the following publicly accessible properties:

- data
- `viking` - Structure of the neuron's info from viking
- `nodes` - Table of all annotations
- `edges` - Table of links between annotations
- `volumeScale` - Units are nm/pix for X,Y and nm/section for Z
- `synapses` - Table of each child structure.
- `geometries` - Table of closed curve geometries
- `analysis` - Containers.map of analyses
- `lastModified` - Last update of neuron from OData

NeuronGroup

Create a single data object to hold related Neurons. Note inputs can be ID numbers or existing Neurons.

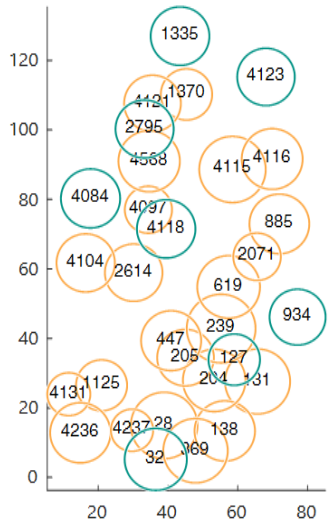
```
h1hc = sbfsem.NeuronGroup([28, 447, 619]);
% Add a neuron to an existing group
h1hc.add(4568);
% Remove a neuron
h1hc.remove(4568);
```

NeuronGroup

Methods

To plot the somas of all Neurons in the NeuronGroup:

```
h1hc.somaPlot();
h1hc.somaPlot('addLabel',true); % Label with ID
h1hc.somaPlot('ax',gca); % Add to existing axis
% Two methods for controlling plot color:
h1hc.somaPlot('Color', [0 0.8 0.3]);
h1hc.setPlotColor([0 0.8 0.3]); h1hc.somaPlot;
```



Mosaic of H1 and H2 somas

NeuronAnalysis Class

The NeuronAnalysis class helps keep population data organized by managing input parameters and results of common analyses. To create a new analysis, subclass NeuronAnalysis and edit the `doAnalysis` and `visualize` methods.

See `Tutorial.m` for information on these existing classes:

- **DendriticFieldHull** - uses convex hull to estimate dendritic field area, includes methods for removing axons prior to analysis.
- **PrimaryDendriteDiameter** - returns the median dendrite diameter at a given distance from the soma.

Views

```
1  c6800 = sbfsem.Neuron(6800, 'i');
2  sbfsem.ui.StratificationView(c6800);
3  c207 = sbfsem.Neuron(207, 'i');
4  sbfsem.ui.SomaDistanceView(c207);
5  c4781 = sbfsem.Neuron(4781, 'i');
6  sbfsem.ui.NodeView(c4781);
7  sbfsem.ui.NodeView(c6800);
```

Convert to Graph

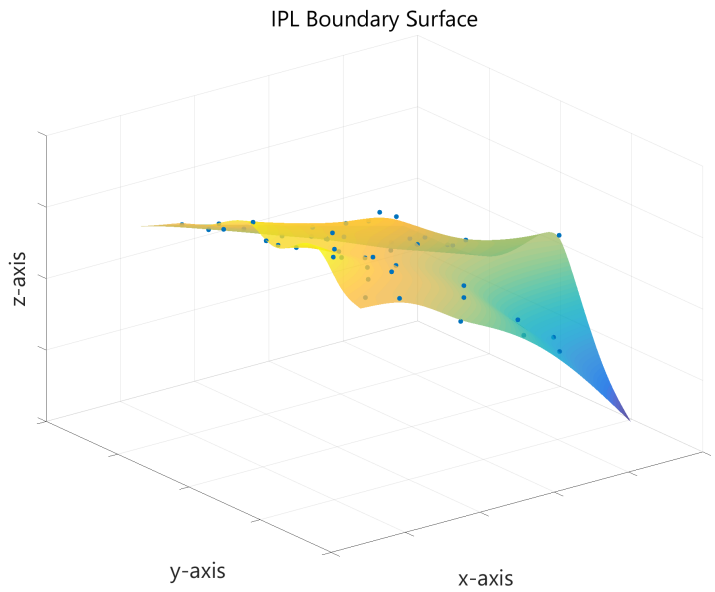
The dendritic structure can be converted to MATLAB's 'graph' and 'digraph' classes:

```
c127 = sbfsem.Neuron(127, 'i');
G = graph(c127);
plot(G, 'Layout', 'force');
% Default is undirected, add true for digraph
G = graph(c127, true);
```

See MATLAB's documentation for more information on how to use the graph class.

