**Viz: Brain Visualizer**

**Interactively Rendering BOSS Input and Output**

Remy Oukaour

Dept. of Computer Science

Stony Brook University

Stony Brook, NY 11794

November 5, 2014

Contents

[1. Introduction 4](#_Toc402545530)

[2. Version History 4](#_Toc402545531)

[3. Compilation 5](#_Toc402545532)

[3.1. FLTK Library 5](#_Toc402545533)

[3.2. Compilation Flags 5](#_Toc402545534)

[3.3. Visual Studio on Windows 6](#_Toc402545535)

[3.4. clang++ on Mac OS X 7](#_Toc402545536)

[3.5. g++ on Linux 8](#_Toc402545537)

[4. File Formats 9](#_Toc402545538)

[4.1. Binary Model Files 9](#_Toc402545539)

[4.2. Text Model Files 11](#_Toc402545540)

[4.3. Compressed Model Files 12](#_Toc402545541)

[4.4. Firing Spike Files 12](#_Toc402545542)

[4.5. Voltage Files 12](#_Toc402545543)

[4.6. Weight Files 13](#_Toc402545544)

[4.7. Pruning Files 14](#_Toc402545545)

[4.8. Selected Soma Report Files 14](#_Toc402545546)

[4.9. Selected Synapse Report Files 14](#_Toc402545547)

[4.10. Conﬁguration Files 15](#_Toc402545548)

[5. Internal Data Structures 15](#_Toc402545549)

[5.1. Coordinates 15](#_Toc402545550)

[5.2. Brain Models 16](#_Toc402545551)

[5.3. Simulation Data 16](#_Toc402545552)

[6. User Interface 17](#_Toc402545553)

[6.1. Menu Bar 19](#_Toc402545554)

[6.2. Toolbar 20](#_Toc402545555)

[6.3. Sidebar 23](#_Toc402545556)

[6.3.1. Overview Area 24](#_Toc402545557)

[6.3.2. Conﬁguration Tab 24](#_Toc402545558)

[6.3.3. Selection Tab 26](#_Toc402545559)

[6.3.4. Marking Tab 26](#_Toc402545560)

[6.4. Model Area 27](#_Toc402545561)

[6.5. Status Bar 28](#_Toc402545562)

[6.6. Simulation Bar 28](#_Toc402545563)

[6.6.1. Static Model 29](#_Toc402545564)

[6.6.2. Firing Spikes 29](#_Toc402545565)

[6.6.3. Voltages 30](#_Toc402545566)

[6.6.4. Weights 31](#_Toc402545567)

[7. Performance 33](#_Toc402545568)

[7.1. Loading Times 33](#_Toc402545569)

[7.2. Memory Usage 33](#_Toc402545570)

[7.3. Frame Rate 35](#_Toc402545571)

[A. Screenshots 36](#_Toc402545572)

[A.1. Synaptic Connections 36](#_Toc402545573)

[A.2. Neuritic Fields 39](#_Toc402545574)

[A.3. Selection 40](#_Toc402545575)

[A.4. Clipping 43](#_Toc402545576)

[A.5. Rotation 45](#_Toc402545577)

[A.6. Conﬁguration 45](#_Toc402545578)

[A.7. Firing Data 48](#_Toc402545579)

[B. Source Code Files 50](#_Toc402545580)

# Introduction

The Brain Organization Simulation System (BOSS) developed at Stony Brook University models electrical activity in neuron networks. It can simulate up to 131 billion synapses and 16,000 dendritic synapses per neuron. It runs on 1,024 nodes of an IBM Blue Gene/P supercomputer, each of which has two gigabytes of memory. The most recent version of BOSS, V8, uses the Izhikevich integrate-and-ﬁre neuron model, a more biologically accurate one than the threshold elements previously used.

BOSS runs in two stages: the initializer, INIT, generates a cerebellar model, and then the simulator, RUNSIM, processes activity in the model over time. Simulating one second of activity with 100 billion synapses takes two to four hours, depending on the neurons’ ﬁring frequencies. It is thus important to verify the model’s accuracy before running BOSS. The Brain Visualizer, or Viz, renders models on a desktop computer. Users can explore the model and check that its neurons and neuritic ﬁelds are as expected. With the most recent version of Viz, they can also visualize the output of a BOSS simulation—data about neurons’ ﬁring patterns, voltages, and synaptic weights.

# Version History

The ﬁrst version of the Brain Visualizer was written in July 2010 by Pablo Montes Arango. It displayed somas and synaptic connections in 3D and allowed users to rotate, translate, and zoom the model. Individual somas could be selected to show their synapses and neuritic ﬁelds in a separate window.

The second version, by Jordan Ponzo and Vikas Ashok in October 2012, added more display options: somas could be hidden on a per-type basis, synapses could be drawn through intermediate locations instead of just direct connections between somas, and the colors used for soma types could be set with a conﬁguration ﬁle. It also reduced memory usage with a data structure by Jack Zito: a table of all possible soma-to-soma pairs was replaced with an array of only the ones actually connected by a synapse.

The third version, by Remy Oukaour in December 2012, improved performance by caching data that did not need to be recalculated, and only redrawing the model when the view changed. The interface was made easier to use by allowing navigation states to be saved and restored, and by rearranging the controls and giving them more descriptive labels.

The fourth version was a complete rewrite using a diﬀerent UI toolkit (FLTK instead of GLUT). It consolidated the visualization of somas, synapses, and neuritic ﬁelds into one window, with an interface that more closely followed OS conventions. Navigation was given a full undo/redo history. Performance was improved, supporting models with more than one million somas and 100 million synapses. It also added the option to view the dynamic ﬁring data output by RUNSIM as animated color changes.

The current version of Viz has had its version number raised to 8, to match that of BOSS. It continues development from version 4, adding the ability to visualize gap junctions, somas’ voltages, and synapses’ weights.

# Compilation

Viz is written in C++ and can be compiled on diﬀerent operating systems, including Microsoft Windows, Apple Mac OS X, and GNU/Linux.

## FLTK Library

Viz depends on a modiﬁed version of FLTK (“Fast Light Toolkit”), a cross-platform GUI toolkit written in C++ that supports OpenGL. The previous version used GLUT (“OpenGL Utility Toolkit”), which was less customizable and had extant bugs after eight years of no development. FLTK can be downloaded from [www.ﬂtk.org](http://www.fltk.org) and built with the same compilers as Viz (Visual Studio, clang++, or g++).

Starting with a copy of FLTK v1.3.3, the ﬁle FL\Enumerations.H, line 30:  
//#define FLTK\_ABI\_VERSION 10303  
has been uncommented:  
#define FLTK\_ABI\_VERSION 10303  
This enables features which break the application binary interface (ABI) of v1.3.x. FLTK v1.4.0 is expected to enable the new ABI by default.

## Compilation Flags

It is recommended to use the maximum optimization possible for your chosen compiler (“/O2” for Visual Studio, “-O3” for clang++ or g++).

By default Viz represents coordinates as single-precision (32-bit) ﬂoating-point values. To use 16-bit signed integers instead, deﬁne SHORT\_COORDS during compilation. To use 32-bit integers, deﬁne INT\_COORDS. The About dialog will indicate the coordinate type. Short coordinates use less memory, but ﬂoating-point values are usually faster on dedicated graphics processors. Short coordinates may not be able to store the full range of coordinates for some models.

The interface uses a 12-point font by default. To use a 16-point font, which is easier to read on high-DPI displays, deﬁne LARGE\_INTERFACE during compilation.

## Visual Studio on Windows

Follow these instructions to build FLTK for Windows using Visual Studio 2010, 2012, or 2013:

1. Extract the contents of viz\lib\ﬂtk-1.3.3-mod.zip to ﬂtk-1.3.3-mod.
2. Open ﬂtk-1.3.3-mod\ide\VisualC2012\ﬂtk.sln in Visual Studio.
3. Set the solution conﬁguration to Release.
4. Set the solution platform to x64 or x86, depending on which architecture you plan to use.
5. Build the ﬂtk, ﬂtkgl, and ﬂtkimages projects.

After FLTK is built, follow these instructions to build Viz:

1. Copy the .lib ﬁles (ﬂtk.lib, ﬂtkgl.lib, ﬂtkimages.lib, ﬂtkpng.lib, and ﬂtkz.lib) from ﬂtk-1.3.3-mod\lib to viz\lib\win32\x64 or viz\lib\win32, depending on which architecture they were built for. You may need to rename fltkzlib.lib to fltkz.lib.
2. Navigate to viz\ide\vs2010, viz\ide\vs2012, or viz\ide\vs2013, depending on which version of Visual Studio you are using, and open Brain Visualizer.sln in Visual Studio.
3. Set the solution conﬁguration to Release and the solution platform to x64 or x86.
4. Build the Brain Visualizer project.
5. Run viz\bin\win32\x64\Release\viz.exe or viz\bin\win32\Release\viz.exe, depending on which solution platform was used.

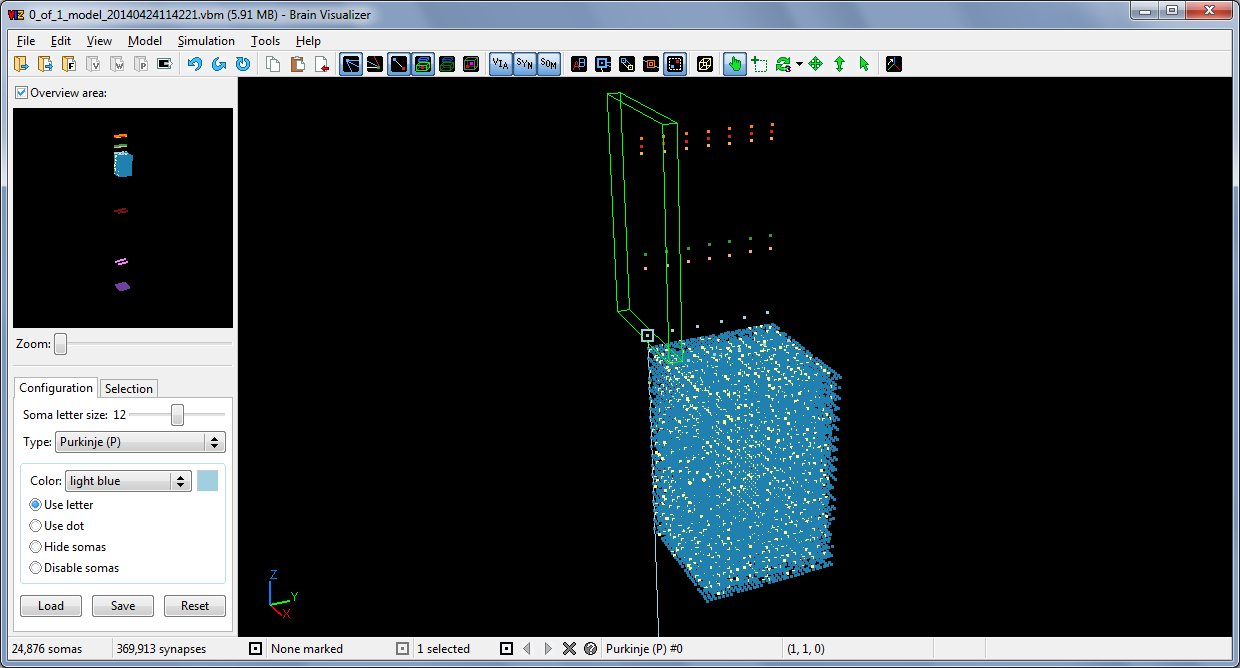


Figure 1: A screenshot of Viz running in Windows 7.

These are the custom properties needed for the Brain Visualizer project:

* General → Platform Toolset = v100 for 2010, v110 for 2012, or v120 for 2013
* General → Output Directory = ..\..\bin\win32\$(Conﬁguration)\
* General → Intermediate Directory = ..\..\tmp\win32\$(Conﬁguration)\
* Debugging → Working Directory = ..\..\conﬁg
* C/C++ → Additional Include Directories = ..\..\include;..\..\res
* C/C++ → Optimization → Optimization = Maximize Speed (/O2)
* C/C++ → Optimization → Enable Intrinsic Functions = Yes (/Oi)
* C/C++ → Optimization → Favor Size or Speed = Favor fast code (/Ot)
* C/C++ → Preprocessor → Preprocesor Deﬁnitions = WIN32;NOMINMAX; \_CRT\_SECURE\_NO\_WARNINGS;NDEBUG;\_WINDOWS;%(PreprocessorDefinitions)
* C/C++ → Code Generation → Floating Point Model = Fast (/fp:fast)
* C/C++ → Advanced → Disable Speciﬁc Warnings = 4068;4351 (/wd"4068;4351")
* Linker → Additional Library Directories = ..\..\lib\win32\x64 or ..\..\lib\win32
* Linker → Input → Additional Dependencies = ﬂtkimages.lib;ﬂtkpng.lib;ﬂtkz.lib;ﬂtk.lib; ﬂtkgl.lib;opengl32.lib;glu32.lib;comctl32.lib;%(AdditionalDependencies)
* Linker → Input → Ignore Speciﬁc Default Libraries = libcmt.lib; %(IgnoreSpeciﬁcDefaultLibraries)

## clang++ on Mac OS X

Follow these instructions to build FLTK for Mac OS X using clang++ (which is included in the Command Line Tools package for Xcode 5):

1. Extract the contents of viz/lib/ﬂtk-1.3.3-mod.zip to ﬂtk-1.3.3-mod.
2. Navigate to ﬂtk-1.3.3-mod and run the command:  
   make

After FLTK is built, follow these instructions to build Viz:

1. Copy the .a ﬁles (libﬂtk.a, libﬂtk\_gl.a, libﬂtk\_images.a, and libﬂtk\_png.a) from ﬂtk-1.3.3-mod/lib to viz/lib/osx.
2. Navigate to viz/ide/clang and run the command:  
   make viz
3. Run the viz/bin/osx/Brain Visualizer application.

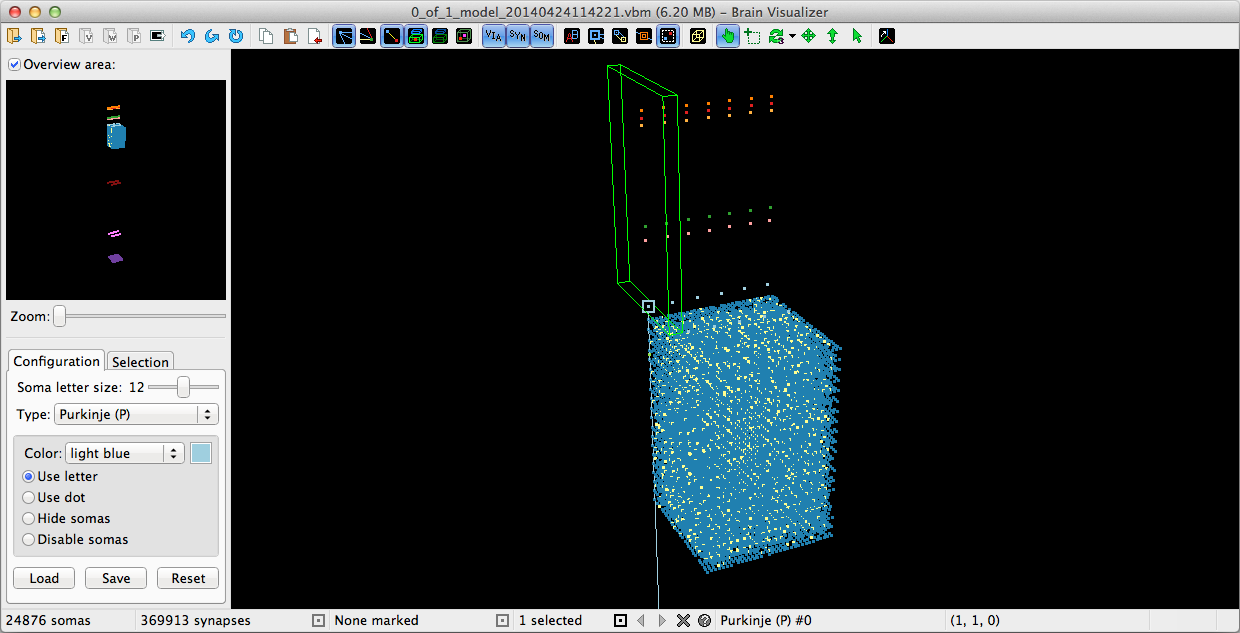


Figure 2: A screenshot of Viz running in Mac OS X 10.9.

