Visualization of the Mapping from a Neuronal Network onto a Neuromorphic Hardware System

Intership report from
Benjamin Hepp
supervised by
Daniel Bruederle

April 5, 2010

Contents

1 Introduction

In the course of my internship for my minor subject computer science I extended a visualization software for the mapping of a biological neuronal network onto a neuromorphic hardware.

1.1 Motivation

The research group Elecronic Vision at the Kirchhoff Institute for Physics of the University of Heidelberg has developed an analog neuromorphic hardware [8][9]. Within the European research collaboration FACETS [1] this hardware is used for neuroscientific experiments. The main advantage of such an approch compared to a software simulation is a significant increase in speed due to the complete parallelization of the neuronal functionality. The neuronal networks that are to be investigated can be described with the generic Python [6] library PyNN [5]. This library is also used for software simulators and thus it is an easy step to transfer an experiment from a software simulator to the hardware emulator. The biological neuronal network described with PyNN has to be mapped onto the hardware. This mapping is rather complicated and thus it is of interest to be able to visualize the mapping between biological network and hardware routing in a clear way. This can help to find systematic errors or flaws in the mapping algorithms but it is also useful for presentation and teaching.

1.2 Hardware Mapping and Limitations

There are different stages of the neuromorphic hardware developed by the Electronic Vision group. There already exists a visualization software for the stage 1 [8] and stage 2 [9] hardware. My task was the improvement and extension of the visualization of the stage 2 hardware. The stage 2 hardware building blocks are so called HICANN chips which consist of two times 256 neurons and about 100000 synaptic connections each. These HICANN chips can be connected and eventually there will be about 400 HICANN chips placed on a single wafer. Of course, as there is a final number of hardware connection lines, the connectivity of the hardware neurons has limitations. Also many parameters of the neurons and synapses are specified in groups and not individually. It is the task of the mapping software to find an appropriate mapping between the biological network and the hardware [11].

1.3 The Grahpics Library OpenGL

The visualization of the mapping is achieved with the graphics library OpenGL [4]. This is a hardware and operating system independent library for 2D and 3D graphics. There are drivers for every modern graphic card that implement the OpenGL library and make use of hardware acceleration. Other platform specific tasks such as window management and mouse and keyboard control are performed by the GL utility library freeglut.

1.4 The GraphModel

To perform the mapping of the biological neuronal network onto the hardware the TU Dresden has developed a graph model [11] that describes the biological neuronal network and the corresponding hardware configuration and is used by rather advanced mapping algorithms. The graph model is used to access the information that is necessary for the visualization.

1.5 Software Architecture

The visualization software is split into two parts:

- glControl
- GraphVisu2

glControl takes care of all things except the drawing and reading of the graph model. This includes mouse and keyboard control, window management, menu creation and control, timing for the animation, terminal functionality.

GraphVisu2 reads information from the graph model and transfers a part of it into local data structures. It also performs all the drawing of the biological network and the hardware including the layer 1 routing.

Instead of describing how the whole software works I refer to the reports of Tobias Harion [10] and Radoslav Rusanov [7] and only describe how the visualization of the layer 1 routing is achieved.

2 Contribution to the Visualization Software

This section describes the new features that were implemented by the author during his internship.