IPL Boundary Surfaces

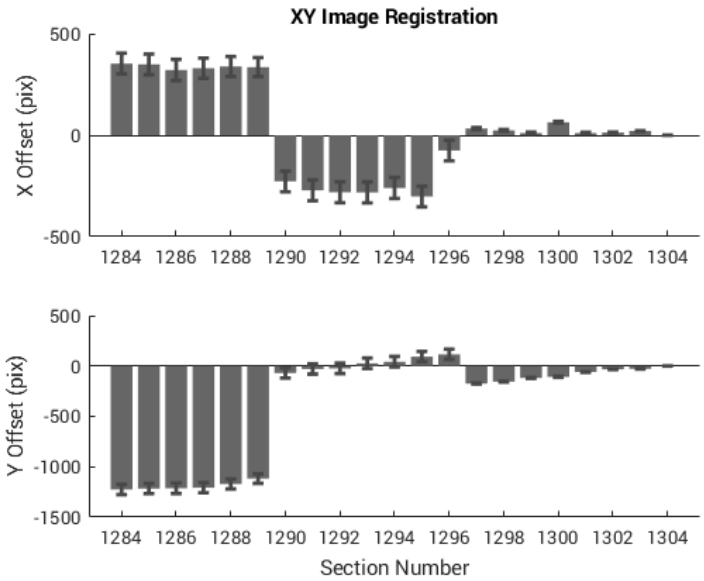
```
inl = sbfsem.core.INLBoundary('i');
% Update marker locations from OData
inl.refresh();
% Create the surface
inl.doAnalysis();
% Plot the surface
plot(inl);
```



XY Offset

The function **xyRegistration.m** calculates the XY offset through a range of Z sections and outputs statistics on the offsets relative to the most sclerad section input.

```
% S xyRegistration(source, [minZ maxZ], plotFlag);
S = xyRegistration('i', [1284 1309], true);
```



Next on the todo list is a systematic way of applying transforms

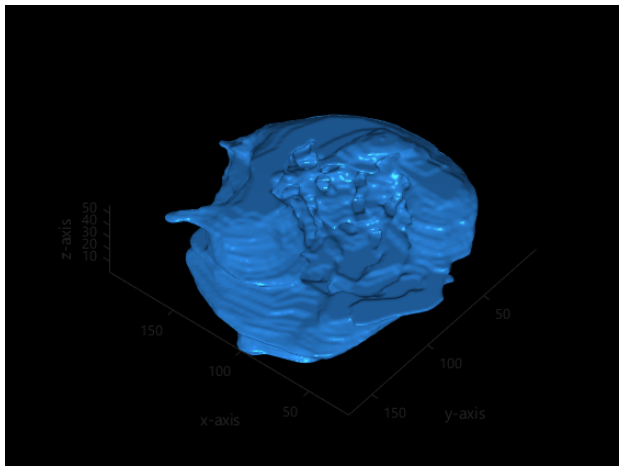
Renders

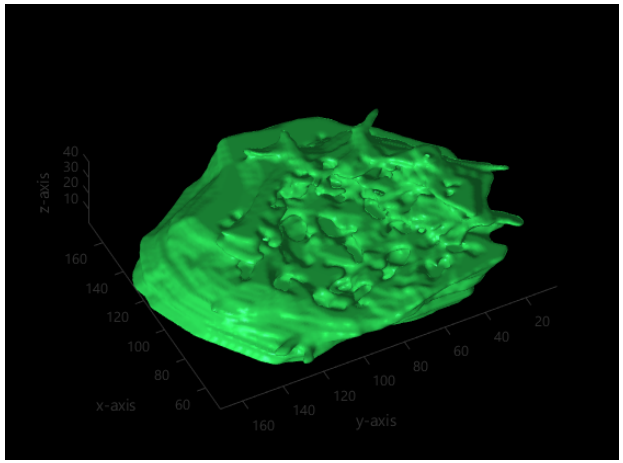
VikingPlot generates 3D models by fitting the data to three stereotyped geometries - sphere, cylinder and cone. I'm focusing my efforts on the opposite approach - rendering the structures 'as is'. By not fitting the data, the renders will be far more accurate. The tradeoff is that this accuracy applies both to the neuron's morphology and the small discrepancies in image registration.