## g++ on Linux

Follow these instructions to build FLTK for Linux using g++:

1. To ensure that all dependencies are installed, run the command:  
   sudo apt-get install g++ make freeglut3-dev zlib1g-dev libpng-dev libxpm-dev libx11-dev libxft-dev libxinerama-dev libfontconfig1-dev x11proto-xext-dev
2. Extract the contents of viz/lib/ﬂtk-1.3.3-mod.zip to ﬂtk-1.3.3-mod.
3. Navigate to ﬂtk-1.3.3-mod and run the commands:  
   chmod +x configure  
   sudo make

After FLTK is built, follow these instructions to build Viz:

1. Copy the .a ﬁles (libﬂtk.a, libﬂtk\_gl.a, and libﬂtk\_images.a) from ﬂtk-1.3.3-mod/lib to viz/lib/linux.
2. Navigate to viz/ide/gcc and run the command:  
   make viz
3. Run viz/bin/linux/viz.

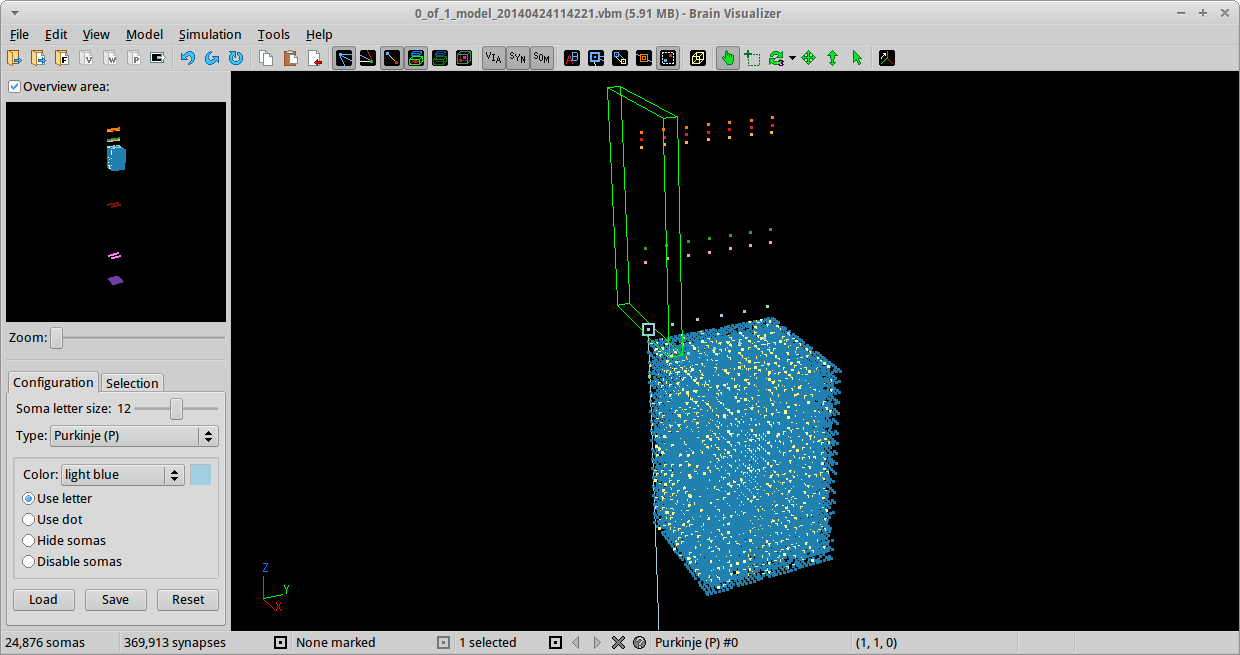


Figure 3: A screenshot of Viz running in Xubuntu Linux 14.04.

# File Formats

Most ﬁles that Viz reads are interpreted as lines of ASCII text with platform-dependent line breaks. Binary ﬁles have their own speciﬁcations.

## Binary Model Files

Model ﬁles are generated by INIT to deﬁne the locations of somas and synapses in a particular model. Model ﬁles use the .vbm extension (for “Viz binary model”). This is a binary format, with all numbers represented as variable-length integers in network byte order (big-endian).

Variable-length integers (varints) allow small, common values to take up fewer bytes than large ones. An unsigned 64-bit integer can be encoded in 1 to 9 bytes; a signed 32-bit integer can be encoded in 1 to 5 bytes.

The leading bits in the ﬁrst byte of a varint indicate how many additional bytes there are. The byte starts with zero to seven 1s, followed by a 0; any additional bits are used to encode data. The ﬁrst byte may also be eight 1s. Decoding is as follows:

|  |  |  |
| --- | --- | --- |
| **First byte** | **Unsigned** | **Signed** |
| 0xxxxxxx | Read 0 more bytes | Read 0 more bytes |
| 10xxxxxx | Read 1 more byte | Read 0 more bytes and negate |
| 110xxxxx | Read 2 more bytes | Read 1 more byte |
| 1110xxxx | Read 3 more bytes | Read 1 more byte and negate |
| 11110xxx | Read 4 more bytes | Read 2 more bytes |
| 111110xx | Read 5 more bytes | Read 2 more bytes and negate |
| 1111110x | Read 6 more bytes | Read 3 more bytes |
| 11111110 | Read 7 more bytes | Read 3 more bytes and negate |
| 11111111 | Read 8 more bytes | Read 4 more bytes and negate if the leading bit is 1 |

Table 1: Lookup table of initial bytes for decoding variable-length integers.

The ﬁrst ﬁve bytes of a model ﬁle are 07 52 4A 56 F7 in hexadecimal. This is a signature identifying the ﬁle as a .vbm ﬁle. Next is a single-byte unsigned integer *v*, which is the ﬁle format version. Version 2 is described here. 0 is not a valid version. Next is a null-terminated string *s*, which is an optional comment string for additional information. The string *s* may be empty, meaning it consists only of the terminating byte 00.

Next is an unsigned varint *n*, which is the number of cell types deﬁned by the following *n* bytes. Each type deﬁnition takes one byte: an ASCII character *ci*, which is the letter used to represent all cells of the type with index *i*.

Next is an unsigned varint *m*, which is the number of somas deﬁned by the following bytes, and an unsigned varint *f*, which is the total number of neuritic ﬁelds belonging to the somas. (In version 1 ﬁles, *f* is not present.) Each soma deﬁnition takes seven varints: *t k x y z a d*. The unsigned soma type index is *t*. The unsigned soma ID is *k*. The signed coordinates of the cell are *x*, *y*, and *z*. The unsigned number of axonal ﬁelds for the cell is *a*, and the unsigned number of dendritic ﬁelds is *d*.

The bytes after a soma deﬁnition deﬁne axonal and dendritic ﬁelds for the cell. Each ﬁeld deﬁnition takes six signed varints: *x*1 *x*2 *y*1 *y*2 *z*1 *z*2. These are the coordinates of the diagonal of the axis-aligned bounding box for the neuritic ﬁeld.

Next after the *m* soma deﬁnitions is an unsigned varint *p*, which is the number of synapses deﬁned by the following bytes. Each synapse deﬁnition starts with an unsigned varint *k* and a single-byte integer *v*. The synapse ID is *k*, and the *via* point ﬂag is *v*. If *v* is 0, the synapse does not have a *via* point and its deﬁnition takes ﬁve more varints: *a d x y z*. The unsigned axonal soma ID is *a* and the unsigned dendritic soma ID is *d*. The signed coordinates of the synapse are *x*, *y*, and *z*. If *v* is 1, the synapse has a *via* point and its deﬁnition takes eight more varints: *a d* *vx* *vy* *vz* *x* *y* *z*. The signed coordinates of the synapse’s *via* point are *vx*, *vy*, and *vz*. The other ﬁve integers are the same as for synapses without a *via* point.

Next after the *p* synapse deﬁnitions is an unsigned varint *g*, which is the number of gap junctions deﬁned by the following bytes. Each gap junction deﬁnition takes ﬁve varints: *s*1 *s*2 *x* *y* *z*. The two unsigned somas’ IDs are *s*1 and *s*2. The signed coordinates of the gap junction are *x*, *y*, and *z*.

## Text Model Files

Model ﬁles used to be generated by INIT in a text-based format, but the current binary format takes up less space. Viz still supports either format.

Legacy model ﬁles use the .txt extension. Tokens are separated by one or more whitespace characters. The “#” character comments out all text until the end of the line. Blank lines and entirely commented lines are ignored.

The ﬁrst line has a single integer *n*, which is the number of cell types. The following *n* lines each have a single integer *t* followed by a single letter *c*. The type index, *t*, increases from 0 to *n* − 1. The letter, *c*, is used to represent all cells of that type.

The next line has a single integer *m*, which is the number of somas deﬁned by the following lines. Each soma is deﬁned by a line that has seven integers: *t k x y z a d*. The soma type index is *t*. The soma ID is *k*. The coordinates of the cell are *x*, *y*, and *z*. The number of axonal ﬁelds for the cell is *a*, and the number of dendritic ﬁelds is *d*.

The next *a* + *d* lines after a cell deﬁnition line deﬁne axonal and dendritic ﬁelds for the cell. Each line has six integers: *x*1 *x*2 *y*1 *y*2 *z*1 *z*2. These are the coordinates of the diagonal of the axis-aligned bounding box for the neuritic ﬁeld.

The line after the *m* cell deﬁnitions has a single integer *p*, which is the number of synapses deﬁned by the following *p* lines. Synapses without a *via* point are deﬁned by a line that has six integers: *k a d x y z*. The synapse ID is *k*. The axonal soma ID is *a* and the dendritic soma ID is *d*. The coordinates of the synapse are *x*, *y*, and *z*. Synapses with a *via* point are deﬁned by a line that starts with a single integer *k*, followed by the letter “v”, followed by eight more integers: *a d vx vy vz x y z*. The coordinates of the synapse’s *via* point are *vx*, *vy*, and *vz*. The other six integers are the same as for synapses without a *via* point.

The line after the *p* synapse deﬁnitions has a single integer *g*, which is the number of gap junctions deﬁned by the following *g* lines. Each gap junction is deﬁned by a line that has ﬁve integers: *s*1 *s*2 *x y z*. The two somas’ IDs are *s*1 and *s*2. The coordinates of the gap junction are *x*, *y*, and *z*.

For backwards compatibility with ﬁles generated before the introduction of gap junctions, everything related to them can be left out, and it will be assumed that there are no gap junctions.

## Compressed Model Files

Both binary and text model ﬁles can be compressed with the DEFLATE algorithm and stored using the gzip ﬁle format. Programs like 7-Zip and GNU Gzip can compress .vbm or .txt ﬁles into .gz ﬁles, which may be less than 20% of their original size. Viz supports opening compressed model ﬁles directly.

Compressed model ﬁles use the .vbm.gz or .txt.gz extensions.

## Firing Spike Files

Firing spike ﬁles are generated by RUNSIM to record the ﬁring spikes of the somas being simulated.

Firing spike ﬁles use the .txt extension. Tokens are separated by one or more whitespace characters. The “#” character comments out all text until the end of the line. Blank lines and entirely commented lines are ignored.

The ﬁrst line optionally has the letter “v” followed by a single integer *v*, expected to be 2, which is the version of the ﬁring spike ﬁle format. This is only for backwards compatibility with old ﬁring spike ﬁles, and will be removed in a future version of Viz.

The second line has a single integer *t*, expected to be from 100 to 10,000, which is the number of microseconds (µs) per cycle. The third line has a single integer *m*, which is the number of somas deﬁned by the corresponding model ﬁle. The fourth line has a single integer *c*, which is the number of cycles deﬁned by the following *c* lines.

Each cycle deﬁnition line starts two integers: *s* *t*. The time instance being deﬁned by the line is *t*. The number of somas that ﬁred at cycle *t* is *s*. In version 2 ﬁles, these are followed by *s* pairs of integers, or 2*s* integers total. In each pair *pi*, the ﬁrst integer is the ID of the ﬁring soma, and the last integer is the state of the ﬁring soma: a bitmask of 1 for natural ﬁring, 2 for forced ﬁring, 4 for binary ﬁring, and 8 for ﬁring suppression. (Suppressions are ﬁring events that aren’t associated with an Izhikevich ﬁring spike; rosette somas can exhibit them.) In ﬁles without a version line, implicitly treated as version 1, there are simply *s* ﬁring soma IDs without corresponding ﬁring states, and the states are assumed to be 1 for all ﬁring events.

## Voltage Files

Voltage ﬁles are generated by RUNSIM to record the voltages of particular somas being simulated. Not all the somas are recorded, since that would usually be too much data. The somas to record are speciﬁed by the selected soma report ﬁle, viz\_selected\_cells.txt.

Voltage ﬁles use the .txt extension. Tokens are separated by one or more whitespace characters. The “#” character comments out all text until the end of the line. Blank lines and entirely commented lines are ignored.

The ﬁrst line has a single integer *m*, which is the number of somas deﬁned by the corresponding model ﬁle. The second line has a single integer *n*, which is the number of somas which have voltages recorded in the ﬁle. The third line has *n* integers, which are the IDs of the somas with recorded voltages. The fourth line has a single integer *c*, which is the number of cycles deﬁned by the following *c* lines.

Each cycle deﬁnition line starts with a single integer *t*, which is the time instance being deﬁned by the line. It is followed by *n* pairs of integers, or 2*n* integers total. In each pair *pi*, the ﬁrst integer is the voltage of the *i*th soma at cycle *t* in millivolts (mV), and the last integer is the state of the ﬁring soma: a bitmask of 1 for natural ﬁring, 2 for forced ﬁring, 4 for binary ﬁring, and 8 for ﬁring suppression.

## Weight Files

Weight ﬁles are generated by RUNSIM to record the weight changes of particular synapses being simulated. Not all the synapses are recorded, since that would usually be too much data. The synapses to record are speciﬁed by the selected soma report ﬁle, viz\_selected\_cells.txt.

Weight ﬁles use the .txt extension. Tokens are separated by one or more whitespace characters. The “#” character comments out all text until the end of the line. Blank lines and entirely commented lines are ignored.

The ﬁrst line has a single integer *m*, which is the number of somas deﬁned by the corresponding model ﬁle. The second line has a single integer *y*, which is the number of synapses deﬁned by the corresponding model ﬁle. The third line has a single integer *c*, which is the number of cycles deﬁned by the corresponding ﬁring spike ﬁle. The following lines deﬁne weight changes for individual synapses.

Each change deﬁnition line has seven integers: *t k a d r wb wa*. The cycle index is *t*. The synapse ID is *k*. The axonal soma ID is *a* and the dendritic soma ID is *d*. The RUNSIM synapse ID is *r*. (It is not used by Viz, but is recorded to reﬂect the reordering of synapses between INIT and RUNSIM.) The weight before the change is *wb* and the weight after the change is *wa*. Weights are measured in microvolts (µV).

## Pruning Files

Pruning ﬁles are generated by RUNSIM to record the weight changes which occur as a result of pruning. They are formatted identically to weight ﬁles.

## Selected Soma Report Files

Soma report ﬁles are generated by Viz to record the selected somas. BOSS uses the soma report ﬁle in its directory to identify somas for which to output voltage and weight ﬁles.

Soma report ﬁles use the .txt extension. Tokens are separated by one or more whitespace characters. The “#” character comments out all text until the end of the line. Blank lines and entirely commented lines are ignored.

The ﬁrst line has a single integer *m*, which is the number of somas described by the following *m* lines. Each soma description line has ﬁve integers: *t k x y z*. The soma type index is *t*. The soma ID is *k*. The coordinates of the cell are *x*, *y*, and *z*.

After the *m* soma descriptions, the optional line “-1 0 0 0 0” prevents BOSS from ﬁnding the parents and children of the *m* somas.

## Selected Synapse Report Files

Synapse report ﬁles are generated by Viz to record the synapses to or from the selected somas. This limited subset of synapses can be more easily read than an entire model ﬁle.