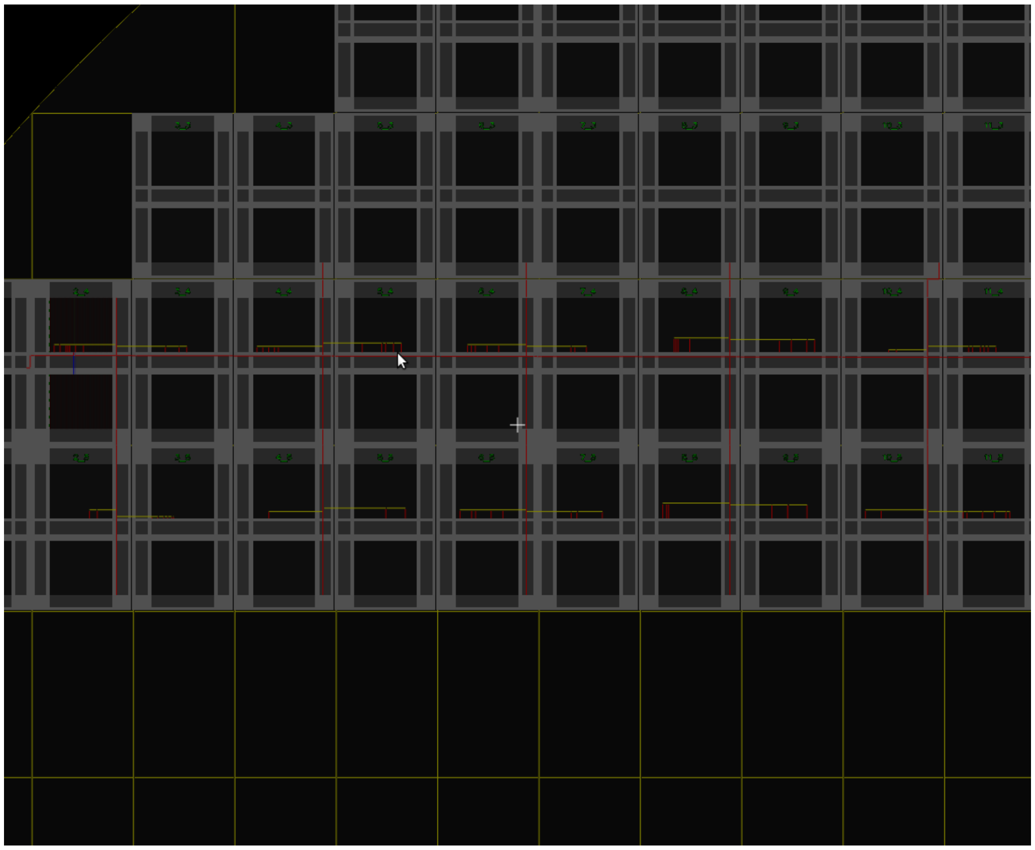


Figure 1: Screenshot of the 2D projection of the hardware

The new features that the author introduced to the software include a 2D-projection of the hardware (see figure 1), the visualization of the layer 1 routing on the hardware (see figure 2, 3 and 4) and the animation of the network activity in the biological neuronal network as well as on the hardware visualization (see figure 6, 7, 8 and 9). Also some cleanup and numerous bug-fixes and small improvements to the code have been made during the intership.

2.1 Drawing of the Layer 1 Routing

To draw the layer 1 routing for a selected neuron or hardware dendrite there are three seperated tasks to perform:

1. Explore the graph model to retrieve information about the layer 1 routing and build up a routing tree for the selected hardware dendrite

2. Create a list of lines, rectangles and diamonds that represent the layer 1 routing on the hardware

3. Actually draw the lines, rectangles and diamonds

The last task has to be repeated whenever the screen is drawn. The first and second tasks only have to be done once for every neuron as the list of lines, rectangles and diamonds is going to be cached. Also the first and second task are done on the fly only when they are actually needed (e.g. due to selection of a neuron or because a neuron spikes during an animation).

To explore the graph model and build up a routing tree for a hardware dendrite there is a simple tree structure defined in *GraphVisu2.h*:

- *struct simple_gmnode_tree_node* represents a node of the routing tree. It contains a pointer to a *GraphModel::GMNode* and a list of children nodes

- *struct simple_gmnode_tree* simply contains the root node of the tree

The routing tree is built up by calling the recursive method *exploreConnections()* of *GraphVisu*. Already explored *GraphModel::GMNode* objects are marked by adding a prefix to their name. These prefixes have to be removed later by calling the method *cleanupTreeNode()* of *GraphVisu* for all routing tree nodes.

After the routing tree is build up, the method *calcConnectionLines()* of *GraphVisu* is called to create the lines, rectangles and diamonds that represent the layer 1 routing. To save these graphic objects there are three structures defined in *GraphVisu2.h*:

- *struct glLine* represents a colored line

- *struct glQuad* represents a colored quad which can be either a rectangle or a diamond

- *struct hardware_routing* contains a vector of *glLine* objects and a vector of *glQuad* objects