Closed Curve Renders

Closed-curve structures can be rendered into volumes:

```
1  c2542 = sbfsem.Neuron(2542, 'i');
2  lmcone = sbfsem.render.ClosedCurve(c2542);
```





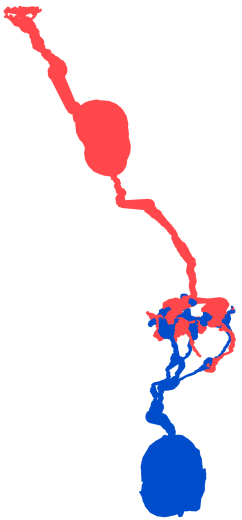
See the `Tutorial.m` file for more information on render colors, lighting and materials.

Disc Renders

Rotated Cylinders Method

There are two options for disc renders so far.

```
r1403 = sbfsem.render.Cylinder(c1403);
r1403.render();
r2578 = sbfsem.render.Cylinder(c2578);
r2578.render('ax', gca, 'FaceColor', [.2 .8 .2]);
```



To do:

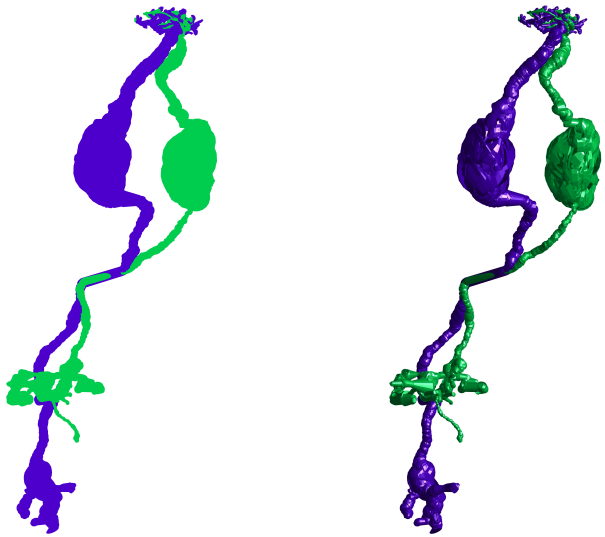
- Dendrites for Cylinder
- Smoothing for Disc
- Image registration

