3D visualisation of the mapping of a neural network model onto a neuromorphic hardware system

Projektpraktikum Informatik

Supervisor: Eric Müller, Daniel Brüderle

v2

1 Introduction

The Electronic Visions group [1] of the Kirchhoff Institut für Physik (KIP) is developing digital hardware with analog main components to simulate neural networks. This approach of using 'neuromorphic hardware', which is not just application-specific reconfigurable hardware for numerical simulations but rather has analog spiking neurons, which operate in parallel for themselves, allows for faster than real time simulation of neural activity, greatly facilitating experiments. As a first stage a chip ('Spikey') with 2 synapse arrays, each propagating to 192 artificial neurons, which together with their array of synaptic weights form a core and in turn are able to forward propagate to every other neuron of its core, was developed. However to reduce the complexity the connections between neurons of different cores are far more limited, a design principle similar to the kind of biological networks targeted, exploiting the sparsity of the connection matrix. This creates unfortunately the problem of allocation of the hardware neurons which correspond to those of the biological model a user of the system has, especially for the second stage of this system [2] which greatly increases the number of possible neurons, synapses and the complexity. This mapping process is done by algorithms, developed at the TU Dresden, that don't need action from the user who doesn't need intimate knowledge of the hardware and its limitations but rather can use the abstracting Python scripting language interface PyNN[1] [3] that is developed in conjunction with other groups of other universities within the scope of the FACETS Project [4] and allows to perform experiments interchangeably on neuromorphic hardware as well as, in principle completely limitation free in-software simulators like NEST, NEURON.

The algorithms use an abstract graph of the hardware and another one for the neural network, both subclassing a common highly generic `GraphModel`. To visualise[2] the mapping of neurons and synapses of a biological neural network model (which for biological simulations should be three-dimensional) to the hardware (which is effectively two-dimensional) in order to help the developer or user of the software environment in its verification, and the models themselves, the internship of Tobias Harion [5] resulted in a module to the mapping software for 3D display using OpenGL.

2 glVisu

'glVisu' is a set of C++ source files (GraphVisu.h / GraphVisu.cpp, glControl.h / glControl.cpp) which use the C API of OpenGL (the 1.X fixed-function pipeline) for 3D rendering of the graphs and GLUT (OpenGL Utility Toolkit) for platform integration like window management and user input with mouse and keyboard. It is is invoked by the function `AlgorithmController::StartVisualization()` from the C++-code that is managed by Boost.Python for the interoperability with Python.

Although more of a demonstrator it has a range of features. Neurons of the hardware are displayed on a square lattice as spheres, the ones used in a different colour as the others ones. The neurons of the bio model

[1] pronounced a like 'pine' tree

[2] As a compromise American English is used for the software, a German variant of British English for this report

are also little spheres and can be positioned on a cubic lattice, a square lattice, or on random positions on the inside of a sphere. They can be selected separately by clicking, by pressing of a key, or by aiming at them and pressing another key. For a selected bio neuron parameters from the graph are displayed on-screen, the outgoing synapses are displayed with different colours for excitatory and inhibitory impact and a mapping connection to hardware neurons, the only part of'stage 1' that is included, can be show. There is a context menu for the window with options, like what components to show or hide.

The user moves around in the 3D space in first-person navigation with mouse input for direction and keyboard for translation. There is also a full-screen mode which locks the mouse, giving standard game controls.

A console can be opened for input of a limited set of commands, the most important being the one to input a fille name for a simulation mode in which a spike train is loaded from the file and the spiking neurons are eye-catchingly coloured.

The scene can be rendered with an anaglyphic 3D effect for use with a set of filtering glasses.

The neurons and synapses of the bio graph are tranferred into `structs` for faster and easier rendering. In this process bio neurons are assigned numbers according to the order in which they appear in the graph.

Because GLUT is a C-API that executes user-provided callbacks, a global variable approach with wrappers is used to mix C callback functions and C++ member functions.

3 Course of the internship

The internship was done for the Softwarepraktikum credit in the semester break in February[3] and a good part of March of 2009 with only short breaks for a blocked lecture and a short excursion.