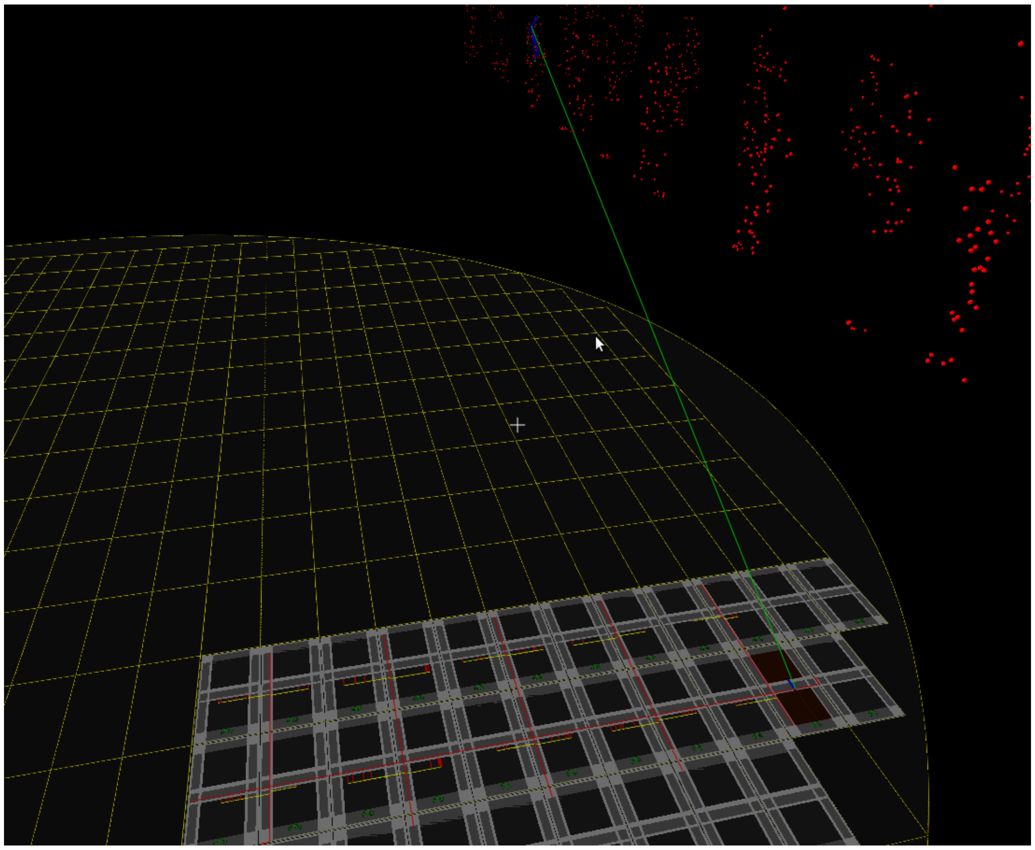


Figure 2: Screenshot of the layer 1 routing showing the biological network and the hardware

The class *GraphVisu* has a vector containing one *hardware_routing* object for each neuron.

After the graphic objects have been created they can be drawn by calling the method *drawConnectionTree()* of *GraphVisu*.

3 Howto: Using the Software

When started, the visualization software shows the biological neuronal network and the corresponding hardware on the screen. Initially a neuron is selected and drawn in a different color. A line connects this neuron with the