Synapse report ﬁles use the .txt extension. Tokens are separated by one or more whitespace characters. The “#” character comments out all text until the end of the line. Blank lines and entirely commented lines are ignored.

The ﬁrst line has a single integer *m*, which is the number of somas described by the following *m* lines. Each soma description line has ﬁve integers: *t k x y z*. The soma type index is *t*. The soma ID is *k*. The coordinates of the cell are *x*, *y*, and *z*.

The line after the *m* cell descriptions has a single integer *y*, which is the number of synapses described by the following *y* lines. Each synapse description line has six integers: *k a d x y z*. The synapse ID is *k*. The axonal soma ID is *a* and the dendritic soma ID is *d*. The coordinates of the synapse are *x*, *y*, and *z*.

## Conﬁguration Files

Conﬁguration ﬁles use the .cfg extension. Fields are separated by single “:” characters. The “#” character comments out all text until the end of the line. Blank lines and entirely commented lines are ignored.

Each line must have at least four ﬁelds. The ﬁrst is a single letter indicating which soma type the line is conﬁguring. The second is the name of the type. The third is the color used to represent the type. There are 24 valid colors: red, maroon, pink, magenta, orange, tan, brown, yellow, goldenrod, olive, yellow-green, green, lime, cyan, turquoise, blue, sky blue, light blue, indigo, violet, purple, lavender, white, and gray. Unknown colors are treated as gray. The fourth is the display status of the type: letter, dot, hidden, or disabled. Any extra ﬁelds are ignored.

Viz includes a hard-coded default conﬁguration equivalent to the following conﬁguration ﬁle:

P:Purkinje:light blue:letter

N:Granule:blue:dot

G:Golgi:yellow-green:letter

B:Basket +x:green:letter

A:Basket -x:pink:letter

S:Outer stellate +x:red:letter

T:Outer stellate -x:tan:letter

I:Outer stellate A:orange:letter

C:Climbing fiber:lavender:letter

M:Mossy fiber:purple:letter

R:Rosette tip:yellow:dot

D:Dentate nucleus:maroon:letter

# Internal Data Structures

The data structures used by Viz are tuned for performance, to allow storing as much data as possible and to speed up the common operations on that data.

## Coordinates

Depending on how Viz is built, coordinates may be represented in diﬀerent ways and have diﬀerent ranges. As single-precision ﬂoating-point values (the default format), they can unambiguously range from −224 − 1 to 224 − 1. As 16-bit integers (speciﬁed by deﬁning SHORT\_COORDS during compilation), they range from −215 to 215 − 1 (32,767). As 32-bit integers (speciﬁed by deﬁning INT\_COORDS during compilation), they range from −232 to 232 − 1.

## Brain Models

A brain model deﬁned by a model ﬁle has numerous soma types, somas, neuritic ﬁelds, synapses, and gap junctions.

Soma types are stored in a single array with between 1 and 28 − 1 (255) soma types.

Somas are stored in a single array with between 1 and 224 − 2 (more than 16 million) somas. Although a 32-bit integer is used to store the number of somas, selecting somas requires their indexes to be encoded as 24-bit RGB colors, with a single reserved background color. Each soma may have between 0 and 28 − 1 axonal ﬁelds, and likewise for dendritic ﬁelds. Neuritic ﬁelds are stored in a single array with between 0 and 232 − 1 ﬁelds.

Synapses are stored in a single array with between 0 and 232 − 1 (more than 4 billion) synapses.

Gap junctions are stored in a single array with between 0 and 232 − 1 gap junctions.

## Simulation Data

Each set of simulation data (ﬁring spikes, soma voltages, and synapse weights) has between 1 and 232 − 1 cycles. Cycle-speciﬁc data is stored in a single array. The data for each cycle is generally an associative map from soma indexes to their simulation data.

Voltages are recorded for a subset of active somas, which are stored in a single array with between 0 and 232 − 1 somas.

Synapse weights range from −214 to 214 − 1 (16,383) microvolts (µV), which is −163.84 to 163.83 millivolts (mV).

# User Interface

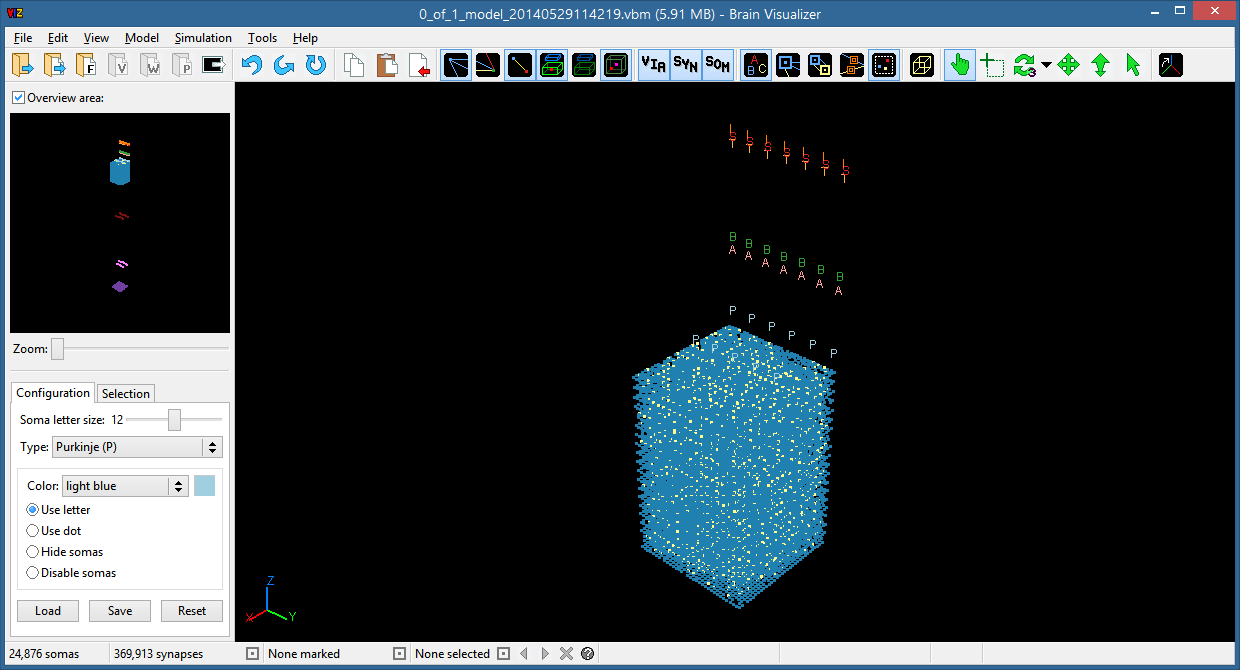


Figure 4: The Viz user interface in Windows 8 with a model ﬁle opened.

Viz uses a single window divided into sections: the menu bar at the very top, the toolbar below it, the sidebar on the left (containing the overview area at its top), the model area to the right, and the status bar at the bottom. The model area and overview area display the opened model; the status bar lists further information about the model; and the menu bar, toolbar, and sidebar contain controls that let the user interact with the model.

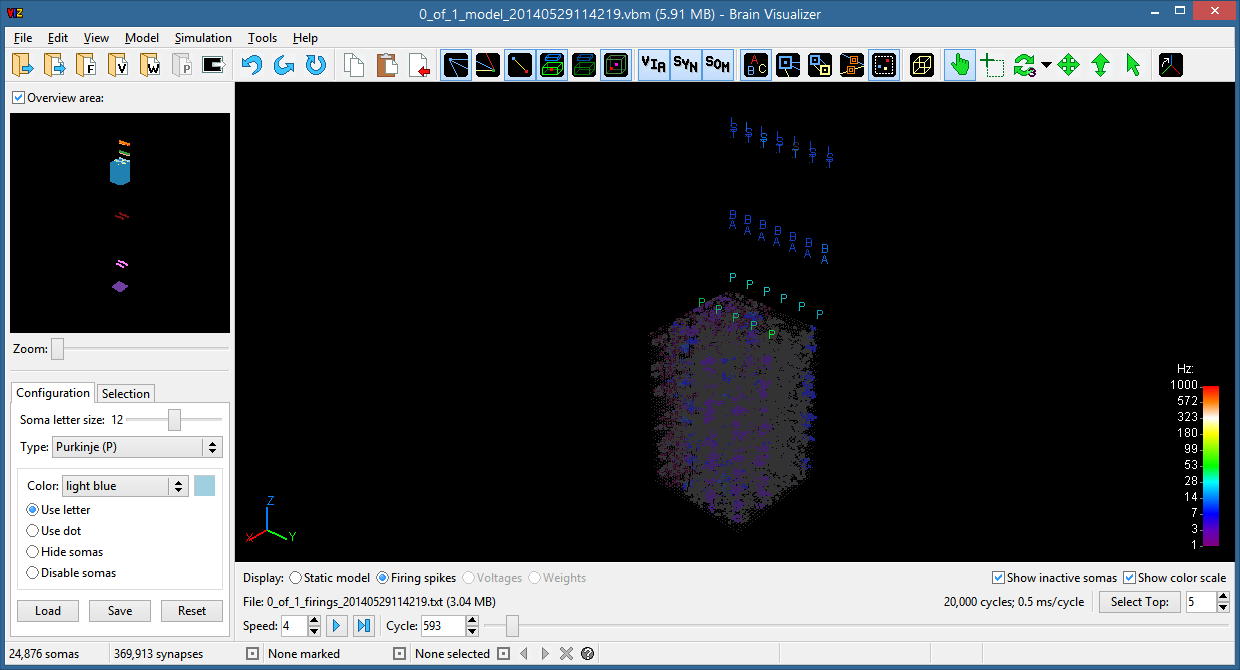


Figure 5: The Viz user interface with a ﬁring spike ﬁle loaded.

When a ﬁring spike ﬁle is loaded after a corresponding model ﬁle has been opened, the simulation bar appears below the model area, and the model area displays the model diﬀerently.

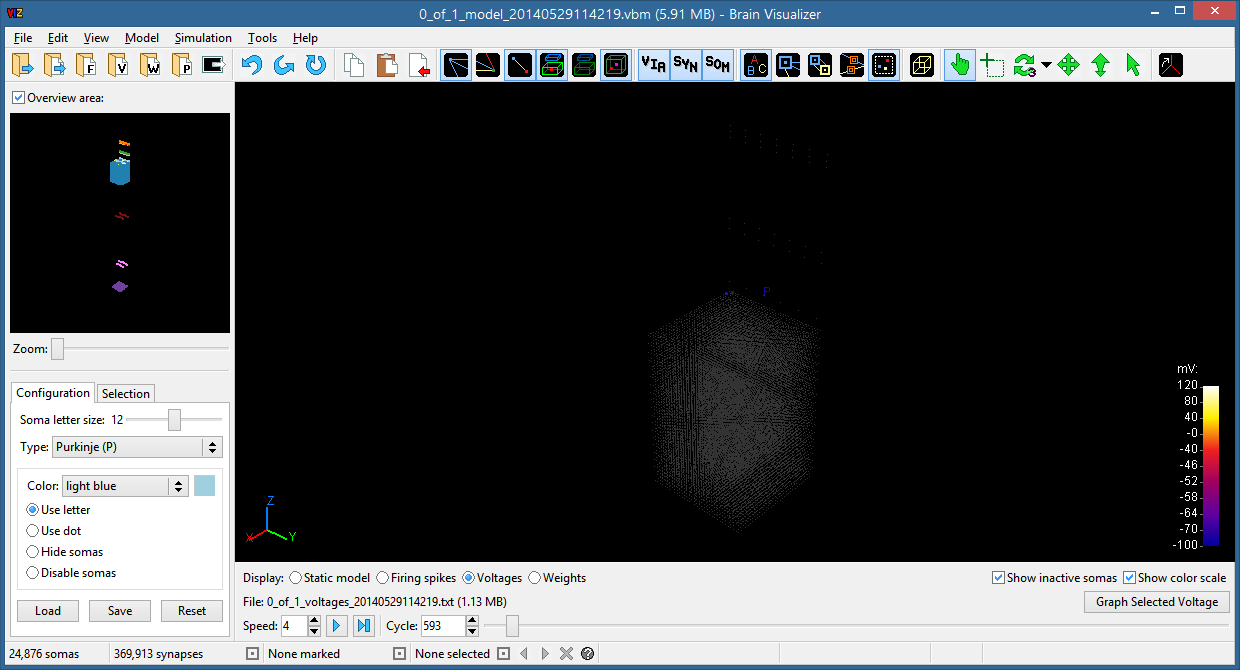


Figure 6: The Viz user interface with a voltage ﬁle and a weight ﬁle loaded.

After a ﬁring spike ﬁle has been loaded, voltage and weight ﬁles can be loaded as well.

## Menu Bar

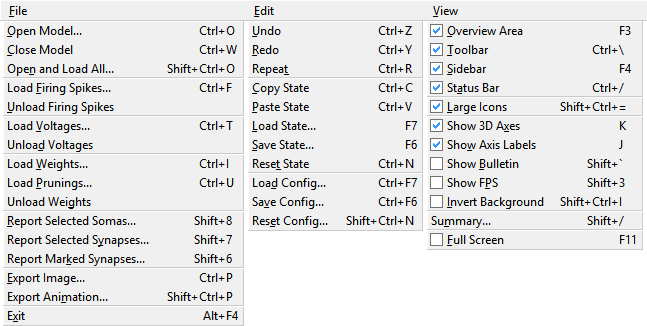


Figure 7: The ﬁrst three menus in the menu bar.

The File menu contains commands for dealing with most types of ﬁles: models, ﬁring spikes, voltages, weights, selected soma or synapse reports, and images. The Exit command is also located here on Windows by convention.

The Edit menu contains commands to manage the user’s sequence of navigation actions. Actions can be undone or redone, and the present state can be kept on the clipboard for short-term storage or saved to a ﬁle for a longer term. Commands to manage conﬁguration ﬁles are also located here.

The View menu contains commands to modify the interface’s appearance. The overall interface can be set to imitate diﬀerent operating systems. Certain parts of the user interface, and attributes of the model area, can be toggled on or oﬀ. The program can also enter full screen mode, which hides window chrome and takes up the whole screen.

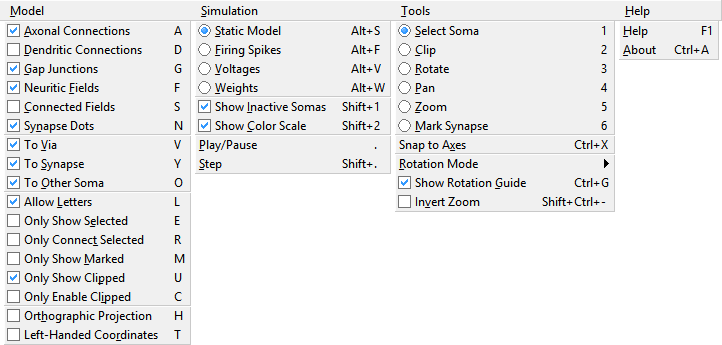


Figure 8: The last four menus in the menu bar.

The Model menu contains options that aﬀect how the model is displayed in the model area.

The Simulation menu contains options that aﬀect how simulation data (ﬁring spikes, voltages, or weights) is displayed in the model area, as well as commands to progress through the data automatically or step through it manually.

The Tools menu contains commands to interact with the model.

The Help menu contains commands to show information about the program. The Help dialog displays basic guidance from the ﬁle help.html, which should be in the same directory as the Viz executable ﬁle. The About dialog shows build information and development credits.

On Mac OS X, the Brain Visualizer application menu (left-most on the menu bar) contains commands dealing with the program itself, including the Exit item from the File menu and the About item from the Help menu. The Help menu has a search box to search the text of menu items.

## Toolbar



Figure 9: The toolbar, with large icons.

The toolbar contains buttons for the most useful commands. Each of them corresponds to a menu item. Some buttons execute a command when pressed; others toggle a setting on or oﬀ. Some buttons are disabled until a model ﬁle is open, or until a ﬁring spike ﬁle is loaded.