Rotated Cylinders

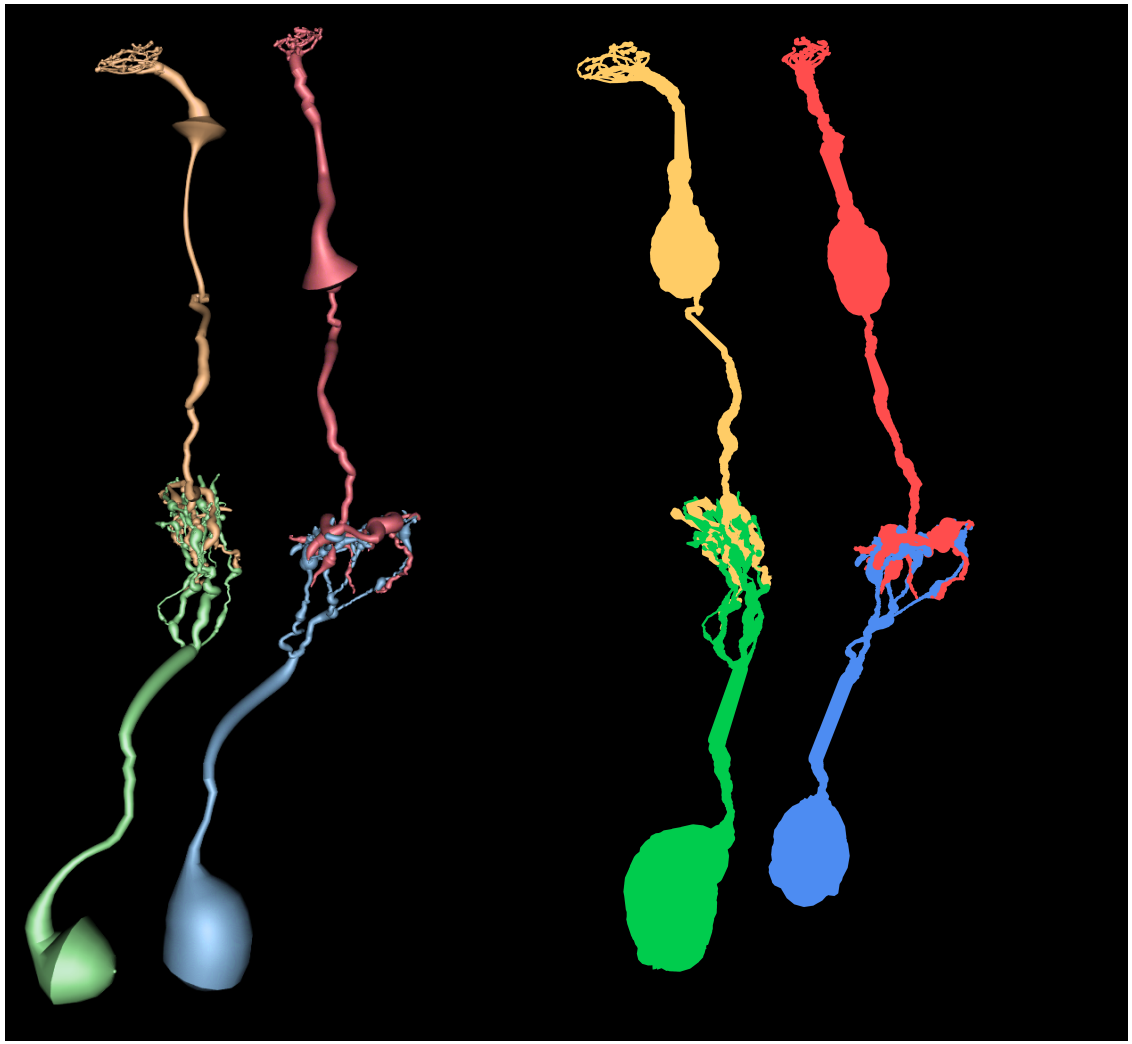
Future Work

I'm working on two approaches:

- Optimizing the existing algorithm to reduce the number of odd rotations.
- Converting the output to a binary mask volume and remodeling the surface.



VikingPlot Comparison



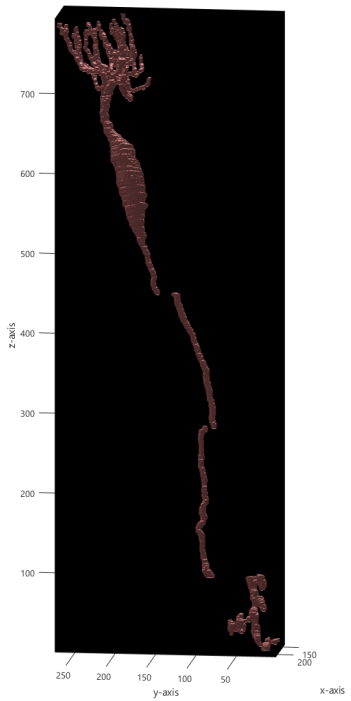
Efficient: This render took 32.77 sec on my laptop.

Disc Renders

Stacked Cylinders method

The Disc Render uses a similar method to the Closed Curve renders. While the result does have 3D lighting, it needs further improvement before being a viable alternative to VikingPlot or rotated cylinders.

```
c1893 = sbfsem.Neuron(1893, 'i');
rodBC = sbfsem.render.Disc(1893);
```

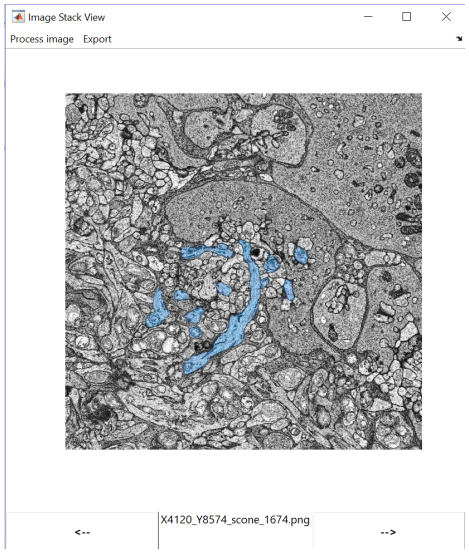


ImageStackApp

The ImageStackApp was built to scan through a series of images exported using Viking's Export Frames option (although this will work with any folder of images). The app will import all images in a given folder. If the filenames contain numbers (Export Frames appends a frame number automatically), these will be used to order the images. Use the buttons (or the left/right arrow keys) to browse through the images.

```
1 ImageStackApp('C:\...\foldername');
```

To crop the images, use `Process Image` $\rightarrow$ `Crop` amd draw a rectangle on the image. The change will not be applied until moving to another image.