The most park of the work was done at the research group's office at the opening hours on a small laptop computer[4] which had to stay in place. There was an advertising gift like set of red-blue anaglyph glasses made of cardboard.

As the GraphModel classes depend on a whole range of in-house developed software and in particular the build system is building the complete shared library 'pyhal_c_interface_s1v2.so' the glVisu code could only be compiled in-house. For testing PyNN had to be run.

Ahead of the internship the papers [2] [3] were provided partially in pre-release version. Mostly in the first two weeks a reference book at the university library [6] was worked through off-time.

The group requires portable, and preferably open source software, and for that reason OpenGL with its wide spread support was chosen, with the benefit of its client / server model allowing using the software over ssh. The library GLUT, although more , had additionally the advantage of being cross-platform. In addition to that, an upper bound was set by the requirement of the soft-

[3]Coincidentally the moment 1234567890 in the UNIX epoch was almost spent working on this project.

[4]System: HP Compaq 6910p, CPU: Intel Core 2 Duo CPU T7300 @ 2.00GHz, RAM: 1978 MiB, GPU: Intel Corporation Mobile GM965/GL960 Integrated Graphics Controller (rev 0c), Resolution: 1440x900

ware being able to run on older hardware, which was the reason the full OpenGL 3.0 function set could certainly not be used and one reason no parts of OpenGL 2.1 were used but instead the limited fixed-function pipeline of OpenGL 1.X.

The C++ Standard Library was used for containers (STL) and string manipulation.

Tasks for development on glVisu were a more flexible colour selection with a high contrast mode and white background, making the hardware neurons selectable, displaying 'parameters' of them and doing the visualisation of the interconnections. Furthermore concepts for the visualisation of 'stage 2' and the interactive editing of the model should be developed. As the work progressed more tasks were introduced.

While the projected time for the actual work as well as the report was 4 weeks, this was more or less doubled.

A lot of time was spent as a first part on cleaning up and structuring the code base for the existing initial visualisation to make it more accessible and most of all easier to implement the given tasks, like the essentially quite easy dynamic replacement of colours. As it was promised work done on the program would be of actual use in the development and use of the neuromorphic hardware, time was spent on making it easy to further build on it and correcting as needed. Close to everything was revised in the process, leaving little untouched. The resulting source files have about 3300 sloc from previously about 1800 sloc

As no documentation between, preferentially graphic for easy access and reference, could be provided on the relevant parts of layout of the chip and its relation to the hardware model – possibly something the visualisation was supposed to be of use to, but questionable giving the lack of knowledge by the person doing it and given that of the 4 components of homogeneous coordinates only 2 are used – but only a couple of short conversations about the topic, the task of drawing the 'stage 1' emerged to be quite unsatisfactory. This, combined with the rendering speed problem and the development state (paradoxically the work was supposed to help with this), was the reason no work was done on 'stage 2'.

Because of this work was spent on a 2D display of the hardware and improving the source package in general for further development.

The tools `KCacheGrind` and callgrind from `valgrind` were used to some extend to profile the drawing process. The results were mixed owing to the abundant and intertwined function calling necessary in the OpenGL 1.x API on one hand and on the other hand the dramatic slowdown when running on the `valgrind` virtual machine and the difficulty of setting up a meaningful test. However, unfortunately there was a lot of compulsion for speed so quite a lot of time was spent on, regrettably, premature optimization.

The source code was documented in large part with comments in the Doxygen syntax and so can be compiled into a software reference documentation. While this was previously the case, this needed work giving the amount of change.

For further improvement of the application a couple of things were investigated.

Looking into the cross-platform application framework Qt for a native C++ replacement of GLUT for improved and easier usabilty, and better multi-view handling, seemed promising and resulted in more separation of the displaying process.

Improvement of the stereoscopic display was researched.

First and foremost for faster rendering, but also for more rich visual presentation the features that separate OpenGL 3.0 and 2.1 from 1.X like Vertex Buffer Objects were looked into. More advanced animation techniques like keyframe animation were considered.

For better and easier image generation the possibility of rendering in higher resolution and output with a library like `libpng`.