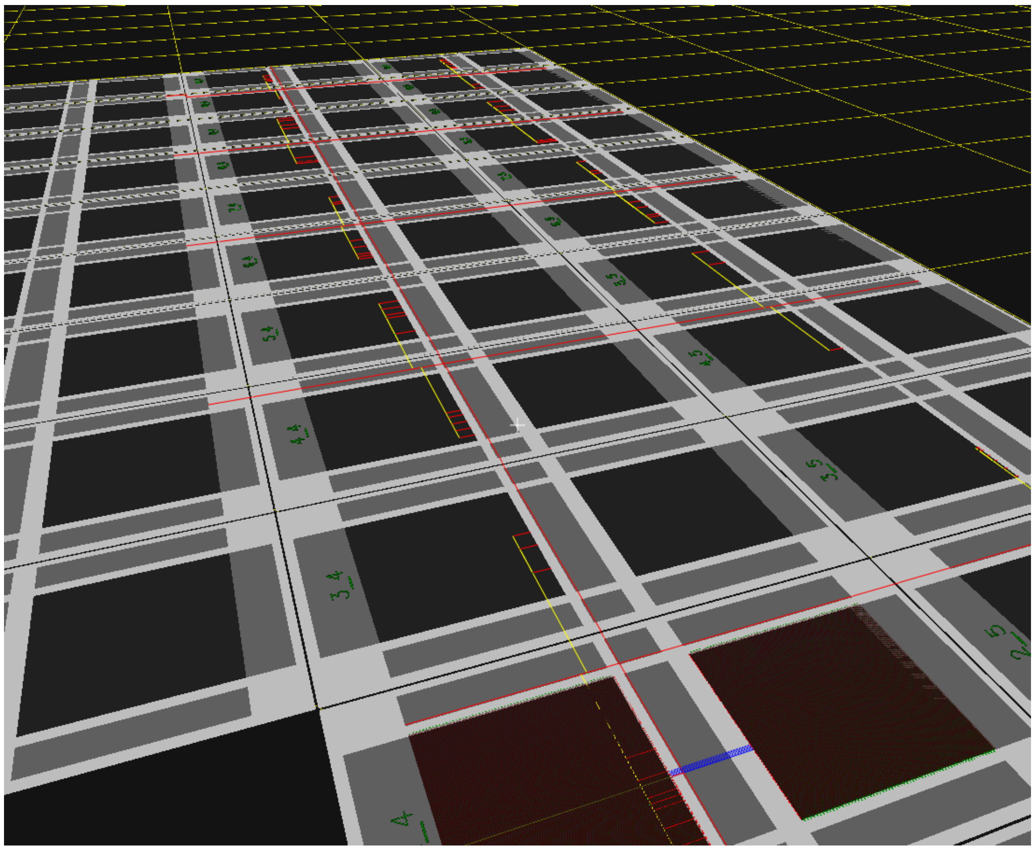


Figure 3: Screenshot of the layer 1 routing showing an overview of the hardware

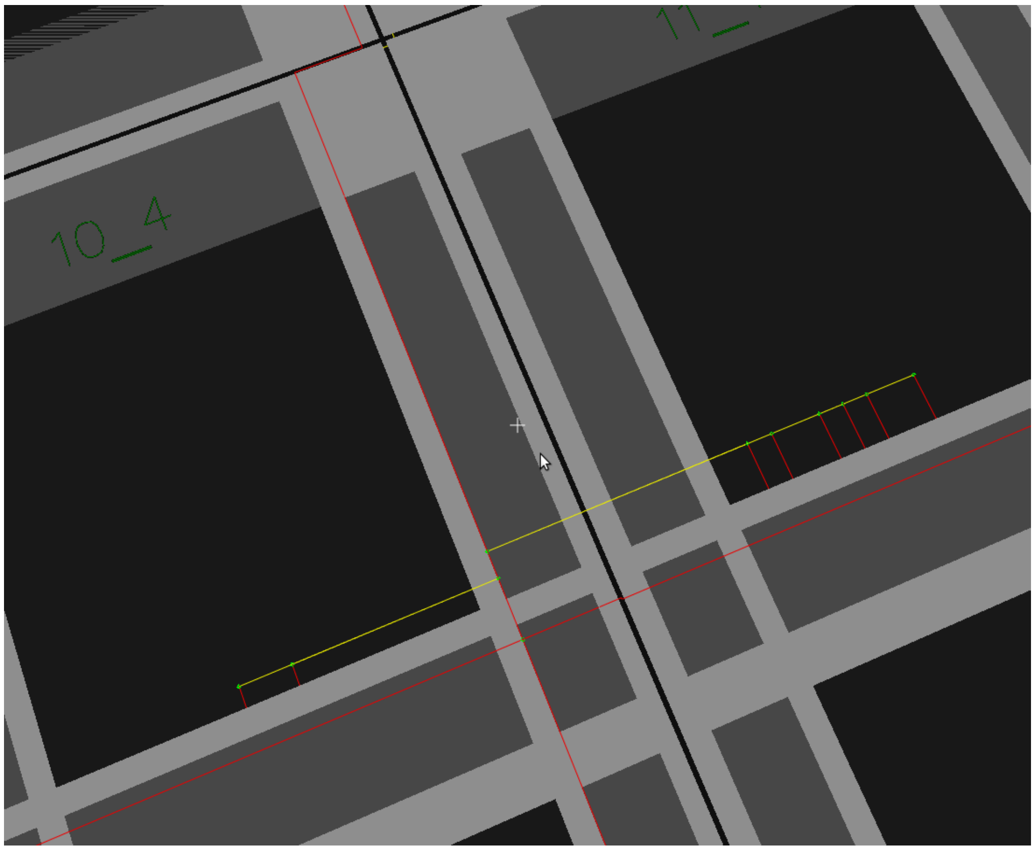


Figure 4: Screenshot of the layer 1 routing showing a small part of the hardware

corresponding dendrite on the hardware and the layer 1 routing of this dendrite is also shown. It is possible to select different neurons by either clicking on the biological neuron or by clicking on the hardware dendrite on the focused HICANN (due to drawing performance only the HICANN containing the currently selected dendrite is focused).

Navigation through space is possible using the e and d key for moving forward or backward respectivly. The s and f key are used to strafe left or right respectivly. It is also possible to move up or down by pressing the t or g key respectivly. One can zoom into the current view by pressing q and zoom out by pressing a. The movement and zoom speed can be changed using the terminal (see the section on using the terminal).

The view can also be panned by clicking and holding the left mouse button and moving the mouse. After the view has been changed, the mouse button can be released.

3.1 Keyboard Layout and Mouse Control

The left mouse button is used to select objects on the screen and to navigate through the menu which can be activated by clicking the right mouse button.

Key	Description
→	Progress simulation for one timestep
←	Rewind simulation for one timestep
↑	Increase simulation timestep by a factor of 2
↓	Decrease simulation timestep by a factor of 2
c	Start simulation
C	Stop simulation
<SPACE>	Toggle animation
V	Save viewport
v	Reload viewport
<TAB>	Print viewport information
J	Use original neuron positions
u	Distribute neurons on a cube
p	Distribute neurons on a plane
j	Distribute neurons on a sphere
k	Display synapses
h	Display hardware graph
i	Display mapping
q	Zoom in
a	Zoom out
e	Move forward
d	Move backward
s	Strafe left
f	Strafe right
t	Move upward
g	Move downward
∼ or ˆ	Open terminal
<ESC> or Q	Quit