*  **Open Model:** Open a model ﬁle (using a platform-dependent ﬁle dialog).
*  **Open and Load All:** Open a model ﬁle and load its corresponding ﬁring spikes, voltages, and weights, if they exist.
*  **Load Firing Spikes:** Load a ﬁring spike ﬁle corresponding to the open model.
*  **Load Voltages:** Load a voltages ﬁle corresponding to the open model.
*  **Load Weights:** Load a weights ﬁle corresponding to the open model.
*  **Export Image:** Export the current model area as a PNG, BMP, TGA, or PPM image.
*  **Undo:** Undo the previous action (up to 50 times). Anything that modiﬁes the navigation state (the position and orientation of the model in 3D space, and the set of selected somas) counts as an action.
*  **Redo:** Redo the last undone action (up to 50 times).
*  **Repeat:** Repeat the previous rotate, pan, or zoom action relative to the current state. This is useful for making incremental rotations in the same direction.
*  **Copy:** Copy the current navigation state to the clipboard.
*  **Paste:** Paste the saved navigation state from the clipboard. If no state has been saved, this acts like Reset.
*  **Reset:** Reset the navigation state to the original when the model was ﬁrst opened.
*  **Axonal Connections:** Toggle showing axonal connections for selected somas (those from the selected somas to others).
*  **Dendritic Connections:** Toggle showing dendritic connections for selected somas (those from other somas to the selected ones).
*  **Gap Junctions:** Toggle showing gap junctions for selected somas.
*  **Neuritic Fields:** Toggle showing the neuritic ﬁelds for selected somas.
*  **Connected Fields:** Toggle showing the neuritic ﬁelds for somas connected to the selected somas. The connected ﬁelds are shown only if the Neuritic Fields option is on.
* E:\Desktop\syn-dots.png **Synapse Dots:** Toggle showing the locations of synapses inside neuritic ﬁelds. The synapses are shown only if the Neuritic Fields option is on.
*  **To Axonal Soma:** Toggle whether synaptic connections pass through the axonal soma’s location.
*  **To Via Point:** Toggle whether synaptic connections pass through the *via* point.
*  **To Synapse:** Toggle whether synaptic connections pass through the synapse’s location.
*  **To Dendritic Soma:** Toggle whether synaptic connections pass through the dendritic soma’s location.
*  **Allow Letters:** Toggle allowing somas to be displayed as letters if their types are conﬁgured for it. This is oﬀ by default to improve performance.
*  **Only Show Selected:** Toggle only showing selected somas (and the ones connected to them, if axonal or dendritic connections are shown).
* **E:\Desktop\conn-selected-24.png Only Connect Selected:** Toggle only showing synaptic connections between pairs of selected somas.
* **E:\Desktop\show-marked-24.png Only Show Marked:** Toggle only showing marked synapses.
*  **Only Show Clipped:** Toggle only showing somas inside the clipping region. The hidden outside somas will only be displayed if they are connected to a selected soma.
*  **Orthographic Projection:** Toggle the use of orthographic projection. The default perspective projection shows further-away points as closer together, whereas orthographic projection does not distort distance.
*  **Select:** Click a soma to select or deselect it. Hold down Shift or turn on Caps Lock while clicking to select multiple somas. Up to 1,000 somas can be selected at once. Click in empty space to deselect them. Hold down Ctrl while clicking a soma, or right-click it, to bring up a summary dialog with detailed information about the soma. Do the same in empty space for information about the whole model.
*  **Clip:** Click and drag to draw a box around a group of somas. The box will then be projected into 3D space, and all somas outside of it will be clipped. Somas outside the clipping region can be hidden or disabled. Clipping also sets the pivot point for rotations to be in the center of mass of the clipping region.
*  **Rotate:** Click and drag to rotate the model around the pivot point. By default, a wireframe sphere with three axes is superimposed on the model while rotating to make results clearer. There are ﬁve rotation modes, accessible via the dropdown arrow to the right of the Rotate button:
  + **2D Arcball:** Dragging left, right, up, and down rotates the model in the direction it was dragged. The initially clicked point is irrelevant; only the dragging direction aﬀects the model.
  + **3D Arcball:** Clicking a point ﬁguratively grabs a point on the sphere surrounding the clipping region (or the whole model, if it has not been clipped), and dragging moves that point to where it was dragged. This lets dragging in two dimensions aﬀect the orientation around all three axes. This is the default mode.
  + **X-Axis:** Dragging perpendicular to the *x*-axis rotates the model around that axis. If the *x*-axis is nearly perpendicular to the screen, then clicking ﬁguratively grabs a point on the cylinder surrounding the *x*-axis, and dragging moves that point to where it was dragged.
  + **Y-Axis:** Behaves like the *x*-axis mode, but for the *y*-axis.
  + **Z-Axis:** Behaves like the *x*-axis mode, but for the *z*-axis.
*  **Pan:** Click and drag to pan (translate) the model in the desired direction. Panning too far away from the center can cause parallax distortion when using perspective projection.
*  **Zoom:** Click and drag upwards to zoom in (making the model larger) and downwards to zoom out (making it smaller). Zooming is relative to the center of the clipping region. If Tools → Invert Zoom is on, the directions are reversed.
*  **Mark Synapse:** Click a synapse dot to mark or unmark it. Hold down Shift or turn on Caps Lock while clicking to mark multiple synapses. Click in empty space to unmark them. Hold down Ctrl while clicking a synapse, or right-click it, to bring up a summary dialog with detailed information about the synapse.
*  **Snap to Axes:** Rotate the model so that its axes are aligned straight horizontally and vertically.
* E:\Desktop\Untitled.png **Reset Settings:** Reset the menu and toolbar toggle settings to their default values.

## Sidebar

The sidebar contains the overview area, as well as tabs containing controls for conﬁguring the soma types, selecting sets of somas, and marking sets of synapses.

### Overview Area

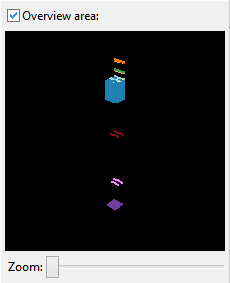


Figure 10: The overview area from the sidebar, fully zoomed out.

The overview area shows the entire model, including even the hidden or disabled somas. It zooms independently of the model area, and cannot be panned. Zooming completely out shows the entire model; zooming completely in focuses on the clipping region. The overview area can be minimized to improve performance and use less vertical space.

### Conﬁguration Tab

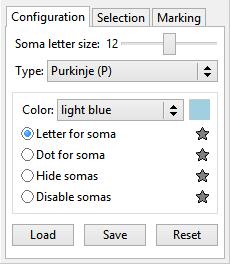


Figure 11: The conﬁguration tab from the sidebar.

From the conﬁguration tab, each soma type can have its color and display state conﬁgured. There are four display states:

* **Letters:** Somas of this type will be displayed as a letter if Allow Letters is on.
* **Dots:** Somas of this type will be displayed as a dot, even if Allow Letters is on.
* **Hidden:** Somas of this type will only be displayed if they are connected to a selected soma or a marked synapse.
* **Disabled:** Somas of this type will not be displayed at all.

The star buttons next to each display state will set all soma types to use that state.

The size of the letters used to display somas can be set to 8, 10, 12, 14, or 16 points.

The conﬁguration can be loaded from a ﬁle, saved to a ﬁle, or reset to its initially-loaded state. Overwriting an existing ﬁle by saving a new one with the same name is not allowed.

The initial conﬁguration for an opened model is loaded automatically from the ﬁle viz.cfg. If no such ﬁle exists, one is generated with these default data:

| **Letter** | **Name** | **Color** | **Display State** |
| --- | --- | --- | --- |
| P | Purkinje | light blue | letter |
| N | Granule | blue | dot |
| G | Golgi | yellow-green | letter |
| B | Basket +x | green | letter |
| A | Basket −x | pink | letter |
| S | Outer stellate +x | red | letter |
| T | Outer stellate –x | tan | letter |
| I | Outer stellate A | orange | letter |
| C | Climbing ﬁber | lavender | letter |
| M | Mossy ﬁber | purple | letter |
| R | Rosette tip | yellow | dot |
| D | Dentate nucleus | maroon | letter |

Table 2: The soma type conﬁguration data used if no conﬁguration ﬁle is available.

(The granule and rosette tip somas are shown as dots, not as letters, because they are the most numerous and drawing them as letters slows performance.)

### Selection Tab

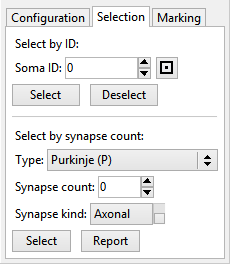


Figure 12: The selection tab from the sidebar.

From the selection tab, somas can be selected or deselected by ID. Somas can also be selected according to whether they have a certain count of axonal or dendritic synapses. Since this may ﬁnd more somas than can be selected, the somas can also be reported in a text ﬁle.

### Marking Tab

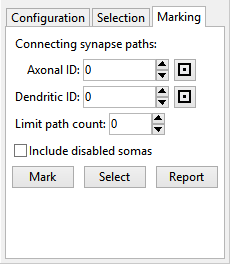


Figure 13: The marking tab from the sidebar.

From the marking tab, synapses can be marked along connecting paths. By choosing an axonal and dendritic soma to start and end the paths, the intermediate synapses can be marked or reported in a text ﬁle, or the intermediate somas can be selected.

## Model Area

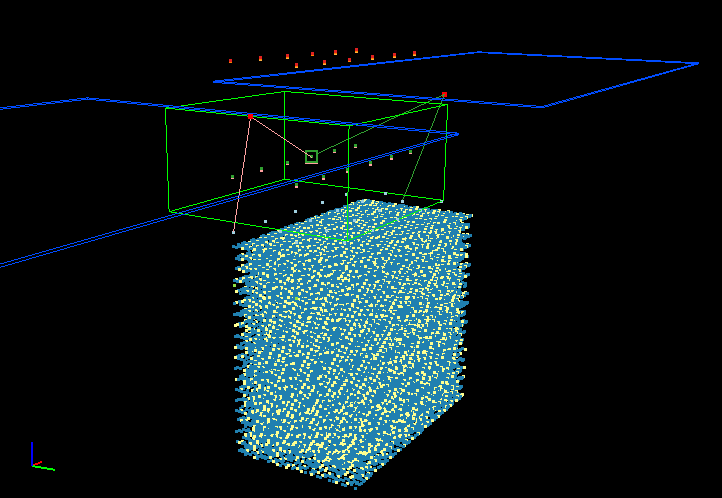


Figure 14: The model area, showing the upper portion of a model emulating a slice of cerebellar cortex. Most of the visible somas are granule cells and mossy ﬁber rosette tips. Two basket cells are selected; their synapses and neuritic ﬁelds are visible as lines and boxes, respectively.

The model area displays the cerebellar model from an opened ﬁle. It uses a perspective projection by default. Each soma is represented as a dot or letter in 3D space, colored according to its type. Selected somas are larger dots with a surrounding circle. They may also have their synaptic connections and ﬁelds displayed. Synapses are represented as straight lines connecting somas, colored according to the type of the axonal soma. Neuritic ﬁelds are represented as rectangular prisms: blue for axonal ones and green for dendritic ones, with the contained axonal synapses as red dots and dendritic synapses as purple dots. Marked synapses are represented as orange concentric circles.

In the bottom-left corner is an axis guide corresponding to the model’s orientation. The *x*-axis is red, the *y*-axis green, and the *z*-axis blue. Viz uses a right-handed coordinate system by default.

In the top-left corner is a bulletin listing the selected somas (hidden by default). When the model area is displaying simulation data, the bulletin also shows the current cycle.

In the top-right corner is an estimate of the frames per second that can be rendered (hidden by default). Larger models take longer to render individual frames.

In the bottom-right corner, when the model area is displaying simulation data, is a color scale showing how soma colors correspond to the quantities being simulated over time.

## Status Bar



Figure 15: The left half of the status bar, with three selected somas.

The status bar displays counts of the somas and synapses in the entire open model, as well as counts of the selected somas and marked synapses. The buttons next to the synapse count, marked count, and selected count generate text ﬁle reports of their respective items. When multiple somas are selected, the one being described can be chosen with the arrow buttons next to the selected count, or deselected with the X button. The pin button sets the selected soma as the pivot point for rotations. (If no somas are selected, it will reset the pivot point to the center of mass of the clipping region.) The question mark button brings up a summary dialog with information about the selected soma. (If no somas are selected, it will bring up information about the entire model.)



Figure 16: The right half of the status bar, describing a selected soma.

For a selected soma, the status bar displays its type, ID, and coordinates. If the soma has recorded ﬁring spikes, its current ﬁring frequency is displayed. If it has recorded voltages, its current voltage is displayed. Coordinates are measured in micrometers (µm) from the origin.

## Simulation Bar

The simulation bar appears only when a ﬁring spike ﬁle has been loaded. It allows the user to choose a simulation cycle time instance at which to view the model. If voltage or weight ﬁles have been loaded, they can also be chosen on the simulation bar.

After choosing an initial time instance, the user can press the Play button to animate the ﬁring pattern. The animation speed can range from 1 (1 cycle per second) to 10 (100 cycles per second). The user can also press or hold down the Step button to manually advance the time instance.

### Static Model

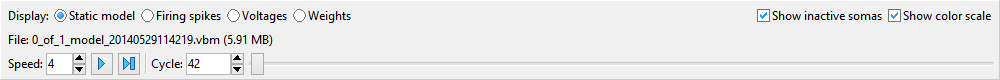


Figure 17: The simulation bar, while displaying the static model.

When displaying the static model, simulation bar shows the model’s size along the three axes. The model area has the same display as if no simulation data were loaded.

### Firing Spikes

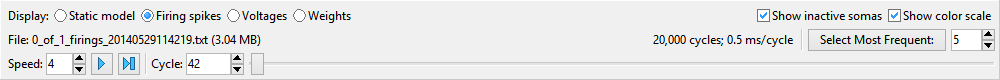


Figure 18: The simulation bar, while displaying ﬁring spikes.

When displaying ﬁring spikes, the simulation bar shows the number of cycles and the milliseconds per cycle, and has controls to generate text ﬁle reports of the current or average ﬁring frequencies of the enabled somas, and to select the most frequently ﬁring somas. (The reports exclude disabled somas, and if any somas are selected, only the selected somas will be included in the reports.) If “Display value for somas” is enabled, somas which would otherwise be displayed as letters will be displayed as their frequency values in Hertz.

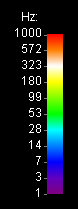


Figure 19: The color scale for ﬁring spikes. The logarithm of the frequency determines the hue.

For the chosen cycle time, ﬁring somas are displayed as large colored dots, and inactive ones are small gray dots. The inactive somas can be hidden so as not to obscure the ﬁring ones. The ﬁring somas’ colors correspond to their frequency, estimated from the past 1,000 time instances. (If there are not enough past instances, at least 5 are assumed to exist so that the initial estimates are not too high.) As time passes without a soma ﬁring again, its color fades to gray.

### Voltages

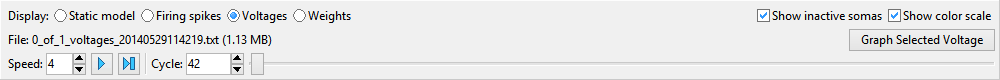


Figure 20: The simulation bar, while displaying voltages.

When displaying voltages, the simulation bar shows the number of somas with recorded voltages, and has a button to graph the selected soma’s voltage. If “Display value for somas” is enabled, somas which would otherwise be displayed as letters will be displayed as their voltage values in millivolts.

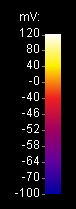


Figure 21: The color scale for voltages. More hues are used for voltages from −70 to −40 mV.

The voltage graph appears in a new window. Red dots indicate natural ﬁring spikes, magenta dots indicate forced ﬁrings, orange dots indicate binary ﬁrings, and green dots indicate suppressions. Clicking anywhere on the graph shows a tooltip with the cycle time instance and voltage at that point. Scrolling up and down with the mouse wheel, or pressing or holding the zoom buttons, zooms in and out of the graph’s *x*-axis. The range of the *y*-axis can also be adjusted. The “Export Image” button exports the entire graph as an image. The “Detect Peaks” button generates a text ﬁle report of peaks in the graph, and marks them with blue dots. Multiple graph windows can be opened at once.