The work on glVisu was incorporated in the dissertation of Daniel Brüderle [7, section 3.2.6 / pp. 68-71] in the form of a couple of screenshots.

The looping functionality of the simulation was done in very short time, and given the groundwork very easily, a short time after the end of the internship to be used in a televised interview.

4 Contributions

4.1 Improvements

Beyond the tasks to be completed a couple of other things were done to lay the groundwork or be of help in the development.

To improve speed additional `structs` were introduced as suggested in [5] for the relevant data from the hardware graph, `hw_neuron`, `hw_core`, `hw_syndriver` (which is an either internal or external input with either positive or negative synaptical weight for all neurons of a core), which keep references to their corresponding nodes for less frequently used information, in addition to `bio_neuron` and `bio_synapse` which were already in existence.

The workings of these structures were changed in the way the they reference each other. Instead of keeping the index into an array, this indirection was avoided to gain speed and let later manual editing by the user become feasible without , by directly linking the nodes together. For this more difficult process a map was used in the new `transferGraphs()` member function, using the pointer as a key to semi-finished synapses. If parallel editing of the original graph and the copy turns out to be impractical, this transferal could be repeated, but this is certainly bad for much larger graphs.

The code was cleaned up and partly rewritten. A first step was automatic formatting and rearrangement. The code makes use of namespaces, enums. Of the Standard Library `string`, `ostringstream`, `istringstream`, `list`, `vector`, `set`, `map`, `pair`, and the respective `iterators` were used, with for example the capacity reserved in advance. C-style strings were avoided as much as possible.

Instead of permanent casting the OpenGL types were used where sensible. It was aimed for const correctness.

The parsing of commands was somewhat simplified. The shortcuts were made to use keycodes where character literals could accidentally replaced or mistaken.

The layout algorithms, which were previously assuming a specific neuron count, were corrected and reverting to the positions from the model was made possible.

A command was introduced to put the center of masses and bounding boxes of the bio neurons and the hardware in respect to each other independently of their positions in their own private frame of reference. And another command to set the position of the origins of the bio model and the hardware model in the scene, putting the matrix stack to good use.

The stereoscopic mode was revised and made more flexible given more and a colour theme introduced with a constrained like

palette so as to have three dimensional parts of the image always in both independent images.

The hardware neurons were made able to be selected in consecutive order, like previously the bio neurons, but with both enumerations repeated at the ends.

Information in the graph model regarding the synapses, hardware neurons, synapse drivers, in addition to the bio neurons is now given in the on-screen display and easier extensible. Giving the groups of information translucent background boxes was tested, but is not in the final version.

A very helpful new feature is saving the position and viewing direction to be able to restore them later using shortcuts.

To help with orientation and the positioning using the commands, coordinate axes are drawn colour-coded RGB for the x (red), y (green), z (blue) coordinates of the right-handed coordinate system. Also for orientation crosshairs were added.

The position in space and line of direction can now be set with the console and outputted to the console. Mind that the output values are the rounded off accumulations of multiple steps with in floating point arithmetic, so feeding back the values will result in slightly different position and orientation.

The simulation mode was improved with an integration time setting by which can be controlled how long a spiking neuron is displayed as such. The outbound connections can be shown. Furthermore the simulation can be paused, for example for inspection. The animation can now be looped, ended at the last event or continue running like previously. The last option was kept combined with the ability to enter simulation mode without a valid simulation file in order to get an idea of the frame rate.

Boost.Python installs a signal handler that prevents termination while running the unmanaged C++ code. For that reason a requirement was to install a custom handler for the `SIGINT` signal. After the interruption the old handler is restored.

As a measure to prevent unintentional termination, the shortcut was changed to use a key combination. Conversely it leaving the full-screen mode was made to be more in line with expectations.

It was tested to use only decimal fractions that are exact in binary representation with only more significant mantissa bits set, preferably few, for better graphics quality.

4.2 Changed

The responsibility of the classes was changed with `GraphVisu` as an entry point, handling of the data and drawing of the models, and `glControl` for user interface, management of OpenGL and control flow. See Section 5.3 for obvious improvements.

