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
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 Electronic 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 approach 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 Graphics 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.
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 internship.

### 2.1 Drawing of the Layer 1 Routing

To draw the layer 1 routing for a selected neuron or hardware dendrite there are three separated 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_gmnodetree_node** represents a node of the routing tree. It contains a pointer to a GraphModel::GMNode and a list of children nodes

- **struct simple_gmnodetree** 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
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 den-
drite 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 fo-
cused 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 respectively. The s and f key are used to strafe left or
right respectively. It is also possible to move up or down by pressing the t or g
key respectively. 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.
<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>→</td>
<td>Progress simulation for one timestep</td>
</tr>
<tr>
<td>←</td>
<td>Rewind simulation for one timestep</td>
</tr>
<tr>
<td>↑</td>
<td>Increase simulation timestep by a factor of 2</td>
</tr>
<tr>
<td>↓</td>
<td>Decrease simulation timestep by a factor of 2</td>
</tr>
<tr>
<td>c</td>
<td>Start simulation</td>
</tr>
<tr>
<td>C</td>
<td>Stop simulation</td>
</tr>
<tr>
<td>&lt;SPACE&gt;</td>
<td>Toggle animation</td>
</tr>
<tr>
<td>V</td>
<td>Save viewport</td>
</tr>
<tr>
<td>v</td>
<td>Reload viewport</td>
</tr>
<tr>
<td>&lt;TAB&gt;</td>
<td>Print viewport information</td>
</tr>
<tr>
<td>J</td>
<td>Use original neuron positions</td>
</tr>
<tr>
<td>u</td>
<td>Distribute neurons on a cube</td>
</tr>
<tr>
<td>p</td>
<td>Distribute neurons on a plane</td>
</tr>
<tr>
<td>j</td>
<td>Distribute neurons on a sphere</td>
</tr>
<tr>
<td>k</td>
<td>Display synapses</td>
</tr>
<tr>
<td>h</td>
<td>Display hardware graph</td>
</tr>
<tr>
<td>i</td>
<td>Display mapping</td>
</tr>
<tr>
<td>q</td>
<td>Zoom in</td>
</tr>
<tr>
<td>a</td>
<td>Zoom out</td>
</tr>
<tr>
<td>e</td>
<td>Move forward</td>
</tr>
<tr>
<td>d</td>
<td>Move backward</td>
</tr>
<tr>
<td>s</td>
<td>Strafe left</td>
</tr>
<tr>
<td>f</td>
<td>Strafe right</td>
</tr>
<tr>
<td>t</td>
<td>Move upward</td>
</tr>
<tr>
<td>g</td>
<td>Move downward</td>
</tr>
<tr>
<td>~ or ^</td>
<td>Open terminal</td>
</tr>
<tr>
<td>&lt;ESC&gt; or Q</td>
<td>Quit</td>
</tr>
</tbody>
</table>

### 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:
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 separated 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 timebar 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 respectively. The timestep itself can be increased or decreased by a factor of 2 using the ↑ or ↓ key respectively. During the simulation spiking neurons are drawn in a different color than other neurons and fade back to the non-active color in a biologically 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
Figure 5: Screenshot of the terminal in use
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 omitting 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