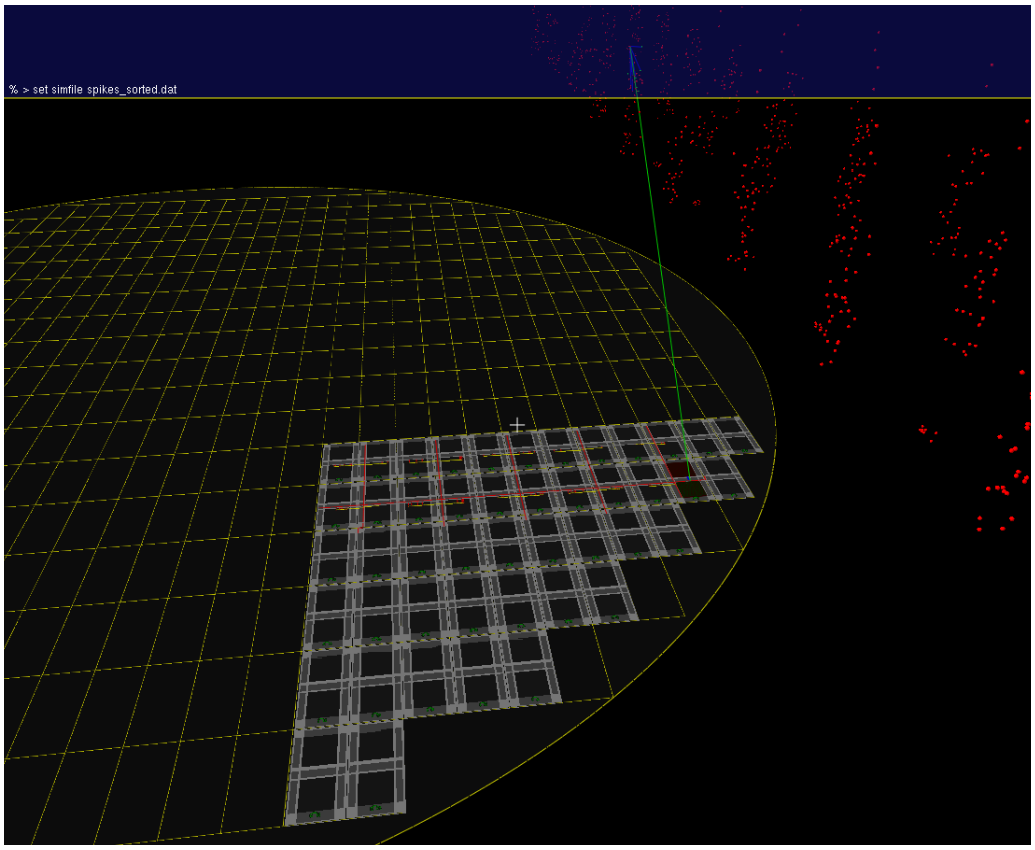
3.2 Using the terminal

The terminal can be used to enter more advanced commands (see figure 5). Following is a list of available commands:

Command and parameters	Description
quit	Quit program
help	Print help text
<un>set anaglyph	Activate/Deactivate anaglyph mode
<un>set hwmodel	Show/Hide hardware graph
<un>set bio_conn	Show/Hide synapses
<un>set params	Show/Hide parameters
set 2d	Switch to 2D projection
set 3d	Switch to 3D mode
set simfile <filename>	Load simulation data from file <filename>
set simtime <time>	Set current simulation time to <time>
set neuronid <id>	Select neuron with number <id>
set timeinterval <interval>	Set simulation timestep to <interval>
set speed <speed>	Set zoom speed to <speed> (default is 0.3)
set strafespeed <speed>	Set strafe speed to <speed> (default is 0.3)
set liftspeed <speed>	Set lift speed to <speed> (default is 1.5)
set mousexspeed <speed>	Set x speed of panning to <speed> (default is 0.4)
set mouseyspeed <speed>	Set y speed of panning to <speed> (default is 0.4)

3.3 Animation of Spiking Data

To view the animation of spiking data a simulation file has to be loaded using the terminal (see previous section). The file has to be in a two column format (both columns are seperated by a single space). Each row represents a single spike event. The first column specifies the time of the spike event and the second column specifies the index of the spiking neuron. After the simulation file has been loaded, the simulation can be started by either using the keyboard shortcut or the menu. While the simulation is running a time-bar is shown on the bottom of the screen and the current simulation time is printed in the bottom left corner of the screen. The simulation can be paused and continued using the <SPACE> key. When paused the simulation can be progressed or rewinded by single timesteps using the → or ← key respectivly. The timestep itself can be increased or decreased by a factor of 2 using the ↑ or ↓ key respectivly. During the simulation spiking neurons are drawn in a different color than other neurons and fade back to the non-active color in a biologicaly relevant time in the order of 10ms. Also the routing on the hardware graph is drawn for all spiking neurons and also fades away after



```
% > set simfile spikes_sorted.dat
```