By using the freeglut extension to GLUT `glutExitMainLoop` and the option `GLUT_ACTION_CONTINUE_EXECUTION` the API user can now simply use a blocking call to `glControl::show()` instead of knowing the intrinsics of a threading workaround.

The menu was newly arranged and further completed. Various menu entries for displaying options of the model were added, for example to show the hardware only. It was made more compact by having a single toggle entry instead of two normal ones, which was also done for the keystrokes. Instead of simulating key presses, the keystroke handler now invokes the menu functions which allows for easy reassignment of the keys but could be further improved by more common functions.

The shortcuts for plane, sphere and cube positioning were permuted so as to be more mnemonic.

4.3 Drawing of the hardware

While previously only the hardware neurons and the labels of the cores were in the scene, now more components are shown: the synapse drivers, the array of their output weights and various connections like the connection of hardware neurons to other hardware neurons via the synapse drivers and the array.

The hardware neurons were given more fitting little cubes using `glutSolidCube`, while for the synapse driver a similar function `solidWedge(length,height, , width)` with correct normals was programmed. The core is now surrounded by a rectangle and the core names slanted for viewing in top-down view.

The difficult part is the weight matrix, because of the more than squared amount of weights. For that reason they are but little squares that have a linear interpolated colour depending on their value from low values to high values and sign, with a special colour for zero.

Synapse drivers getting their input not from their core are coloured differently, the same holding true for mapped hardware neurons.

The hardware was more or less drawn in a more logical than physical way. Completely unattached from that fact a problem was that the weights couldn't be read from the graph model as they are not included in the stage 1 graph model. For that reason a realistic configuration was not depictable. Rather random values were generated with $p(0) = \frac{3}{4}$ and the weight linearly decreasing down to zero for increasing weights by inverse transform sampling, which was tested. The synapse drivers that are configurable with numbers smaller than 192 were given a 40% chance to be `FEEDBACK` or `FEEDBACK2` instead of `EXTERNAL`.

4.4 Selection

Previously out of the object in the scene the user could only select the bio neurons. Now in addition to focussing on bio neurons, the user can click as well on bio synapses, hardware neurons, mapping connections between the two graphs, and synapse drivers. When connections are selected information is also given on the binding partners.

In addition to the focus mode, a select mode was incorporated, which can be activated through the console. In this mode focus is not changed, but sets of objects can be selected, which can be read out later from data members of `GraphVisu` with a name of the form `selected*`, as a way to pass information on to the controller. The invoker can also read out the last selected item type, the last selected item per type. The same holds for the items. This was done to make the visualisation more useful to the mapping process, by not only showing the result without outcome, as progress to modification within the visualisation instance was limited to the improvement of the graph transferral and data structures (see 4.1). It could be used for batch processing as well.

The detection is done by using OpenGL support for picking with the same basic principle that it was done previously. With the help of `gluPickMatrix()` a projection matrix is calculated, that restricts the view volume in such a way that its projection on the screen surface is within the surroundings of the mouse cursor. By setting a special rendering mode by invocation

of `glRenderMode(GL_SELECT)` all the primitives of all the objects that are then rendered get an assigned key value, that is chosen to be distinct for each object, in a buffer that has previously to be specified. The key value is then mapped back to the object with the help of a table. Since multiple types of object are now selectable, each type got its own table that is populated anew for each drawing cycle with the help of a `numDrawn` variable and the keys given to the API have an offset calculated from the total number of the elements of each type. As this packs the table entries tightly, care must be taken not to draw elements multiple times. There is a way to built a hierarchy, but it was not used as it appeared to be a waste of space. Another approach would have been to use parts of pointers for identification. Use of the special rendering mode can be avoided by colour-coding the objects and then reading the pixel values from the frame buffer, but this makes the selection of underlying objects, which was planned, quite harder.