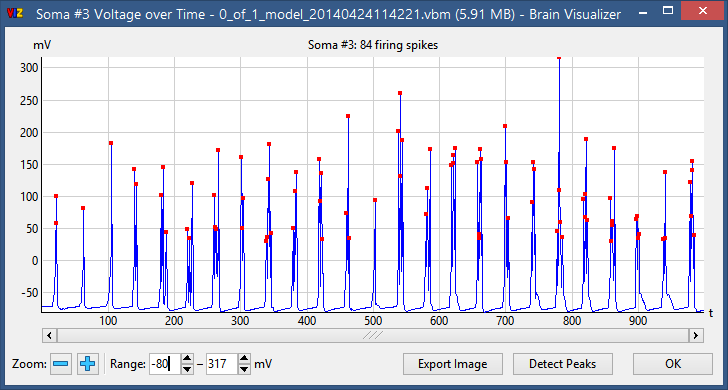


Figure 22: The voltage graph, showing the voltage of a Purkinje cell.

### Weights

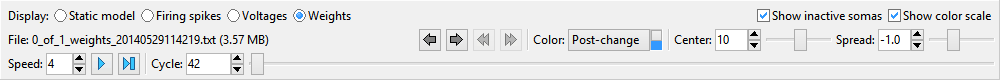


Figure 23: The simulation bar, while displaying weights.

When displaying weights, the simulation bar has buttons to skip to the previous or next time instance at which any weights changed, or at which a weight was changed involving the currently selected soma. It also has controls to toggle the color scale to correspond to pre-change or post-change weights, and to adjust the scale’s center and spread. (Positive spreads are more linear, negative ones are more logarithmic.) The center and spread values are initially set to maximize the number of colors used, given the range of weights present in the ﬁle.

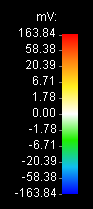


Figure 24: The color scale for weights. Cool colors represent negative weights, warm colors represent positive ones.

If prunings data is loaded after weights data, the weight changes due to prunings are added to the existing weight changes.

# Performance

Viz has been tested on a variety of systems, from a laptop manufactured in 2008 with a dual-core 32-bit processor and 4 GB of RAM, to a desktop with a quad-core 64-bit Intel Xeon processor and 64 GB of RAM. The latter system was used to gather the data below, since testing very large models requires a correspondingly large amount of memory.

## Loading Times

The 53.3 GB ﬁle viz\_input\_X1200xY6000​\_​2013​10​09​202848.txt, with 4.37 million somas and nearly 1.1 billion synapses, takes 4 minutes and 33 seconds to load from a solid-state hard drive. A more reasonably sized ﬁle, the 2.0 GB viz\_input\_X720xY1000​\_​2013​10​09​012923.txt, with 437 thousand somas and 46.6 million synapses, takes 11 seconds to load. Almost all the time is spent loading the synapses; overall Viz loads over 4 million synapses per second.

## Memory Usage

Viz takes up approximately 10 to 12 MB of memory when it ﬁrst starts, before opening any ﬁles. The 64-bit builds use slightly more than their 32-bit counterparts, but are capable of opening larger ﬁles since they can address more RAM. Builds with ﬂoating-point (32-bit) coordinates need around 80% as much memory compared to the input ﬁle size; those with short (16-bit) coordinates need only 55%.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **File Name** | **File Size (MB)** | **Somas** | **Synapses** | **Memory Usage (MB)** | | | |
| **x86 SC** | **x86 FP** | **x64 SC** | **x64 FP** |
| None | 0 | 0 | 0 | 10.55 | 10.56 | 12.43 | 11.80 |
| viz\_input\_X120xY100 \_20131009011601.txt | 5.36 | 7,440 | 163,552 | 22.25 | 24.21 | 25.29 | 26.47 |
| viz\_input\_X360xY100 \_20131009011613.txt | 15.30 | 22,167 | 414,113 | 30.05 | 35.38 | 33.35 | 37.47 |
| viz\_input\_X360xY200 \_20131009011627.txt | 45.13 | 44,459 | 1,149,520 | 51.47 | 65.65 | 54.80 | 68.01 |
| viz\_input\_X360xY300 \_20131009011643.txt | 91.43 | 65,624 | 2,274,627 | 82.70 | 110.54 | 87.38 | 113.30 |
| viz\_input\_X360xY400 \_20131009011705.txt | 155.38 | 87,612 | 3,813,107 | 125.84 | 171.23 | 130.76 | 174.60 |
| viz\_input\_X360xY500 \_20131009011732.txt | 234.29 | 109,829 | 5,705,086 | 178.05 | 246.22 | 183.22 | 249.39 |
| viz\_input\_X720xY500 \_20131009011813.txt | 542.56 | 220,112 | 12,679,186 | 373.14 | 523.48 | 380.93 | 527.11 |
| viz\_input\_X720xY600 \_20131009011929.txt | 760.69 | 262,164 | 17,561,094 | 507.04 | 714.45 | 516.03 | 718.35 |
| viz\_input\_X720xY700 \_20131009012113.txt | 1,019.61 | 306,501 | 23,321,502 | 664.37 | 939.00 | 674.32 | 943.61 |
| viz\_input\_X720xY800 \_20131009012324.txt | 1,333.09 | 350,792 | 30,314,184 | 855.09 | 1,210.49 | 865.78 | 1,215.71 |
| viz\_input\_X720xY900 \_20131009012607.txt | 1,688.84 | 392,990 | 38,195,074 | 1,069.52 | — | 1,081.22 | 1,521.36 |
| viz\_input\_X720xY1000 \_20131009012923.txt | 2,067.26 | 437,295 | 46,586,740 | 1,297.10 | — | 1,309.73 | 1,846.59 |
| viz\_input\_X1200xY1000 \_20131009013330.txt | 3,339.37 | 729,176 | 74,210,394 | — | — | 2,077.88 | 2,933.60 |
| viz\_input\_X1200xY2000 \_20131009014104.txt | 12,685.56 | 1,457,531 | 266,043,045 | — | — | 7,282.41 | 10,343.92 |
| viz\_input\_X1200xY3000 \_20131009021032.txt | 22,946.90 | 2,182,778 | 472,403,983 | — | — | 12,876.12 | 18,309.40 |
| viz\_input\_X1200xY4000 \_20131009024959.txt | 33,420.25 | 2,912,478 | 680,615,814 | — | — | 18,519.34 | 26,345.46 |
| viz\_input\_X1200xY5000 \_20131009190347.txt | 44,017.51 | 3,641,476 | 890,376,475 | — | — | 24,204.03 | 34,441.41 |
| viz\_input\_X1200xY6000 \_20131009202848.txt | 54,548.08 | 4,366,322 | 1,097,004,014 | — | — | 29,804.36 | 42,417.13 |

Table 3: Comparing memory usage for diﬀerent input ﬁles and Viz builds, built with Visual Studio 2012 on Windows 7. All measurements are averages of three runs. Memory measurements marked FP use ﬂoating-point coordinates, while those marked SC use short coordinates. Empty cells for the 32-bit builds indicate that the ﬁles needed more memory than a 32-bit process can allocate.

Figure 52: A chart of the ratio of memory usage to input ﬁle size for some of the input ﬁles in the above table. The very small ﬁles have exceptionally high ratios and are not present on the chart. For very large ﬁles, the ﬂoating-point coordinate (FP) builds level oﬀ at around 80%, and the short coordinate (SC) builds at 55%.

The most signiﬁcant factor for memory usage is the number of synapses in a model, followed by the number of somas (which is one or two orders of magnitude smaller). The data structures for storing synapses and somas have been optimized to be as small as possible. To avoid 64-bit builds using extra memory, Viz often uses 32-bit array oﬀsets instead of 64-bit pointers.

## Frame Rate

For input ﬁles with up to around a million somas or half a billion synapses, the frame rate when interacting with models is smooth. For the ﬁles with closer to a billion synapses, actions like rotating and panning are choppier, but still responsive. Features that iterate over all the synapses, like when displaying an information dialog about the model, take a few seconds to complete. Interactive actions can be sped up by hiding some somas, either by disabling their types or by zooming in or clipping to a particular section.

1. Screenshots

These screenshots illustrate particular visualization options, as well as how to use the tools for interacting with the model area and conﬁguring the model.

* 1. Synaptic Connections

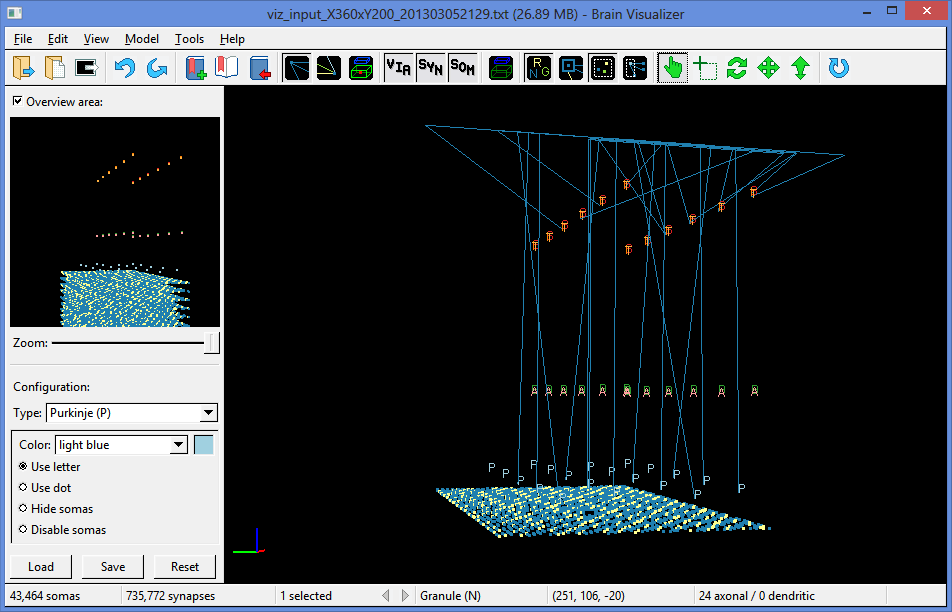


Figure 25: With the To Via, To Synapse, and To Other Soma options on, axonal connections are drawn from the selected granule cell, through its single *via* point and many synapses (located on its parallel ﬁber), to the dendritic somas (of types P, S, T, and I). The model has been clipped to show only the top part of a cerebellar cortex model.

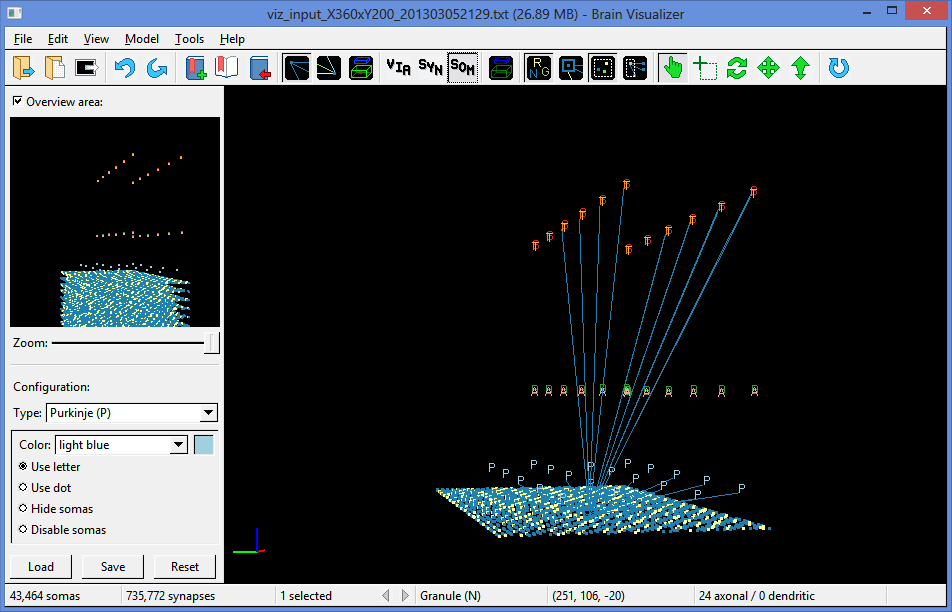


Figure 26: With only the To Other Soma option on, axonal connections are drawn directly from the selected granule cell to its dendritic somas (P, S, T, and I).

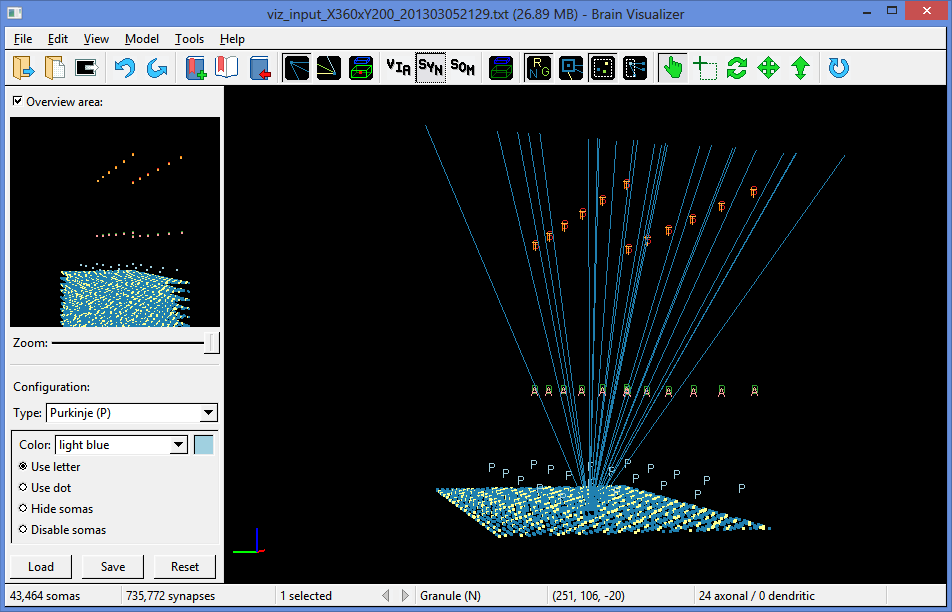


Figure 27: With only the To Synapse option on, axonal connections are drawn directly from the selected granule cell to its axonal synapses (all on the same parallel ﬁber).

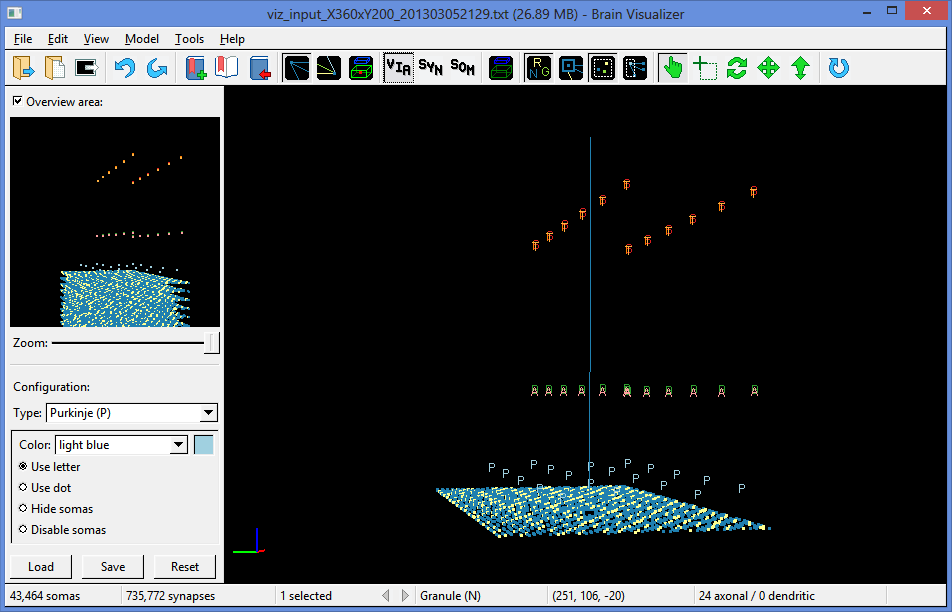


Figure 28: With only the To Via option on, axonal connections are drawn directly from the selected granule cell to its *via* point, where its axon bifurcates to form a parallel ﬁber (not shown).