Appendix

Here's some information and methods that are less essential:

1. Resources
2. Links
3. Tulip import method
4. Old version of NeuronApp

Links

The SBFSEM-tools repository can be found on Github.

The software used for annotations is Viking, developed by Jamie Anderson and the Marc Lab at University of Utah. Useful free programs involved in these analyses:

- Tulip supports graph visualization. The documentation for Tulip's Python package can be found here.
- Blender for 3D renders of neurons.

SBFSEM-tools was developed in the Neitz lab at University of Washington.

Resources

3D rendering

- Lorenson & Cline (1987) *Computer Graphics*, 21(4)
- 3D Math Primer For Graphics and Game Development by Fletcher Dunn and Ian Parberry
- Blender Cheatsheet

OData

- Microsoft Developer OData tutorials

Data Structures

- BaseCS
- Data structure visualization

Import

Step One: Tulip

Open a cell in Tulip and then open the Python command line. It's on the bottom toolbar.
Set the file name* and file path:

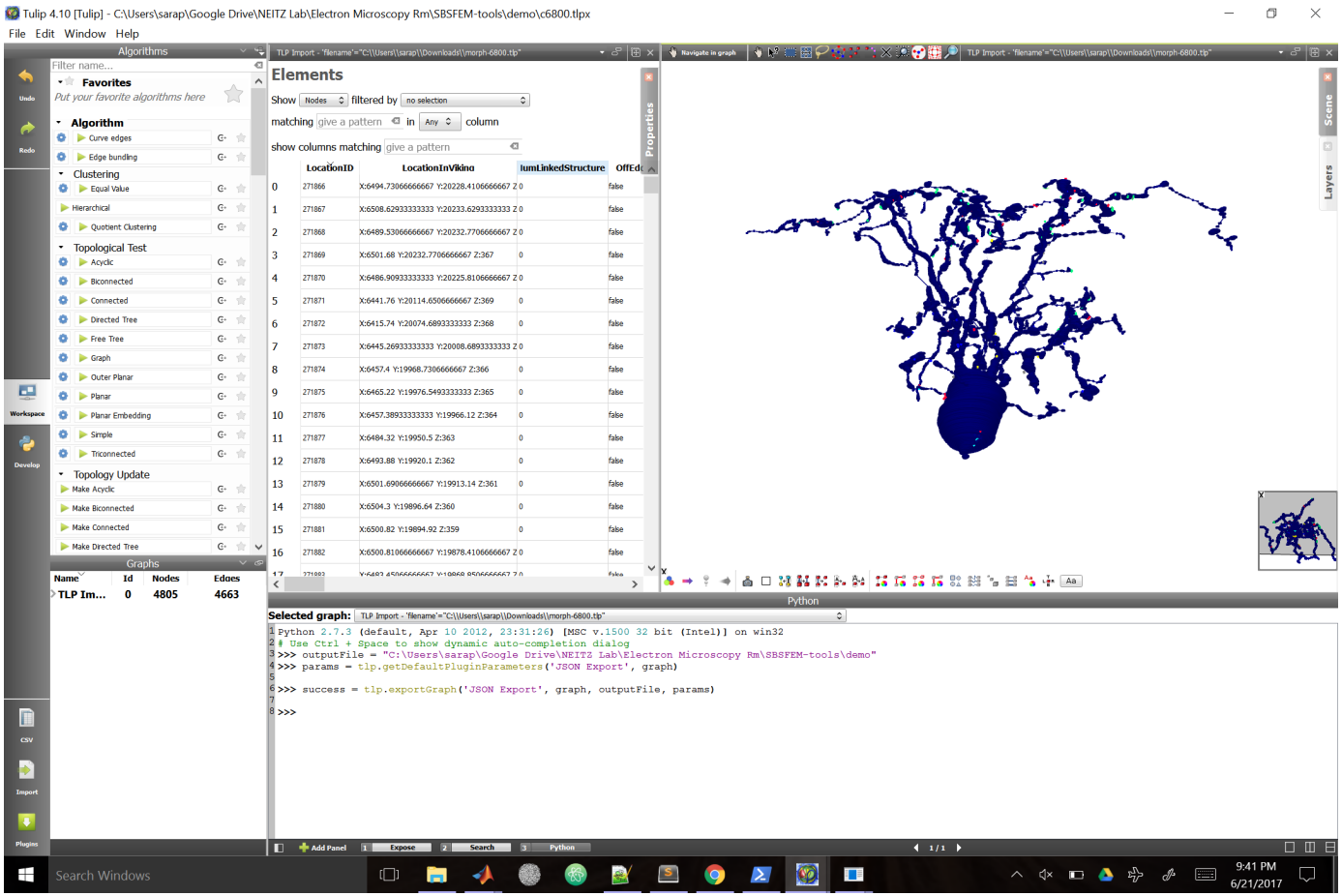
```
1    outputFile = "C:\...\c207.json"
```