4.5 Colour management

All occurrences of colour usage that were previously distributed as literals all over the code base organized with a `color_scheme` structure with member names that correspond to the logical meaning. At the moment there are three hard-coded themes which can be selected using a `ColorSchemes enum`, including a higher contrast one with white background for printing and presentation with a beamer. This management allows for easy introduction of nice improvements like coloured headings of the parameter boxes that separate them visually. `style_scheme` is another structure that, at the moment, bundles the properties of synapses, including how thick the connections are drawn depending on synpatic weight.

Previously only either inbound or outbound connections could be viewed for a selected neuron. By using OpenGL line stippling simultaneous presentation became reasonable.

4.6 Viewing modes

Since a 2D view of the hardware is simply more practical a 2D view mode was integrated which can also be used as an overlay to the conventional 3D mode giving a bombardier perspective. This last, effectively splitscreen, view is more useful with wide aspect ratio of for example 16:9. The zoom factor of the 2D view and the field of view of the 3D view can be set using the console. Combined with the commands and the underlying member functions for placement this proved to be useful in development. At the same time it made the display code more general.

However, because the scene is now rendered twice, it is unpleasantly slow on the development system.

4.7 Rendering speed

With the provided target hardware the frame rate is noticeably slow, especially in the split screen mode. By running the simulation in continuous mode the update rate can be seen without changing the view manually (when simulating as an end in itself the hardware model should be hidden).

The bottleneck is the high vertex count (589 824) for the triangles (196 608) of the two cores. In comparison to contemporary computer games this is quite high given the simplicity.

In addition to the display list, that can be compiled by the graphics driver into an efficient format and saves data transfers and function calls, that was used previously for the spheres, display lists for the other items were used. Each was core was given a list, handled by the core structure, which is updated as needed. Replacing the squares with dots was tested, resulting in better performance but unusable graphics and furthermore painting only non-zero weights.

Leading to the conclusion that that in this way not many cores can be dsplayed.

5 Proposals

5.1 Miscellaneous

...

By switching to Qt, the user interface for editing of the graphs would be easier to develop.

If not choosing to replace GLUT then the menus could be improved by rebuilding the menus after each change of state.

...

The bio model should be scalable and rotatable around the vertical axis.

...

Entering commands would be a lot more useful if a command history was retained. The GNU History library could be used, making it easy to persist.

...

There should definitely be an Easter egg in the form of replacing the bio neurons with Utah teapots.

5.2 Rendering speed

While the drawing could further benefit from additional display lists for drawing of the bio neurons, especially if there are a lot more of them or all synapses are displayed, this will not solve the problem of too many weights without dramatically better hardware of the far distant future. But as the weights are not really that important or helpful if they are far away, they can simply be left out for cores that are not in the field of view or distant ones, or somehow combined into a meaningful form. Animation would be quite hard.

However, it could be that using more recent features of OpenGL can bring improvements. Using Vertex Buffer Objects could be useful by avoiding branching and save the initial costs, but the display lists probably use a similar mechanism anyway and then a lot of colour values, that are not very informative, have to be copied.

For better analysis a frames per second indicator should be included.

By rewriting to use GLSL shaders for the different components the graphics could not only improve, but another technique becomes possible: using texture memory to store the values of the weights and then a fragment shader to read out the right value. Rather than compiling the texture with restricted resolution and then simply applying it to the area of the weight array, this could be made to look similar and even more appealing to what is now being displayed.

5.3 Software architecture

While there were a couple of reasons the two existing classes were kept it seems natural to divide them up massively into a lot more classes and source files.

Most importantly the hardware model should be drawn by a hardware version specific subclass of a more general hardware drawing class. This would make necessary changes to the layout algorithm as the graph

model changes localized to this class alone. The stage 2 hardware could be another subclass.

There should be a class or template for encapsulation of the selection mechanism, making it much less verbose, redundant and maintainable.

The console should be managed by a class for its data and parsing, and using a command line history class. Another class can then display the state on the screen.

There should be a single concise class to read spike trains, that even could be easily reused.

The and keystroke and menu handlers and their callbacks should be put into a separate file respectively.

Instead of retaining switching depending on a lot of flags, special subclasses that draw a part could be used in some cases to save some of the accompanying costs and make the drawing code more readable.

There should be a better setup to compile the code.