Figure 5: Screenshot of the terminal in use

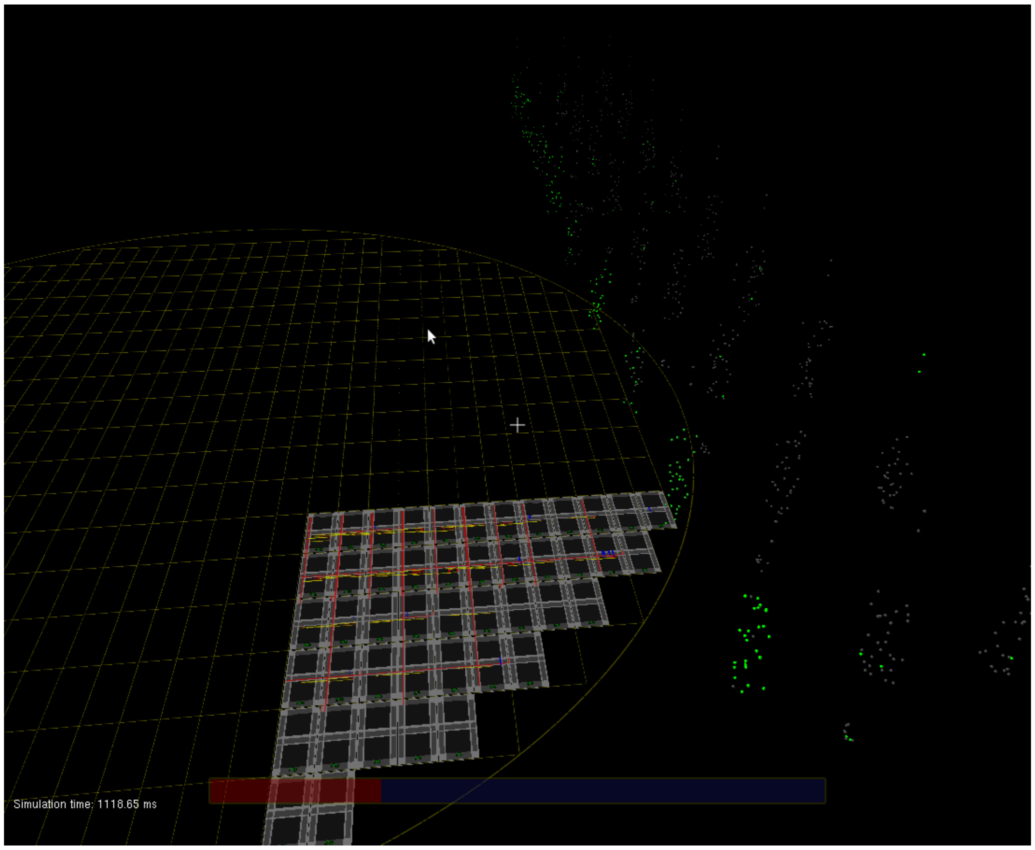


Figure 6: Screenshot of the spiking animation with column 1 being active

some time.

4 Outlook

In its current state the software can be used for demonstration and teaching purposes and also to track some flaws in the mapping algorithms. Nevertheless the software [3] can still be improved and extended. Following is a small list of features and improvements which might be worth implementing at some point:

- Visualization of the layer 2 routing

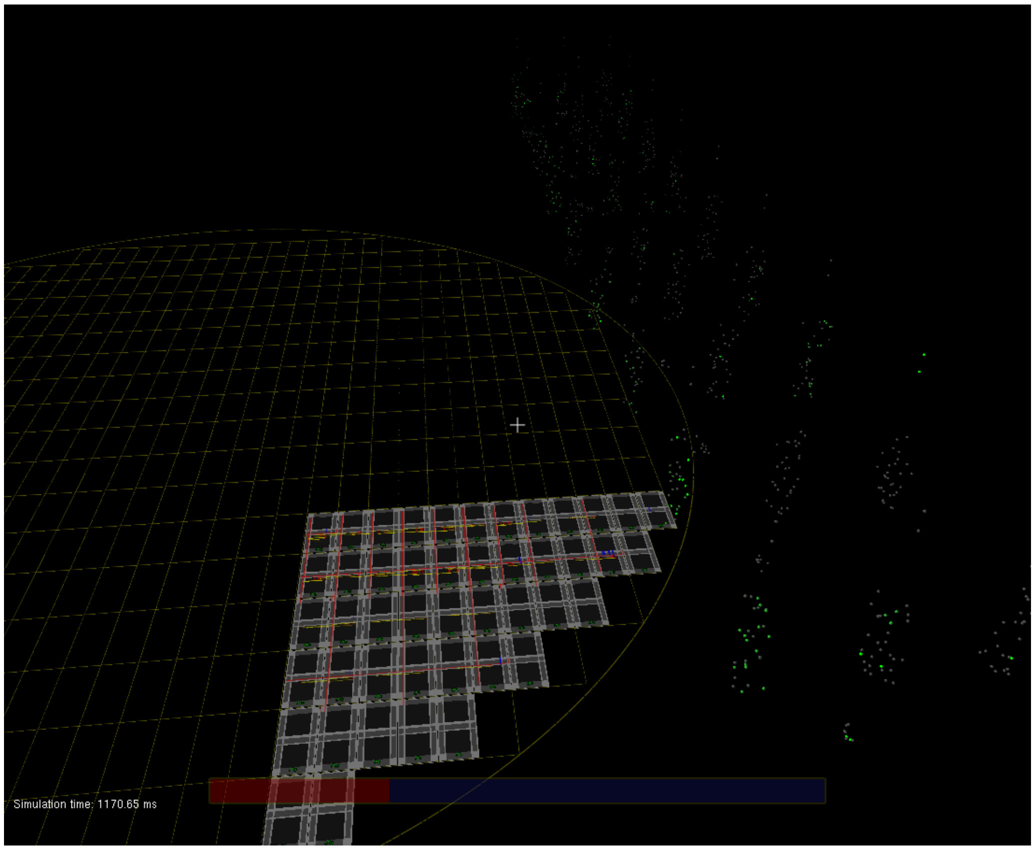


Figure 7: Screenshot of the spiking animation with column 1 activating column 3

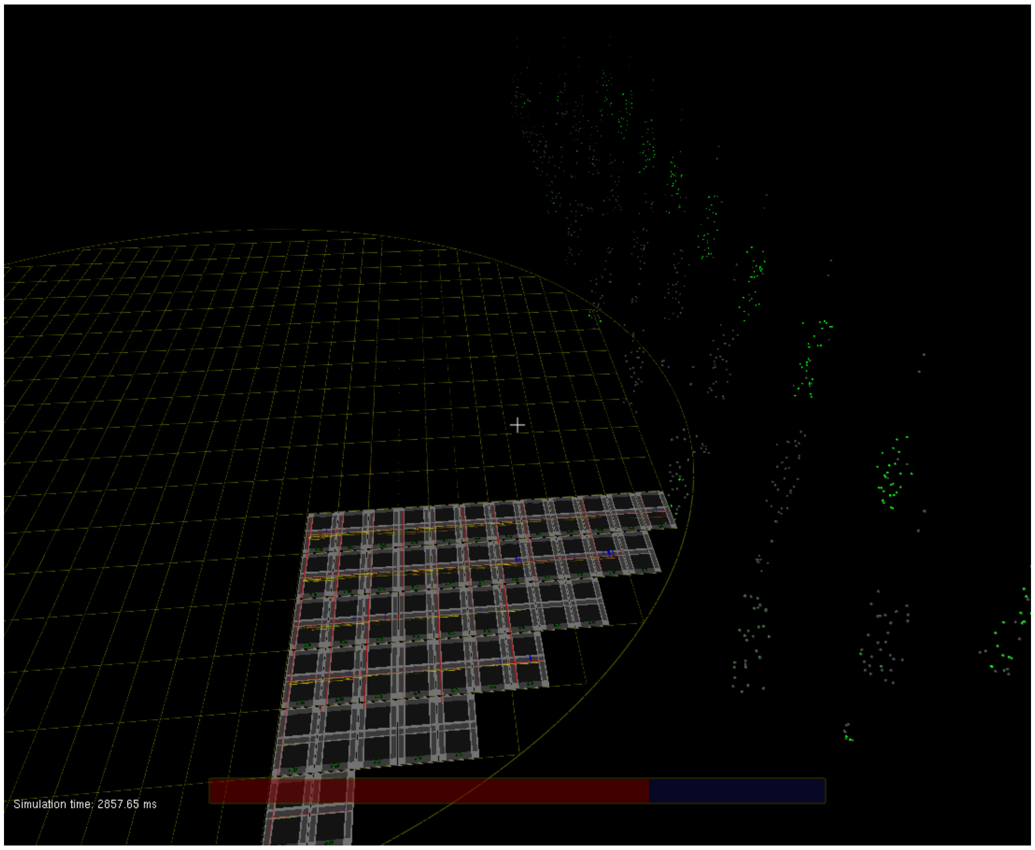


Figure 8: Screenshot of the spiking animation with column 3 begin active

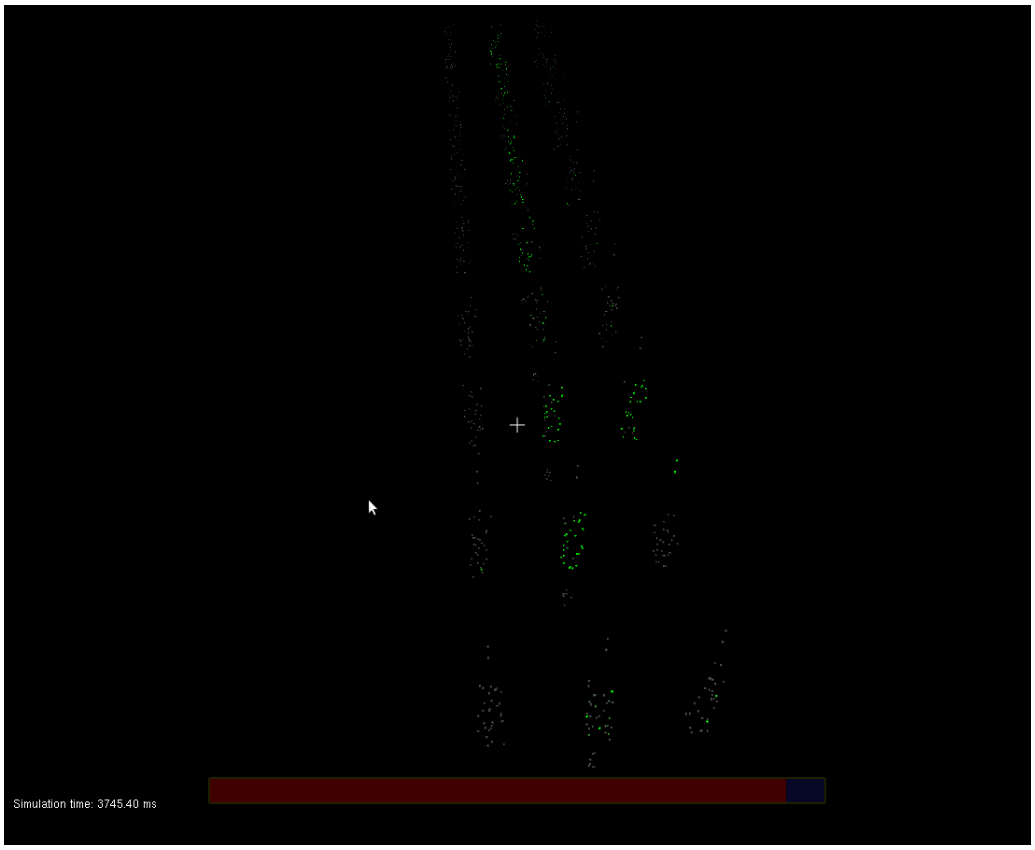


Figure 9: Screenshot of the spiking animation with column 2 begin active and no hardware

- Enable the retrieval of further information about individual routing elements
- Improve performance, e.g. by ommitting details that are not important for the visualization or fix a viewport during animation and only redraw the routing and biological neurons for each frame
- Improve user interface, e.g. Allow selection of any dendrite on the hardware, not only on the focused HICANN
- Implement a timer that makes the animation speed independent of the number of spiking neurons in each timestep
- Implement a batch mode that allows recording and saving of the visualization within the make-flow

The software [3] has been integrated into **git** [2] and is available either from `https://gitviz.kip.uni-heidelberg.de` or from the author (`benjamin.hepp@gmail.com`) or from Daniel Bruederle (`bruederle@kip.uni-heidelberg.de`).

A couple of short movies showing the running software are also available from the author (`benjamin.hepp@gmail.com`) or from Daniel Bruederle (`bruederle@kip.uni-heidelberg.de`).

References

[1] FACETS, *Fast Analog Computing with Emergent Transient States – project website*, `http://www.facets-project.org`, 2009.

[2] git, *Git- Fast Version Control System – website*, `http://git-scm.com`, 2010.