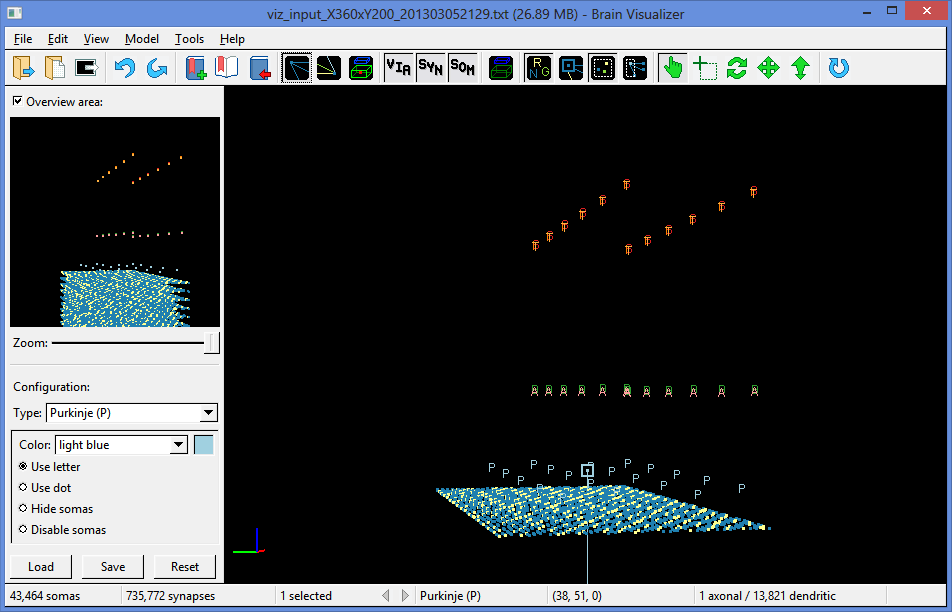


Figure 29: With the Axonal Connections option on and the Dendritic Connections option oﬀ, only the axonal connection from the selected Purkinje cell is drawn.

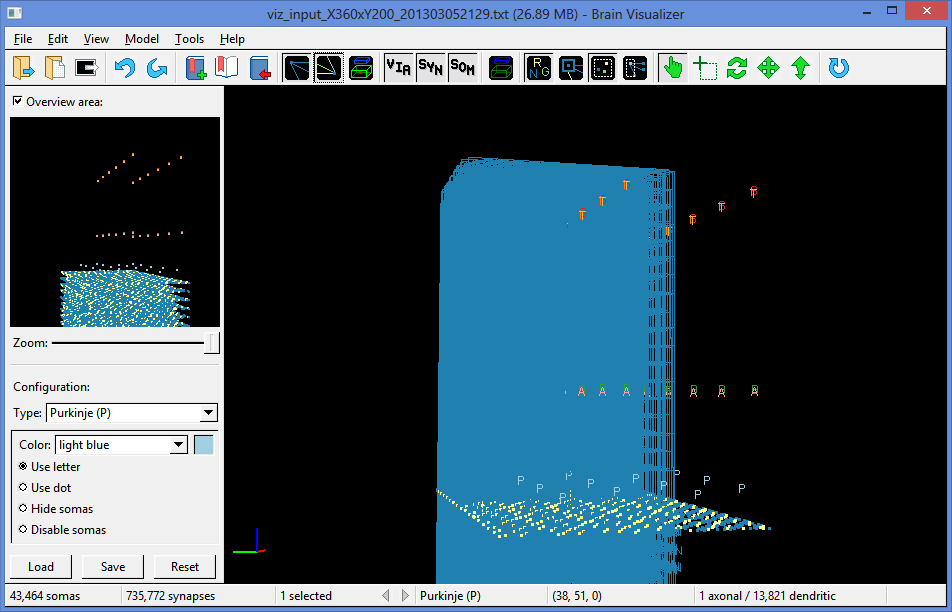


Figure 30: With both the Axonal Connections and the Dendritic Connections options on, the myriad axonal and dendritic connections of the selected Purkinje cell are drawn.

* 1. Neuritic Fields

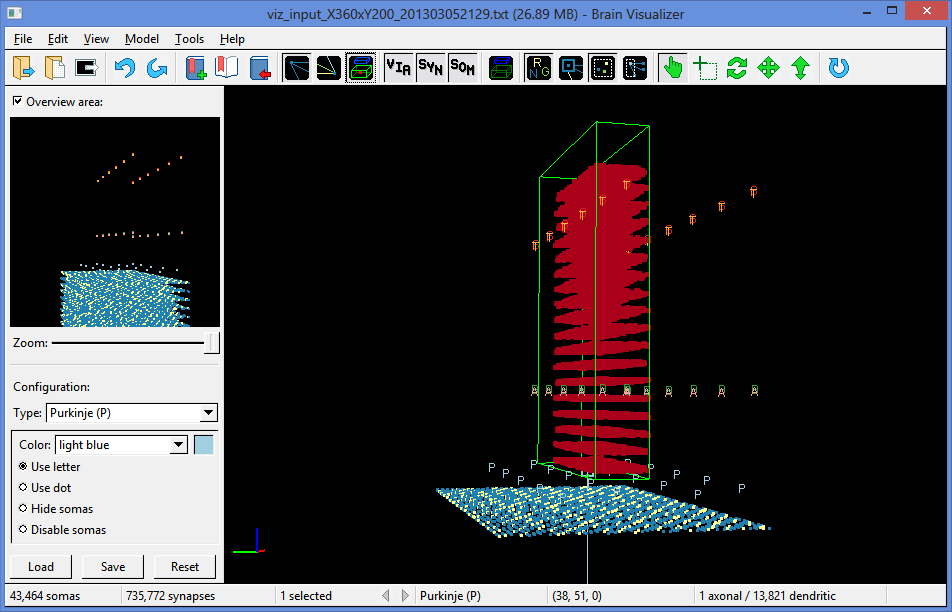


Figure 31: With the Neuritic Fields option on, the neuritic ﬁelds and synapse locations of the selected Purkinje cell are drawn. The axonal ﬁeld and synapses are outside the visible area, below the region being shown.

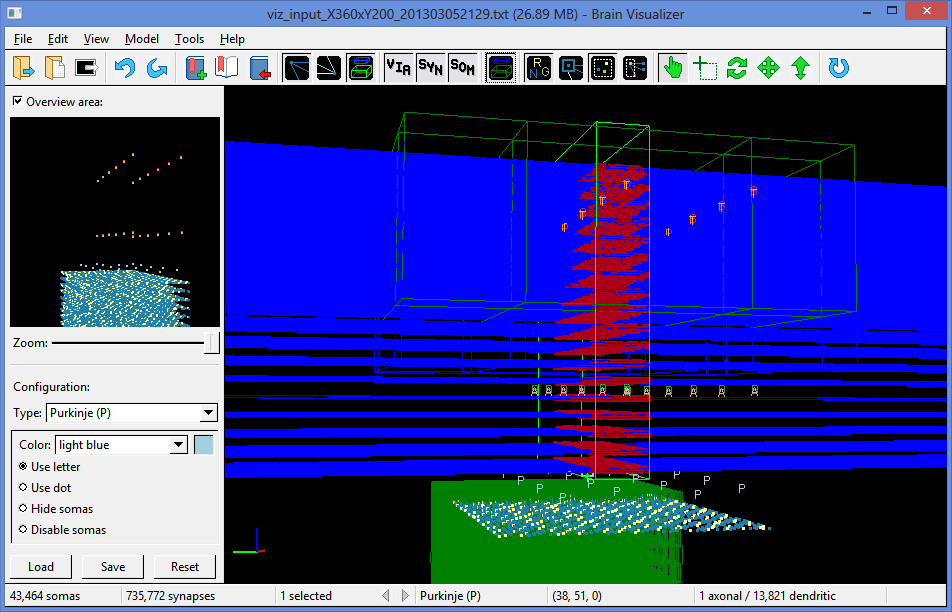


Figure 32: With the Connected Fields option on, the neuritic ﬁelds of the somas connected to the selected Purkinje cell are drawn. The blue axonal ﬁelds are parallel ﬁbers, and the green dendritic ﬁelds belong to granule cells making synapses (the red dots) within the Purkinje cell’s large dendritic tree.

* 1. Selection

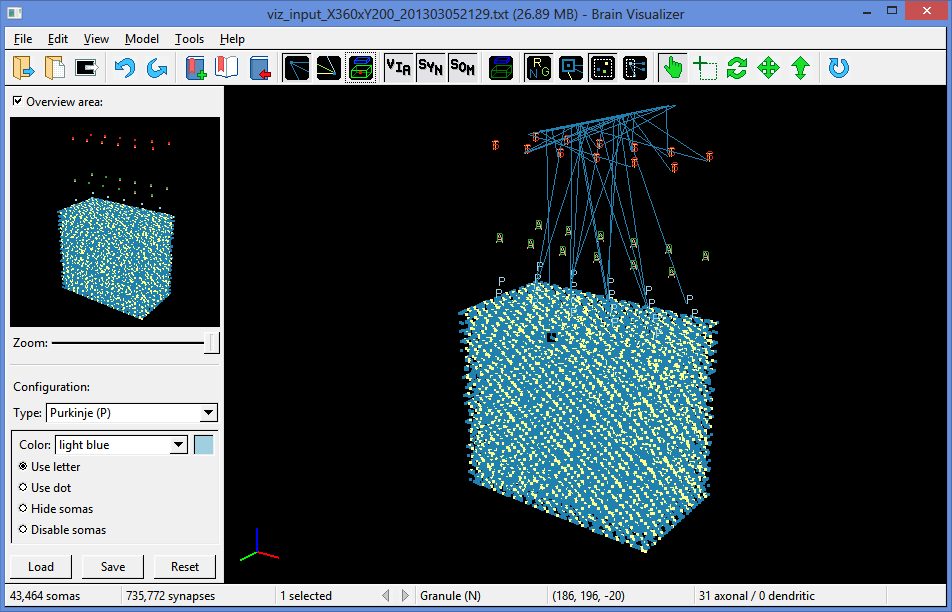


Figure 33: One selected granule cell.

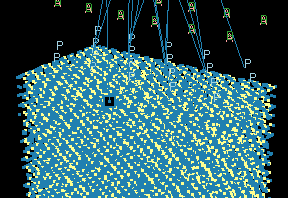


Figure 34: A close-up of the selected granule cell in the previous ﬁgure.

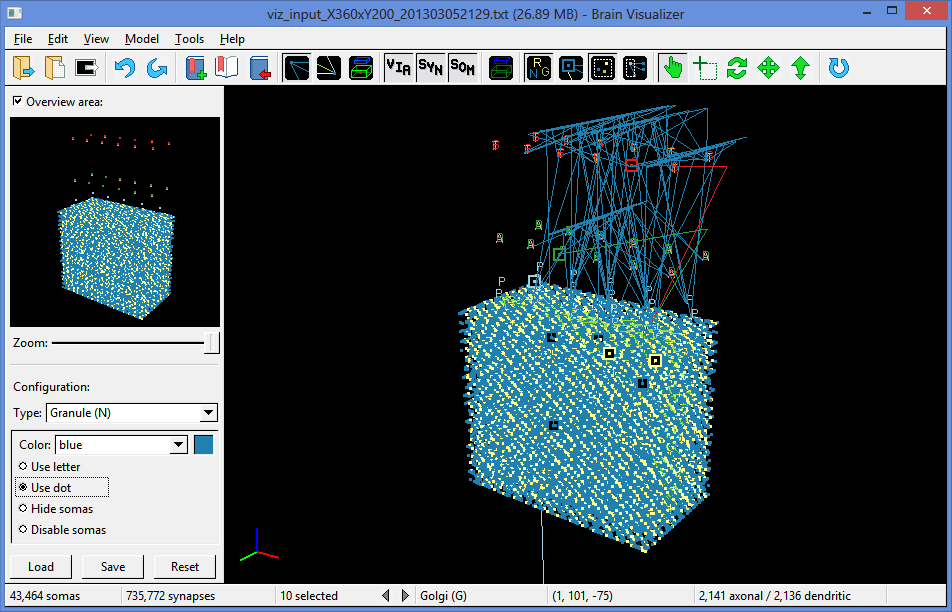


Figure 35: Ten selected somas: four granule cells, two mossy ﬁbers’ rosette tips, a Purkinje cell, a basket cell, an outer stellate cell, and a Golgi cell.

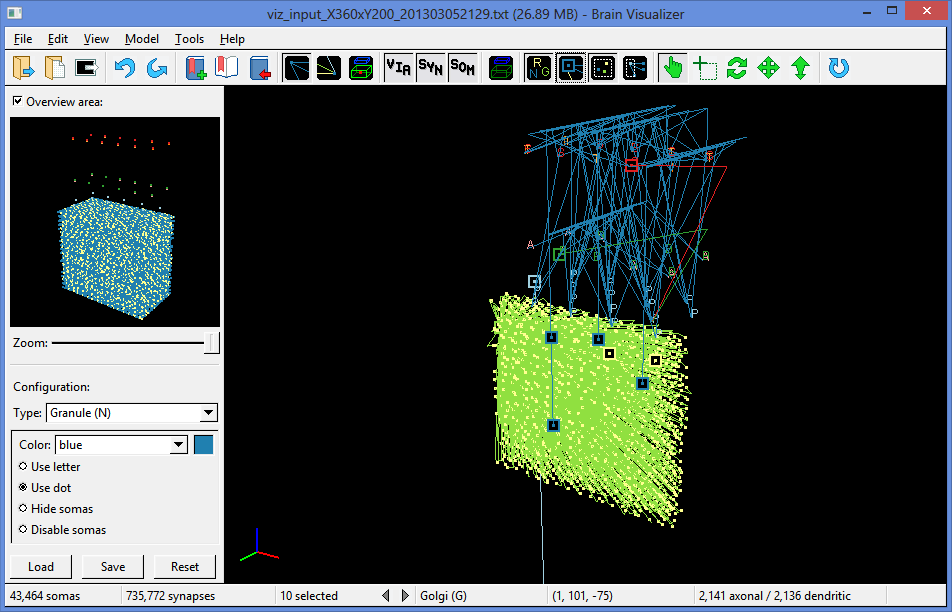


Figure 36: The same ten selected somas as above, including their axonal connections, with the Only Show Selected option on.

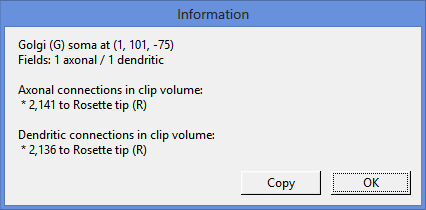


Figure 37: The information dialog brought up by Ctrl+clicking or right-clicking on a Golgi cell.

* 1. Clipping

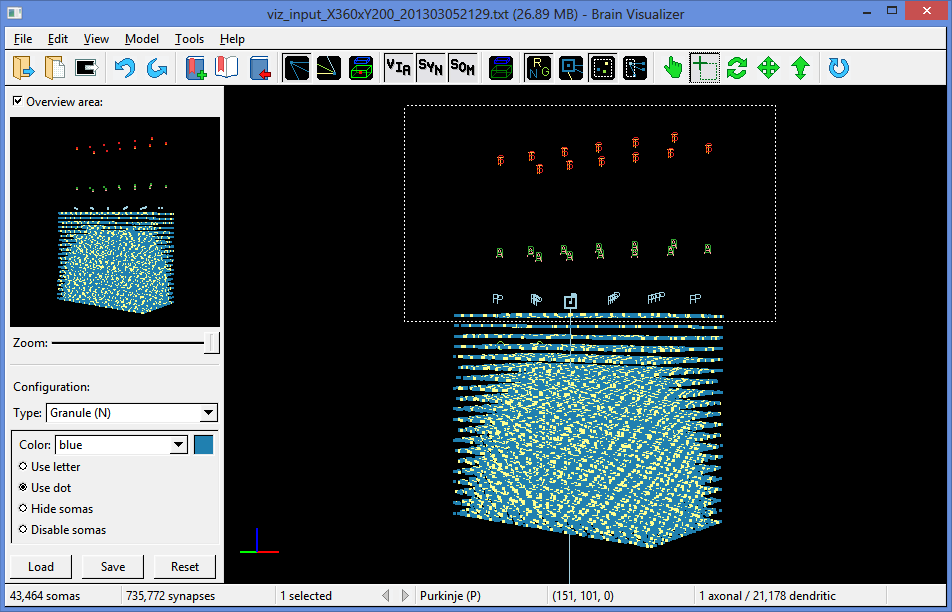


Figure 38: A clipping rectangle being drawn with the Clip tool. A single Purkinje cell is selected and the top of its long descending axon is shown.