6 Known issues

Kno

Appendix A

Updated user manual

1 Invocation

Terminate using the window controls of the platform, the menu, shortcuts, the console, or from the Terminal with Ctrl+c.

1.1 PyNN

The visualisation is displayed when passing the `visualization` keyword argument to the simulation function `PyNN.run`.

1.2 C++

Usage of the visualisation can look like the following. The models have to be properly initialised.

```
1 BioModel bioModel;
2 HWModel hwModel;
3 // ... initialise models ...
4 GraphVisu* gv = new
      GraphVisu(&bioModel, &hwModel);
5 gv->control->show();
```

Before the blocking call of `show()` properties of the display can be changed on the `GraphVisu` class instance. A Doxygen API reference can be compiled from the header files.

2 Shortcuts

See Table 1.

3 Commands

See Table 2.

Remember the console is opened with ⇧ +~, ^.

4 Menu

See Table 3.

Key	Action
w, a, s, d	move forward, left, backward, right
r, f	move up, down
Esc	*exit visualisation*
Esc	*exit full screen mode*
⇧ + Q	exit visualisation
⇧ + F	maximize window
g	resize window to 800x600
⇧ + ~, ^	open the console
⇤⇥	print information (*position and direction of the camera*)
o	*cycle* through the bio neurons
Alt + o	*cycle through the hardware neurons*
k	*toggle* display of synapses
e	*toggle* display of synapses in the animation
h	*toggle* display of hardware
n	*toggle* display of core names
i	*toggle* display of mapping
c	start the simulation
⇧ + C	end the simulation
Space	*pause / continue animation*
1, 2, 3, 4, 5	timestep 0.01 *(new)*, 0.1, 1.0, 10, 100 (default 1.0)
u	position neurons on a cubic lattice
p	position neurons on a square lattice
j	position neurons randomly within a sphere
⇧ + J	*reset positions to the original positions from the bio model*
⇧ + V	*save view*
v	*load view*
t, ⇧ + T	enable / disable culling
z, ⇧ + Z	*toggle cull back / front faces*
b, ⇧ + B	switch transparency on / off

Table 1: Shortcuts available. New functionality in italics.

Command	Action
↵	close the console
quit	*exit the visualisation*
help	*print (not really helpful) help to stdout*
`set neuronid <n>`	select a specific bio neuron
set mode select / set mode focus	*change mode to select/deselect bio neurons, bio synapses, hardware neurons with the mouse into a list with visual feedback, or to show information and mapping*
`set anaglyph` / `unset anaglpyh`	turn on/off anaglyph effect for
`set hwmodel` / `unset hwmodel`	show/hide hardware
`set bio_conn` / `unset bio_conn`	draw synaptic connections in the bio model
`set param` / `unset param`	Show/hide the names and parameters from the graph model of selected items
`set simfile <name>`	use the file with the provided name for the simulation
`set simtime <t>`	set the simulation time to the instant provided
`set timeinterval <interval>`	set the step size of the simulation (default 1.0)
set integration <time>	*set for how long after a spike event a neuron is displayed as active (default* 10.0*)*
set borigin <x> <y> <z>	*set the origin of the bio neuron locations to the specified coordinate*
set horigin <x> <y> <z>	*set the origin of the hardware to the specified coordinate*
set origins	*set the 'center of mass' of hardware and bio model to lie on the y axis, with the hardware below $y = -5$ and the biomodel on top of $y = +5$*
set pos <x> <y> <z> <heading> <pitch>	*set the camera location to the given position (heading and pitch in degrees) (default* $(-15.5, 1..0, -7.0, 135.0, -22.5)$
set pos <x> <y> <z> <target-x> <target-y> <target-z>	*set the camera location to the given position, looking at the target position*
set 3d	*set the view to 3D (default)*
set 2d	*set the view to 2D, looking down along the y axis*

set 3d2d	*set the view to 3D with a 2D bombardier perspective looking down along the y axis displayed on the right side*
set fov <fov>	*set the field of view in degrees for 3d display (default* 50*)*
set zoom <factor>	*set zoom level for 2D display (default* 0.0625*)*

Table 2: Commands available in the console. New functionality in italics.