Then run these two lines:

```
1    params = tlp.getDefaultPluginParameters('JSON
     Export', graph)
2    success = tlp.exportGraph('JSON Export', graph
     , outputFile, params)
```

Import

Step One: Tulip



Alternative Streamlined Import

Step One - Tulip

If you're comfortable with Python, you can skip opening Tulip's UI entirely.
First get the Tulip modules:

```
$ pip install tulip-python
```

This will allow you to export `.tlp` (or compressed `.tlp.gz`) files to JSON from a command line without Tulip's UI.

```
from tulip import tlp
graph = tlp.loadGraph("C:\...\morph-207.tlp")
outputFile = "C:\...\c207.json"
params = tlp.getDefaultPluginParameters('JSON Export', graph)
success = tlp.exportGraph('JSON Export', graph, outputFile, params)
```

Import

Step Two - MATLAB

Load in the JSON file and create a Neuron object:

```
c207 = Neuron('c207.json');
```

A dialog box will ask for the cell number and source (temporal, inferior, rc1). To avoid that, include them while creating the Neuron object:

```
% output = Neuron(filename, cellNumber, source);
c207 = Neuron('c207.json', 207, 'temporal');
```

To update the underlying data for an existing Neuron:

```
c207.updateData('c207.json');
```

Open up the user interface:

```
c207.openUI;
```

Cell Info Panel

This entire panel is designed with a future directory class in mind. As of now, you don't need to save each Neuron, so only set these if you have a specific reason for doing so.

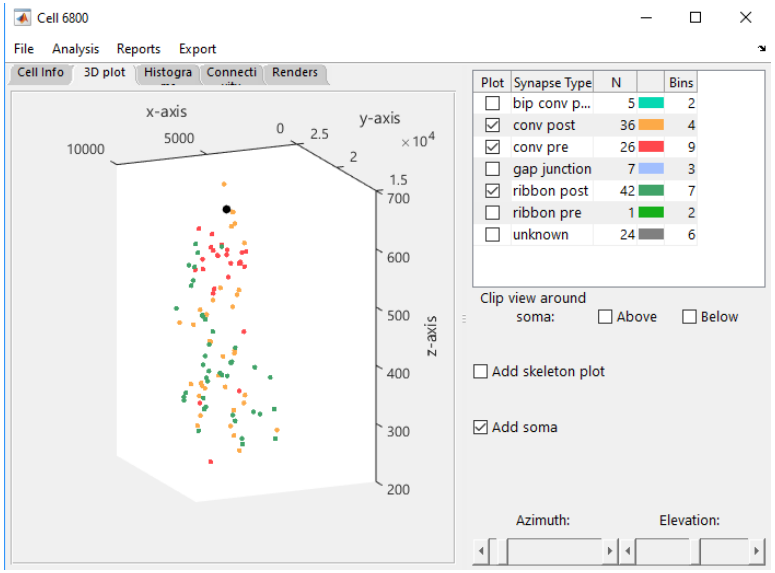
- If known, the cell type and subtype will be helpful for connectivity analysis.
- The other properties will eventually be used for cell queries but don't have much use yet.

After changing any of the attributes on the Cell Info Panel, make sure to press the `[Add to cell data]` button. This will make sure your changes are reflected next time you open the UI.