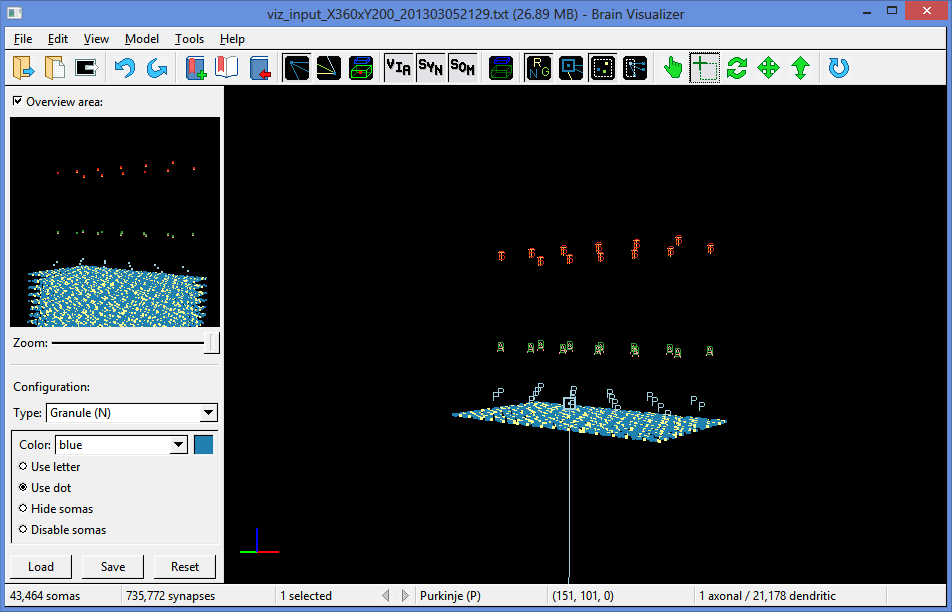


Figure 39: The result of the Clip action.

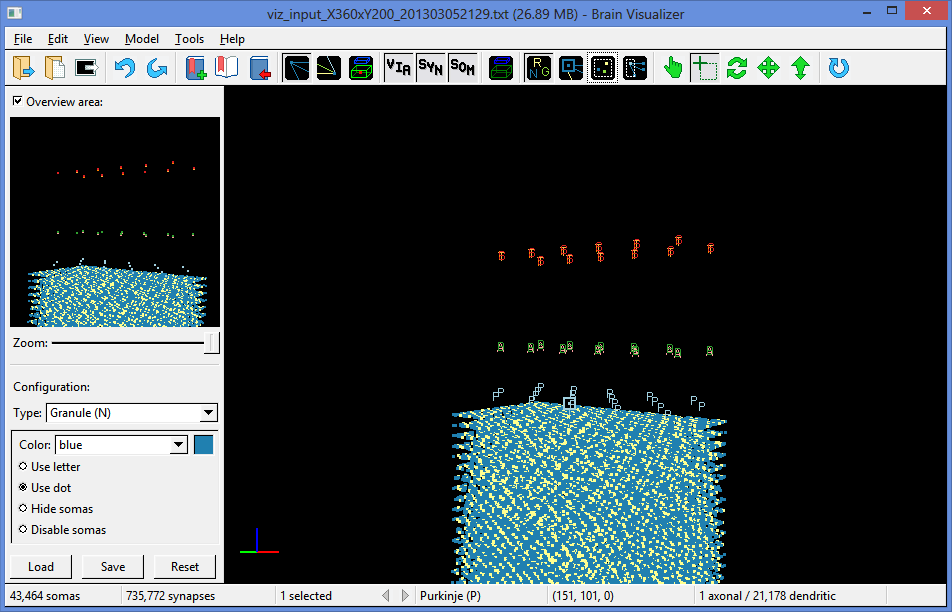


Figure 40: A clipped region with the Hide Clipped option oﬀ shows somas outside the clipped region.

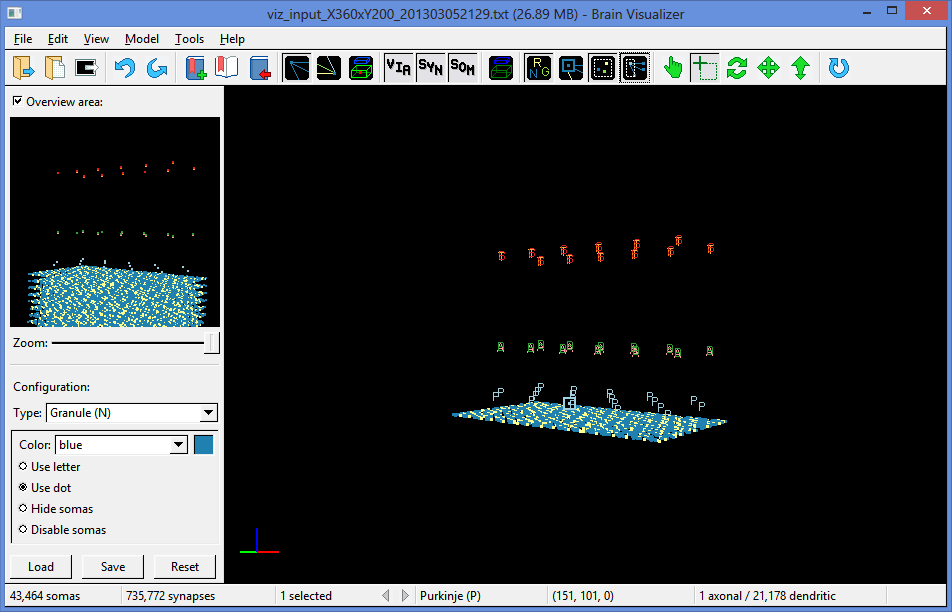


Figure 41: A clipped region with the Only Enable Clipped option on; the selected Purkinje cell’s connection to a dentate nucleus below the cortex and outside the clipped region is no longer shown.

* 1. Rotation

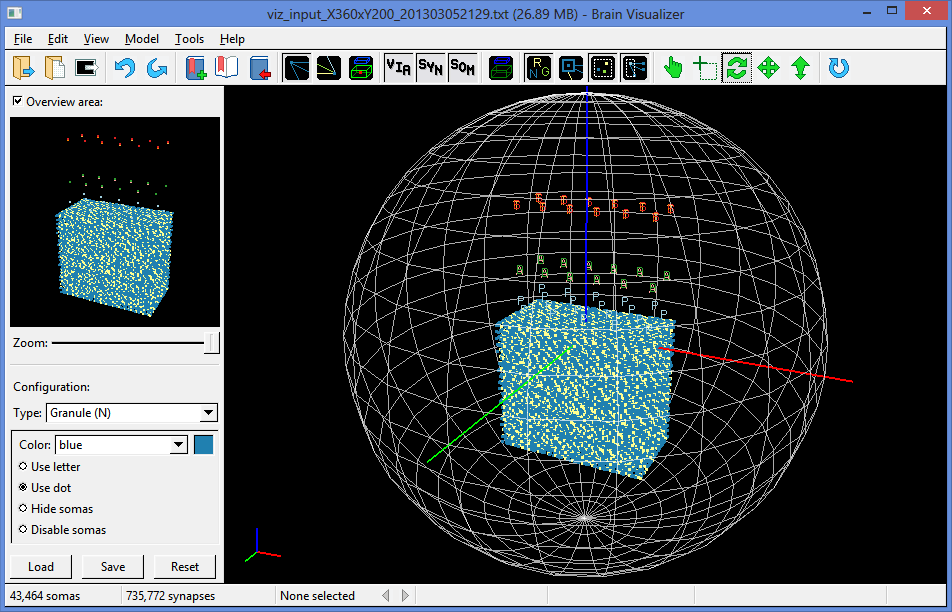


Figure 42: Using the rotation guide to rotate a model.

* 1. Conﬁguration

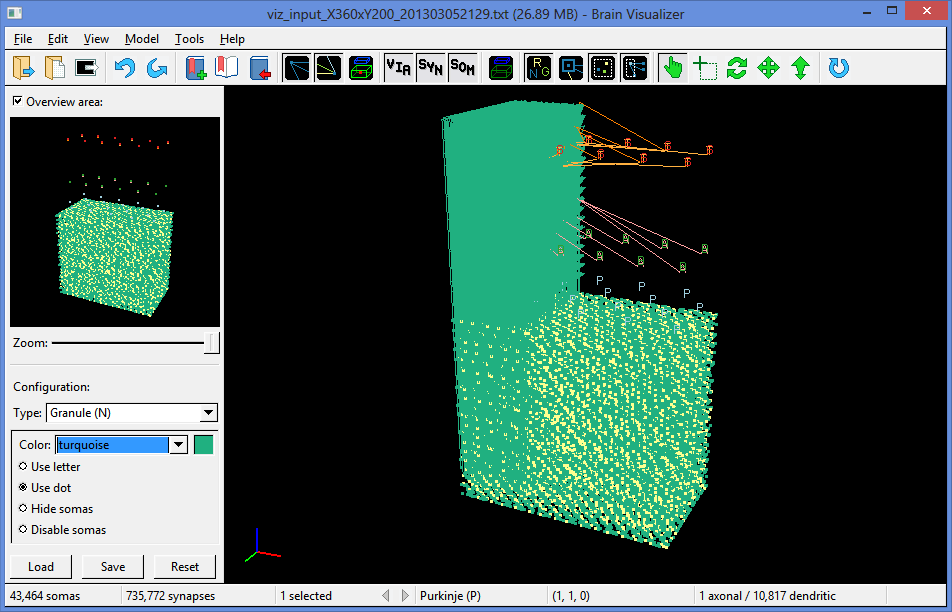


Figure 43: A model with the granule type set to a diﬀerent color. Granule cells and their axonal connections to a single selected Purkinje cell are shown in turquoise instead of blue.

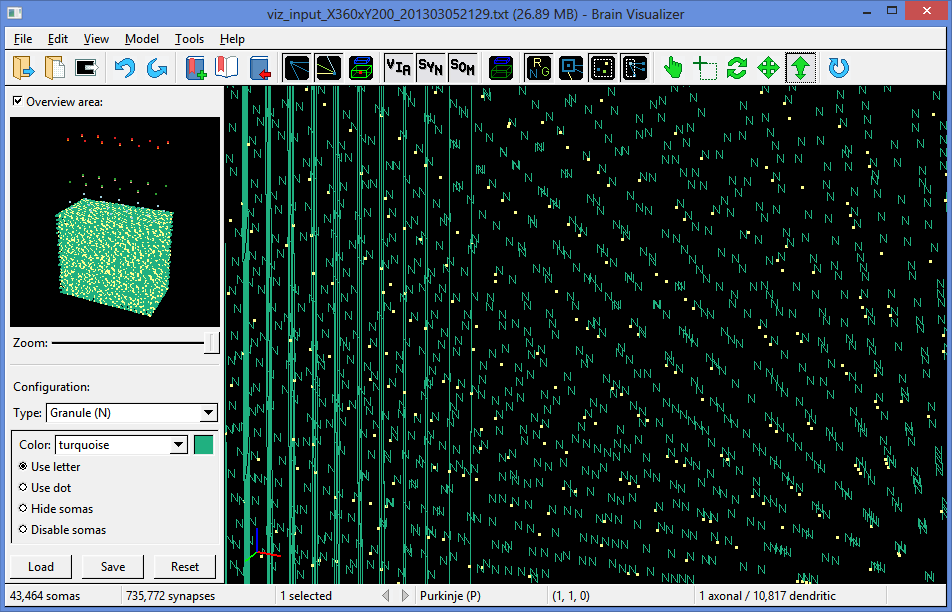


Figure 44: A zoomed-in close-up of a model with the granule cells displayed as letters. Some of the granule cells’ axonal connections to a Purkinje cell are in the visible area.

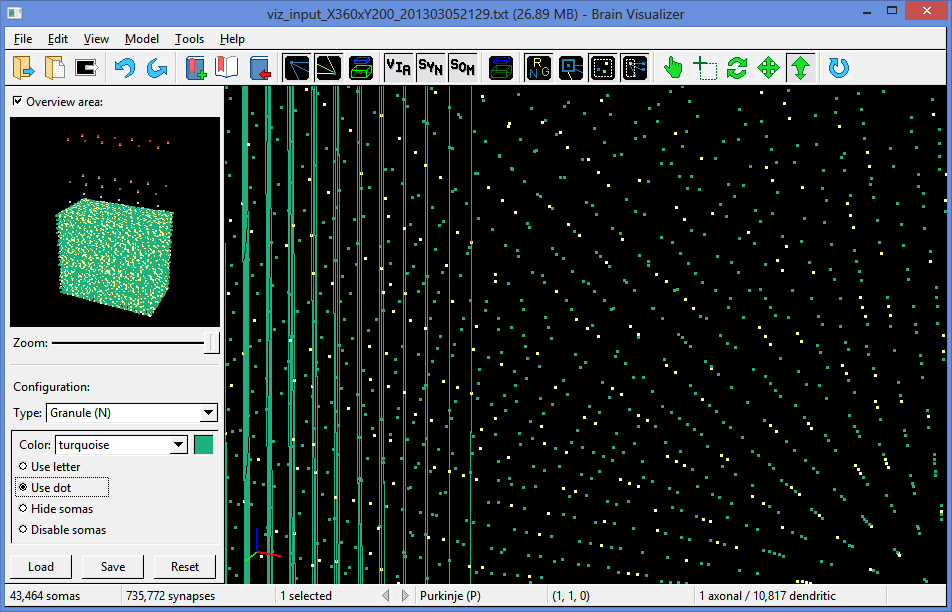


Figure 45: The same model as the previous ﬁgure, but with the granule cells displayed as dots.

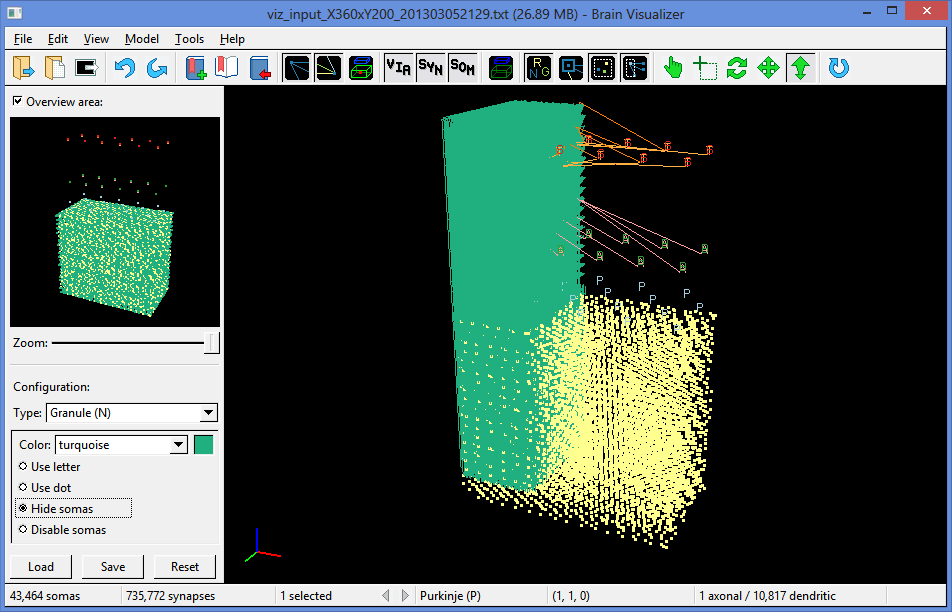


Figure 46: The same model with the granule type hidden. Only the granule cells connected to a selected Purkinje cell, and their axonal connections to that Purkinje cell, are still shown. (The yellow dots are mossy ﬁber rosette tips.)