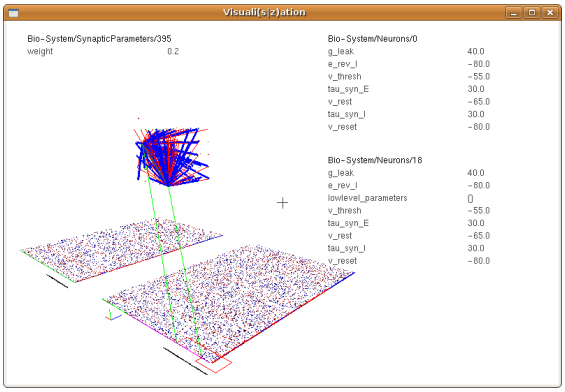
Menu entry	Action
Positioning » *x*	menu for positioning of the bio neurons
x » Original (J)	*reset neurons to the positions provided by the graph model (default)*
x » Cube (j)	put the neurons on a cubic lattice
x » Plane (p)	put the neurons on a square lattice
x » Sphere (u)	put the neurons randomly on the inside of a sphere
Performance » *x*	menu for performance and testing settings
x » Select GL_FLAT shading model	use the simpler flat shading model
x » Select GL_SMOOTH shading model	use the more complex smooth shading model (default)
x » Enable culling	enable the culling of back faces and / or front faces (default)
x » Disable culling	disable culling
x » Cull back face	*cull back faces, show front faces (default)*
x » Cull front face	*cull front faces, show back faces*
x » Enable fog	*enable simple OpenGL fog*
x » Disable fog	*disable fog (default)*
Full-screen » *x*	menu for full-screen display (leave with Esc
x » 640x480	
x » 800x600	
x » 1024x768	*new*
x » 1280x800	
x » 1280x1024	
x » 1440x900	
x » 1600x1200	*new*
x » Leave full-screen	leave full-screen mode
Simulation » *x*	menu for the animation of a spike train from a file
x » Start simulation	*change in to simulation mode*
x » Pause / Continue	*toggle running of the simulation*
x » End simulation	*leave the simulation mode*
x » Reset time to zero	*reset simulation to the beginning*
x » Continuous simulation	*keep simulation running at the end of the spike train*
x » Load file	load the simulation file
x » Loop animation	*continue the simulation at the end of the spike train beginning at its start*
Display options » *x*	menu for displaying options of the bio model and the hardware model

Menu entry	Action
x > Show parameters	show on-screen display with information on the selected items (default)
x > Hide parameters	hide on-screen display
x > Toggle bio graph	*show/hide bio model* (default show)
x > Toggle targeted neuron identification	*emphasize postsynaptic neuron* (default on)
x > Toggle synapses	*show/hide synapses* (default show)
x > Toggle incoming synapses	*toggle* display of synapses from presynaptic neurons *independently of those to postsynaptic neurons* (default hide)
x > Toggle outgoing synapses	*toggle* display of synapses to postsynaptic neurons *independently of those from presynaptic neurons* (default show)
x > Toggle only synapses of focused	*show/hide synapses from focused neuron or all synapses* (default single)
x > Toggle animation of spiking synapses	*display the synapses (incoming / outgoing like configured) of a spiking neuron in the simulation also* (default off)
x > Toggle hardware graph	*toggle* display of the hardware model (default on)
x > Toggle core names	*toggle* display of the core labels (default on)
x > Toggle mapping	*toggle* display of the mapping connections between bio model and hardware model (default on)
x > Toggle only mapping of focused	*switch between displaying all mapping connections or those of a focused hardware or bio neuron* (default single)
x > Transparency on (b)	turn transparency on
x > Transparency off (B)	turn transparency off (default)
x > Maximize window (F)	*maximize* the visualisation window
x > Enable anaglyphic 3D	render the scene as a 3D anaglyph image
x > Disable anaglyphic 3D	turn of anaglyphic 3D (default)
Colors > x	*menu to change between colour themes*
x > Black is beautiful	*black background*
x > Higher contrast	*white background*
x > Safe colors for anaglyph images	*use colours that have components in both filtered images*
Quit (q)	leave the visualisation