[3] KIP, *gitviz repository*, `https://gitviz.kip.uni-heidelberg.de`, 2010.

[4] OpenGL, *Website*, `http://www.opengl.org`.

[5] PyNN, *A Python package for simulator-independent specification of neuronal network models – website*, `http://www.neuralensemble.org/PyNN`, 2008.

[6] Python, *The Python Programming Language – website*, `http://www.python.org`, 2009.

[7] Radoslav Rusanov, *3D-Visualisierung einer wafer-scale neuromorphen Hardware und des darauf abgebildeten biologischen neuronalen Netzwerkes*, `http://www.kip.uni-heidelberg.de/cms/fileadmin/groups/vision/Downloads/Internship\_Reports/report\_rusanov.pdf`, 2010.

[8] J. Schemmel, D. Brüderle, K. Meier, and B. Ostendorf, *Modeling synaptic plasticity within networks of highly accelerated I&F neurons*, Proceedings of the 2007 IEEE International Symposium on Circuits and Systems (ISCAS'07), IEEE Press, 2007.

[9] J. Schemmel, J. Fieres, and K. Meier, *Wafer-scale integration of analog neural networks*, Proceedings of the 2008 International Joint Conference on Neural Networks (IJCNN), 2008.

[10] Tobias Harion, *3D-Visualisierung einer Abbildung von neuronalen Netzwerkmodellen auf eine neuromorphe Hardware*, `http://www.kip.uni-heidelberg.de/cms/fileadmin/groups/vision/Downloads/Internship\_Reports/report\_harion.pdf`, 2008.

[11] Karsten Wendt, Matthias Ehrlich, and René Schüffny, *A graph theoretical approach for a multistep mapping software for the facets project*, CEA'08: Proceedings of the 2nd WSEAS International Conference on Computer Engineering and Applications (Stevens Point, Wisconsin, USA), World Scientific and Engineering Academy and Society (WSEAS), 2008, pp. 189–194.