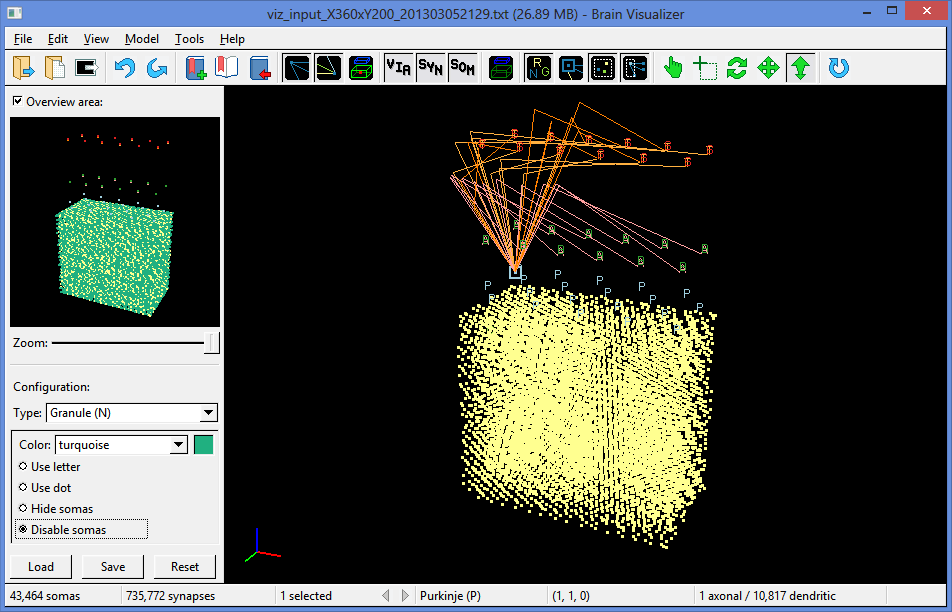


Figure 47: The same model with the granule type disabled. Granule somas and axonal connections are no longer shown.

* 1. Firing Data

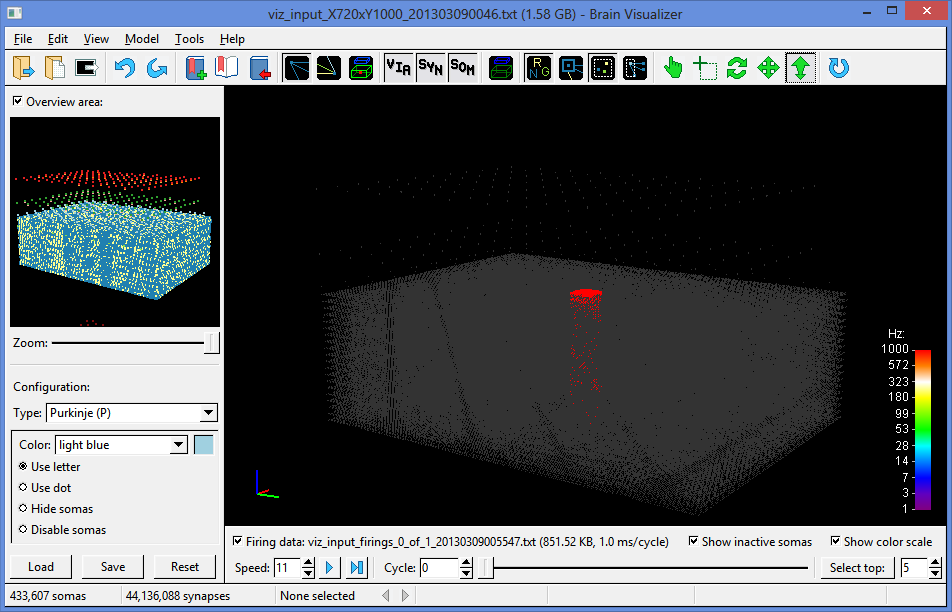


Figure 48: A larger cerebellar cortex model (containing 433 thousand somas and 44 million synapses) with ﬁring data loaded. All the granule cells within a small central column are stimulated to ﬁre once every 20 cycles (which is every 10 milliseconds).

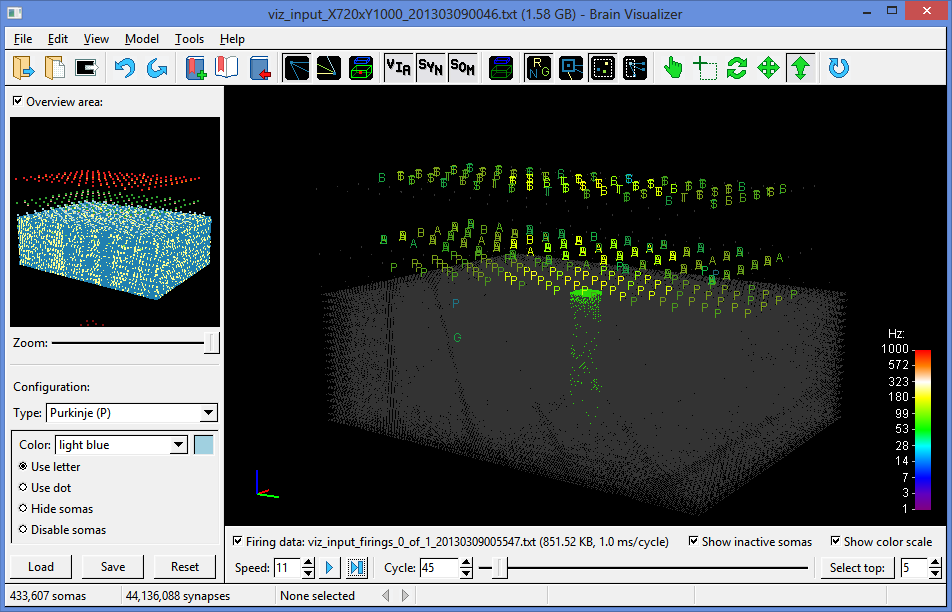


Figure 49: The ﬁring data of the model from the previous ﬁgure after simulating 45 cycles (22.5 ms) of electrical interaction.

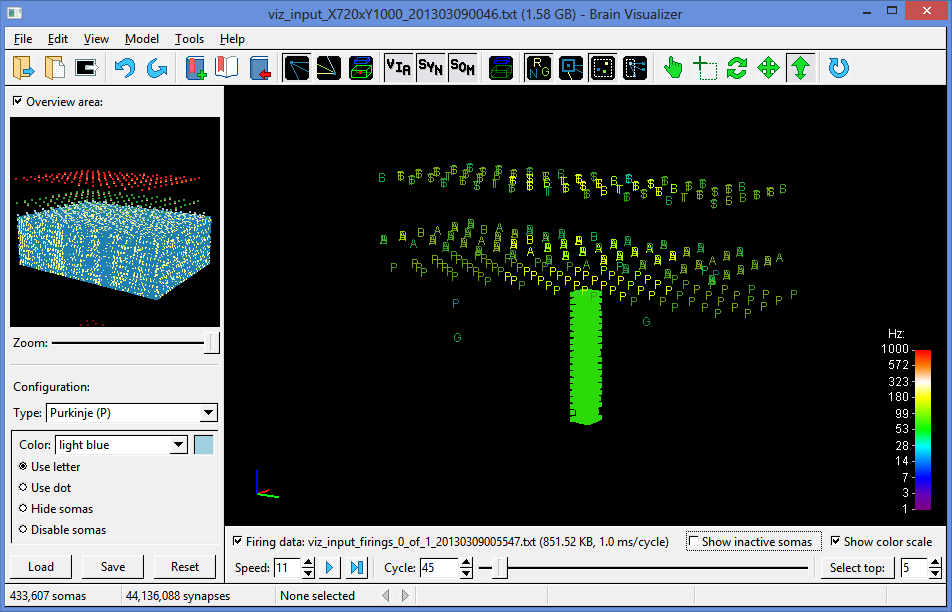


Figure 50: The same ﬁring data as the previous ﬁgure, with inactive somas hidden.

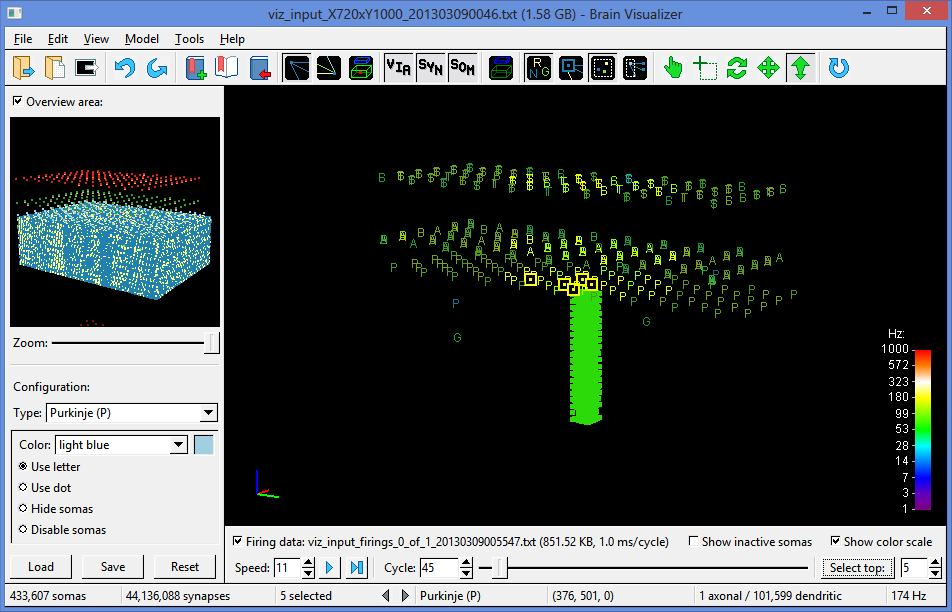


Figure 51: The ﬁring data with the top 5 most frequently ﬁring somas (all Purkinje cells) selected.

1. Source Code Files

These ﬁles are the contents of the src directory of the source code for Viz. Unlisted ﬁles include IDE project ﬁles and Makeﬁles, documentation, conﬁguration ﬁles, FLTK library and header ﬁles, and resource ﬁles such as icons.

* main.cpp: Deﬁnes the main() function, the entry point into the Viz program that initializes FLTK, creates a new window and opens a model ﬁle if passed one via command-line arguments.
* version.h: Deﬁnes VIZ\_VERSION and VIZ\_VERSION\_STRING, two constants that specify the current version of Viz. They are automatically updated by the build script for deploying a production copy of Viz.
* utils.h: Deﬁnes some useful low-level macros, types, and constants.
* algebra.cpp and algebra.h: Deﬁne mathematical constants like PI, and functions for vector and matrix algebra.
* from-ﬁle.cpp and from-ﬁle.h: Deﬁne the abstract From\_File class for containing data read from a ﬁle. The class only stores the ﬁle’s name and size; it can be extended to store more particular data.
* input-parser.cpp and input-parser.h: Deﬁne the Input\_Parser class for parsing text ﬁles and the Gzip\_Input\_Parser class for parsing Gzipped text ﬁles. The text is expected to contain space-separated numeric tokens, with “#” characters commenting out all text until the end of the line.
* binary-parser.cpp and binary-parser.h: Deﬁne the Binary\_Parser class for parsing binary ﬁles and the Gzip\_Binary\_Parser class for parsing Gzipped binary ﬁles. The binary ﬁle format is meant to store brain models.
* image.cpp and image.h: Deﬁne the Image class for writing RGB image data to a ﬁle in PNG, BMP, TGA, or PPM format.
* soma-type.cpp and soma-type.h: Deﬁne the Soma\_Type class for storing a soma type’s letter, name, color, and display state.
* soma.cpp and soma.h: Deﬁne the Soma class for storing a soma’s coordinates and associating it with a set of neuritic ﬁelds and synapses.
* neuritic-ﬁeld.cpp and neuritic-ﬁeld.h: Deﬁne the Neuritic\_Field class for storing a neuritic ﬁeld’s bounding box.
* synapse.cpp and synapse.h: Deﬁne the Synapse class for storing a synapse’s coordinates and via coordinates, and associating it with an axonal and dendritic soma.
* gap-junction.cpp and gap-junction.h: Deﬁne the Gap\_Junction class for storing a gap junction’s coordinates and associating it with a pair of somas.
* brain-model.cpp and brain-model.h: Deﬁne the Brain\_Model class for storing a brain model’s arrays of soma types, somas, neuritic ﬁelds, synapses, and gap junctions, as well as associated simulation data.
* sim-data.cpp and sim-data.h: Deﬁne the abstract Sim\_Data class for storing data associated with a simulation run. The class only keeps track of a current cycle and associated brain model and color map; it can be extended to store more particular data.
* ﬁring-spikes.cpp and ﬁring-spikes.h: Deﬁne the Firing\_Spikes class for storing ﬁring spike data from a simulation run. Somas’ current ﬁring frequency can then be estimated from a window of their past spikes.
* voltages.cpp and voltages.h: Deﬁne the Voltages class for storing soma voltage data from a simulation run.
* weights.cpp and weights.h: Deﬁne the Weights class for storing synapse weight data from a simulation run.
* fps.cpp and fps.h: Deﬁne the FPS class for estimating frames per second from counting clock ticks over time.
* bounds.cpp and bounds.h: Deﬁne the Bounds class for storing a 3D coordinate boundary around a set of points, and calculating their center and range.
* clip-volume.cpp and clip-volume.h: Deﬁne the Clip\_Volume class for storing four clipping planes and enabling or disabling them in OpenGL.
* color.cpp and color.h: Deﬁne the Color class for storing a named RGB color.
* color-maps.cpp and color-maps.h: Deﬁne the abstract Color\_Map class for mapping numeric values to colors, and subclasses that specify particular gradients to map onto, including Rainbow\_Map, Thermal\_Map, and Opposed\_Map.
* draw-options.cpp and draw-options.h: Deﬁne the Draw\_Options class for storing the drawing state of a model area.
* model-state.cpp and model-state.h: Deﬁne the Model\_State class for storing the state of a model being visualized, including its rotation, panning, zoom, clip volume, selected somas, and marked synapses.
* model-area.cpp and model-area.h: Deﬁne the Model\_Area class for displaying a brain model and maintaining an undo/redo history of its state changes.
* overview-area.cpp and overview-area.h: Deﬁne the Overview\_Area class for displaying an overview of a brain model.
* widgets.cpp and widgets.h: Deﬁne various classes for common GUI widgets, including buttons, sliders, and spinners.
* os-themes.cpp and os-themes.h: Deﬁne appearances for GUI widgets which imitate the native widgets in Windows 7, Windows 8, Mac OS X Lion, and Linux (assuming a GTK desktop environment).
* modal-dialog.cpp and modal-dialog.h: Deﬁne the Modal\_Dialog class for presenting a modal dialog with a message, OK button, and optional icon.
* progress-dialog.cpp and progress-dialog.h: Deﬁne the Progress\_Dialog class for presenting a progress dialog with a message, progress bar, and Cancel button. On Windows the progress state is reﬂected in the taskbar button.
* waiting-dialog.cpp and waiting-dialog.h: Deﬁne the Waiting\_Dialog class for presenting an indeterminate progress dialog with a message, progress bar, and Cancel button. On Windows the progress state is reﬂected in the taskbar button.
* report-dialog.cpp and report-dialog.h: Deﬁne the Report\_Dialog class for presenting a dialog with options for generating a text report ﬁle.
* summary-dialog.cpp and summary-dialog.h: Deﬁne the Summary\_Dialog class for presenting a dialog summarizing a brain model, a single soma, or a single synapse.
* viz-icons.h: Deﬁnes many icons based on XPM ﬁles.
* viz-window.cpp and viz-window.h: Deﬁne the Viz\_Window class for presenting a complete windowed instance of Viz, and callback functions for many user actions.
* help-window.cpp and help-window.h: Deﬁne the Help\_Window class for presenting a window with help information loaded from an HTML ﬁle.
* voltage-graph.cpp and voltage-graph.h: Deﬁne the Voltage\_Graph class for presenting a graph of soma voltage over time, the Voltage\_Graph\_Tooltip class for presenting a particular (time, voltage) pair as a tooltip, and the Voltage\_Graph\_Scroll class for scrolling a voltage graph when zooming into a small region.
* voltage-graph-window.cpp and voltage-graph-window.h: Deﬁne the Voltage\_Graph\_Window class for presenting a voltage graph in a modal window.