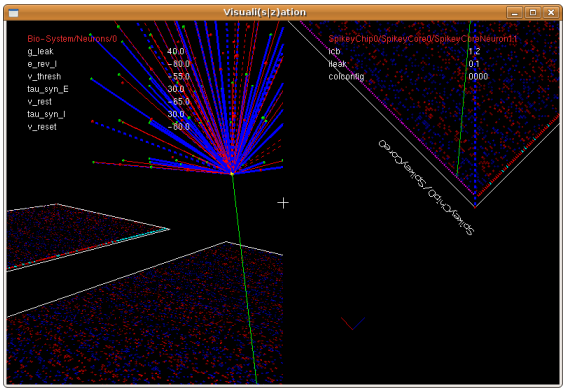
Table 3: Menu items available. New functionality in italics.

Appendix B

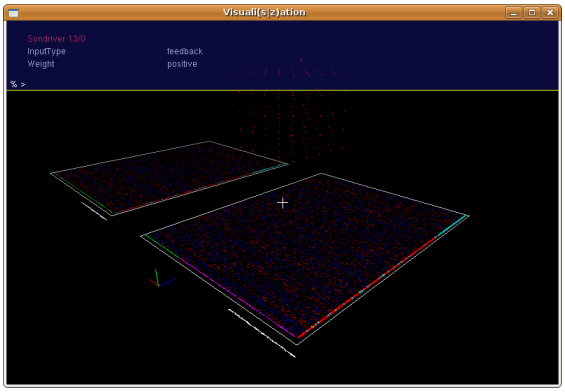
Screenshots



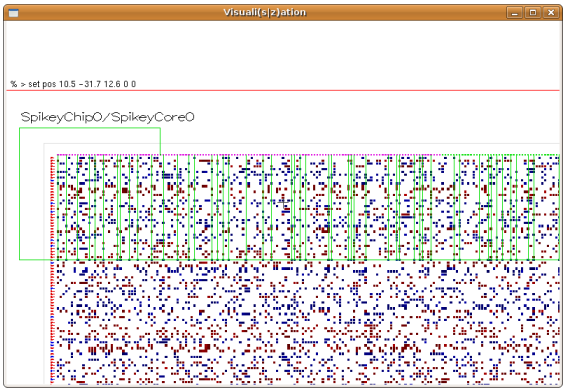
(a) synapse selected and mappings between two pairs of bio and hardware neurons, repositioned bio model, light theme



(b) 3D2D mode with inbound connections shown, mapping connection shown in both views, crosshairs visible



(c) console open, syndriver selected, crosshairs visible, fog turned on, bio graph hidden, axes visible, starting location



(d) 2D mode, console showing positioning command, parameters and bio model hidden, light theme

Figure 1: Four images with the same graphs. Colour `weight_disabled` set to translucent for the screenshots.

References

[1] URL: http://www.kip.uni-heidelberg.de/vision/.

[2] J. Schemmel, J. Fieres, and K. Meier. "Wafer-Scale Integration of Analog Neural Networks". In: *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. (June 1–8, 2008). Hong Kong: IEEE Press, 2008, pp. 431–438.

[3] D. Brüderle, E. Müller, A. Davison, E. Muller, J. Schemmel, and K. Meier. "Establishing a novel modeling tool: a python-based interface for a neuromorphic hardware system". In: *Frontiers in Neuroinformatics* 3 (2009), p. 17.

[4] URL: http://facets.kip.uni-heidelberg.de/.

[5] T. Harion. "3D-Visualisierung einer Abbildung von neuronalen Netzwerkmodellen auf eine neuromorphe Hardware". In: (2008), p. 33.

[6] *Programming guide: the official guide to learning OpenGl, Vers. 1.2*. eng. 3. ed., [Nachdr.] Reading, Mass.: Addison-Wesley, 2002, XL, 730 S.

[7] D. Brüderle. *Neuroscientific Modeling with a Mixed-Signal VLSI Hardware System*. 2009.