Design and Implementation of a Graphical Visualization Tool for NCS

Justin E. Cardoza[1]
Alexander K. Jones[1]
Denver J. Liu[1]
Roger V. Hoang[1]
Devyani Tanna[1]
Laurence C. Jayet Bray[1,2]
Sergiu M. Dascalu[1]
Frederick C. Harris, Jr.[1]

Brain Computation Lab
http://www.cse.unr.edu/brain/

[1]Computer Science and Engineering
University of Nevada, Reno
Reno, NV, USA

[2]Bioengineering
George Mason University
Fairfax, VA, USA

Abstract

A core goal for neuroscientists is to understand the physiological processes behind memory, learning, and cognition. By developing computational models of the brain, scientists can simulate how neurons interact with each other and study these interactions more closely. However, these simulations often require technical expertise to create and can be difficult to analyze due to a lack of comprehensive visualization tools. NeoCortical View, or NCV for short, aims to solve these problems by providing a simple, convenient graphical interface for managing virtual brain models. NCV interacts with the NeoCortical Simulator (NCS) developed by the Brain Computation Laboratory at the University of Nevada, Reno (UNR), allowing users to build models, distribute simulations across available hardware, and view the current live network state in 3D. NCV provides an intuitive and extensible simulation platform with a reduced learning curve, abstracting away the complex setup and analysis of neural simulations and allowing neuroscientists to focus on neuroscience.

Keywords: Simulation, GPU, Computational Neuroscience

1 Introduction

The primary goal of this software is to simplify the process of analyzing, managing, and launching brain simulations. NCV [5] represents a major breakthrough in how scientists can interact with computational brain models under NCS [2, 3, 8]. The latest version of NCS, version 6, is a CPU/GPU simulation environment for large scale neural networks and systems. It is open-source, parallelizable and runs in quasi-real time. NCS6 supports two major neuron model types: Leaky Integrate-and-Fire (LIF) and Izhikivich. LIF neurons in NCS6 are based on a reordered form of the Hodgkin-Huxley (HH) equations [7]. Izhikevich neurons are modeled using dynamical systems and Izhikevich equations [4]. In addition to these model types, users can create plugin interfaces for other neuron models.

In the past, NCS did not provide any visualization tools. As a matter of fact, no other major neural simulation package provides any 3D visualization of its models. If information was required on a part of a model, scientists had to manually search through text output from the simulator. NCV aims to eliminate the need for this sort of low-level interaction by providing a graphical interface to represent the results of a simulation in a simpler way as they are produced. NCV also provides other tools to help scientists launch and manage simulations more easily. One of these tools is the cluster editor, a section of the interface to aid in specifying what hardware NCS should use. Also of note are the distribution tools used to launch NCS given a hardware configuration from the cluster editor.

Due to its potentially diverse user base and runtime environment, NCV has been built from the ground up with multiple computing platforms in mind. The cross-platform Qt and OpenGL libraries are utilized to ensure that NCV can run under a variety of different systems. NCV has been tested on several computer hardware configurations running the Windows 7 and Ubuntu Linux operating systems.

The rest of this paper is structured as follows: Section 2 outlines the functional and non-functional requirements for the software; Section 3 details the use cases for the application; Section 4 gives an overview of the design and application architecture; Section 5 presents the user interface and visualization capabili-

ties; Section 6 discusses conclusions; Section 7 outlines future plans for additional development.

2 Requirements Specification

The requirements specification for NCV follows the format in Ian Sommerville's book *Software Engineering* [6]. Active behavior which the software must exhibit is described using *functional requirements*; passive constraints on the software or its development are described by *non-functional requirements*.

2.1 Functional Requirements

The functional requirements describe the most important behavior of the software, including user interactions, rendering processes, and network communications:

1. NCV shall be able to communicate with a local or remote installation of NCS.
2. NCV shall support launching simulations of the types supported by NCS.
3. The interface shall give the user options to start, stop, and pause the current simulation.
4. NCV shall display a 3D representation of the neurons and connections in the current simulation.
5. At application exit, NCV shall give the user a choice to either leave the simulation running, or to end it immediately.
6. In the visualization, the virtual camera shall obey user commands to move and rotate.
7. Neurons and connections shall be selectable through the user interface.
8. There shall be an option to deselect any previously selected neurons or connections.
9. Sizes of simulation elements shall be adjustable.
10. Colors used for rendering shall be customizable.
11. State data recieved from NCS shall be passed to the visualization and analysis tools.

2.2 Non-Functional Requirements

The non-functional requirements outline the most important constraints on the system, such as timing and some details of the implementation that needed to be established early on.

1. NCV shall be a cross-platform application.
2. NCV shall be implemented in C++ using Qt and OpenGL.
3. TCP shall be used to communicate with the simulator while it is running.
4. The application shall operate on large datasets.
5. NCV shall perform rendering in as close to simulation time as possible.

3 Use Cases

The use case diagram in Figure 1 shows the most important functionality of the software. Arlow and Neustadt [1] have established a format for software use cases which is used here. The detailed use cases are as follows:

UC01 Initialize Connection
When the user launches an external simulation, NCV attempts to establish a connection with the primary host running the simulation. Once the connection has been established, state updates can be requested from the simulator on an ongoing basis.

UC02 Disconnect
The user can disconnect from the running simulation and halt the analysis tools by interacting with a button in the application's main toolbar.