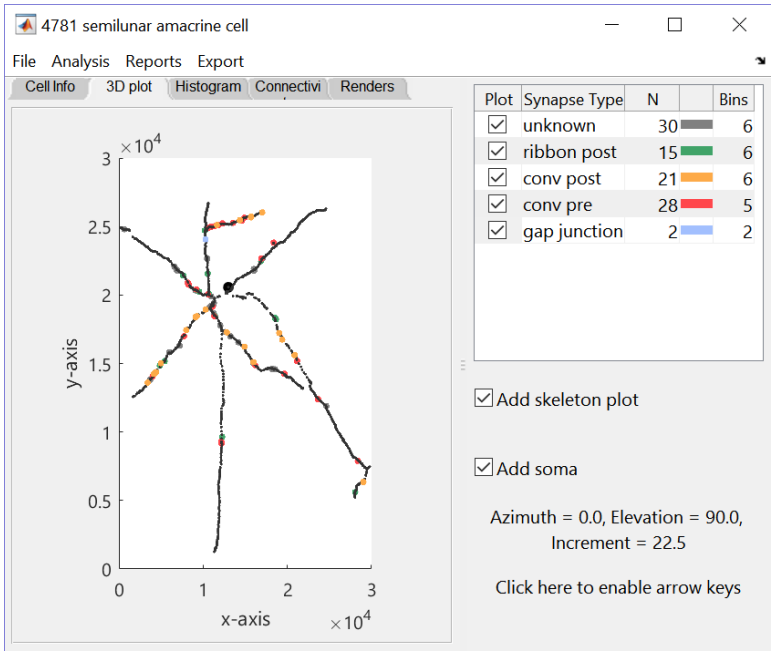
3d plot

Components

You can add and remove each synapse type, the soma node and the skeleton independently using the checkboxes.
Rotate the plot with the elevation and azimuth sliders.



All amacrine cell synapses



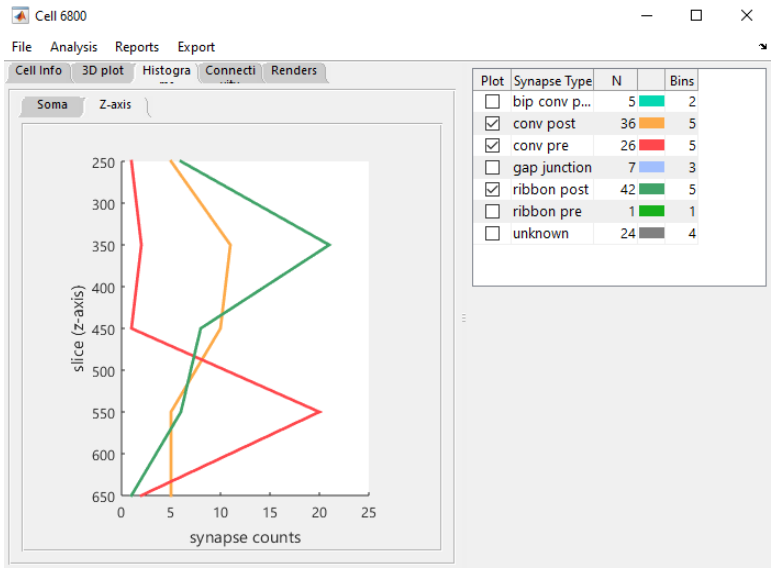
Semilunar synapses & skeleton

Histograms

There are two histograms. These plot synapse count as a function of:

- Distance from soma
- Section number (z-axis)

Use the Synapse Table to edit the number of bins. Add the cell skeleton to see dendrite stratification.

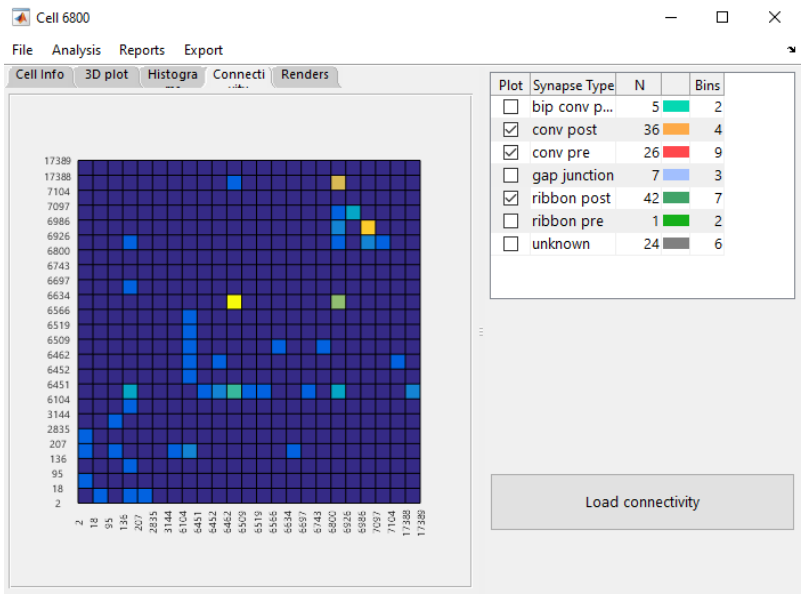


Z-axis synapse distribution for a putative AII AC. This synapse asymmetry isn't news but is still good to see.

Connectivity

To add connectivity data, save a network map in Tulip as a JSON file, as described above. Then in Matlab:

```
c207.addConnectivity('c207hops.json');
```



The connectivity matrix is weighted by the number of unique synapses between two cells (dark blue for 0 synapses). Directed synapses will only register a contact from the pre $\rightarrow$ post-synaptic neuron.

Connectivity for the AII AC.

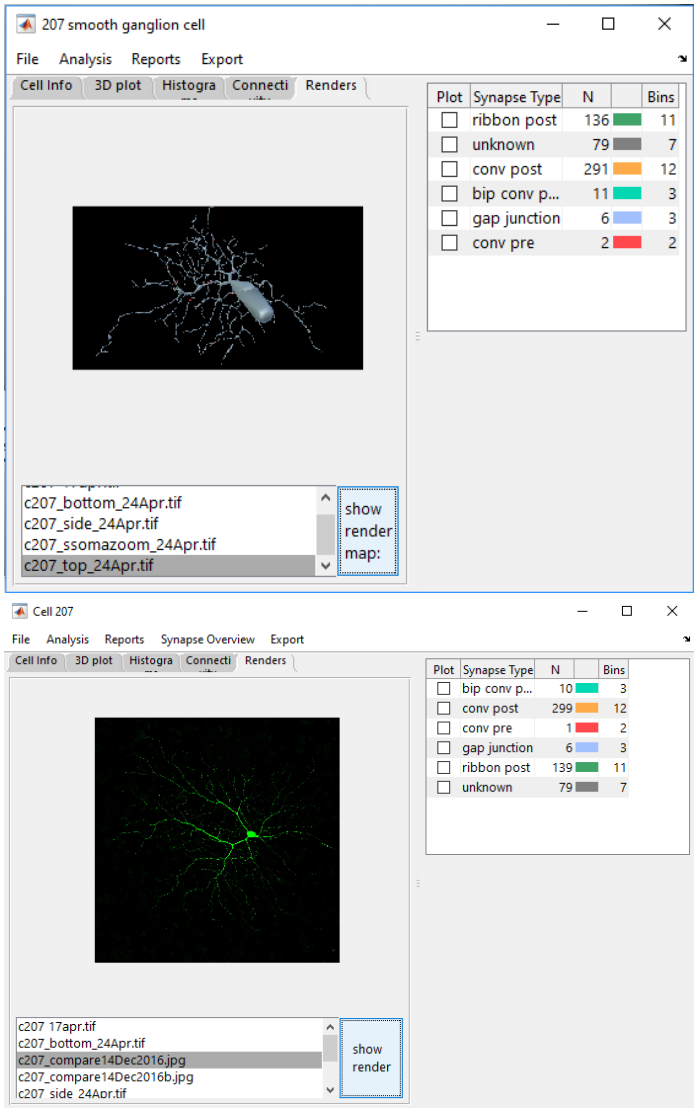
Connectivity

The network data is split into edges and nodes, each with their own table. The tables can be exported to Excel as `.csv` or as a text file from the UI menu bar.

To print a easily readable table to the command line use the networkTable function

```
1 networkTable(c207);
```

Blender Renders and Cell Fills



For now, set the `renderDir` in `getFilepaths.m` to the file your images are saved into. The UI find the images if their filename includes the letter 'c' followed by cell number (like 'c207'). I hope to improve this at some point.

This isn't limited to renders and could include whatever images and diagrams are helpful. For example, I can compare my ON-smooth cell reconstructions and cell fills.