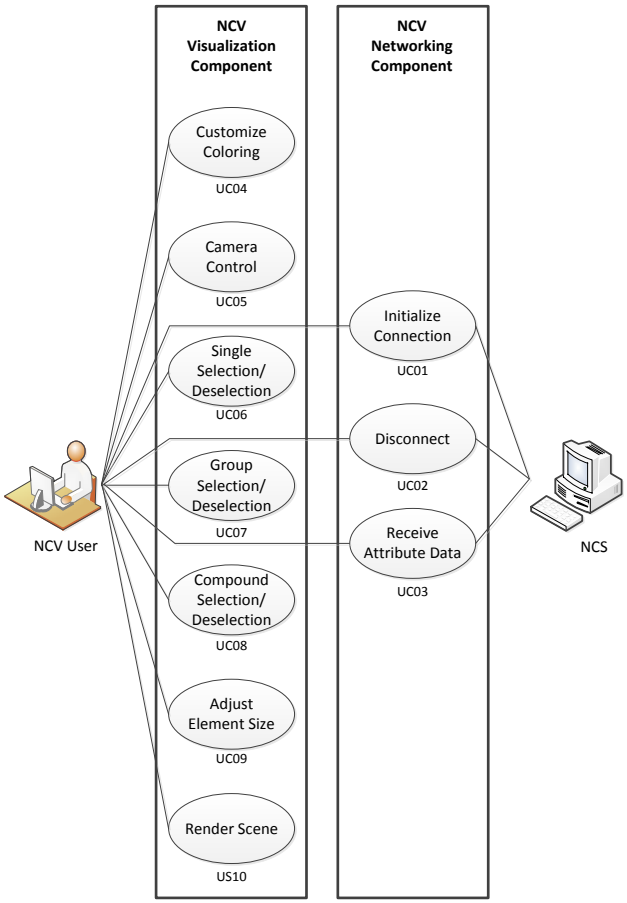

Figure 1: The full use case diagram for the system. Each bubble represents a scenario in which particular behavior is expected of NCV.

UC03 Receive Attribute Data
At each update interval, new data is received from NCS over the network connections for the current attributes. The attribute data is then sent to any active analysis plugins and the GPU, after which the visualization of the model is updated.

UC04 Customize Coloring
The user can choose a custom color configuration for the neurons and connections using an RGB color picker. Several ranges can be selected for different colors and the neuron and connection colors can be either linked or have separate color tables.

UC05 Camera Control
Camera movement includes rotating the viewpoint and navigating through the visualization space. The camera is controlled by both the mouse and the keyboard. With the keyboard, the user can move the camera around the visualization; with the mouse, the user can rotate the camera to point in different directions.

UC06 Single Selection/Deselection
When the user clicks on an element with the mouse, that element is selected and highlighted. If the user clicks on a previously selected element, that element is deselected and no longer appears highlighted.

UC07 Group Selection/Deselection
The user can select multiple elements by supplying a selection rectangle. The selection rectangle is determined by the mouse position when the user first presses the button and the mouse position when the user releases the button. The rectangle is shown on screen to aid in selection. If the user clicks in an empty region of the screen, all selected elements are deselected.

UC08 Compound Selection/Deselection
If there is already at least one element selected, the user can change the selection mode through the tool panel next to the visualization canvas. In compound selection mode, clicking on an element in the visualization adds it to the current selection if it was not previously selected or subtracts it from the current selection if it was previously selected.

UC09 Adjust Element Size
The size of the rendered neurons can be adjusted through the tool panel next to the visualization.

UC10 Render Scene
NCV renders all the neurons and connections using the OpenGL graphics pipeline. Rendering occurs when data is received over the network or when the viewpoint has changed.

4 Design Overview

The core design philosophy of NCV is driven by accessibility. As the application is targeted at professional neuroscientists, it is important to abstract the command line utilities of NCS into a graphical user interface and simplify the process of interacting with a simulation without sacrificing any performance or extensibility. Using the Qt framework gives NCV an efficient and powerful user interface, facilitates deployment on multiple platforms, and supports extensibility through a plugin loading architecture. The latest version of NCV has been tested on Windows 7 and Ubuntu Linux, and while deployment on Macintosh OS X remains untested, the application does not contain any platform specific dependencies that would cause difficulties with installation on such systems.

As indicated by Figure 2, NCV begins by prompting the user for a project directory. The user is then asked to supply a path to an installation of NCS which may be hosted a local machine or a remote one. Once a NCS installation has been validated, NCV passes control to the loaded plugins until a new project is opened or the application is closed. If NCS applications are still running upon application exit, the user can decide whether to end the processes or leave them running.

In order for NCV to provide access to the evolving feature set of NCS, it utilizes plugins to implement the majority of simulation processes. By requiring that plugins satisfy one or more globally declared interfaces, NCV can understand how to interact with the loaded plugins without knowledge of their specific functionality. This enables the application to supply users with a wide variety of capabilities without imposing restrictions on what functionality is contained in the installations of NCS. Currently, there are three interface types plugins may implement. The utility interface allows plugins to interact with NCS applications by supplying a bridge to a NCS installation. The distribution interface also supplies plugins with a bridge to a NCS installation, but is aimed at distributing and launching simulations. When a plugin satisfying the distribution interface launches a simulation, it notifies the main application and supplies a bare-bones description of the simulation context. Lastly, the analysis interface enables plugins to subscribe to data being reported by the simulation so that these plugins may provide users with in-depth analysis of the running simulation.

The largest and most powerful plugin currently incorporated into NCV is the visualization widget which interacts with the main application using the analysis interface. This plugin renders the topology of the running simulation in 3D and enables the user to navigate the visualization with a mouse and keyboard. As shown in Figure 3, the visualization plugin captures

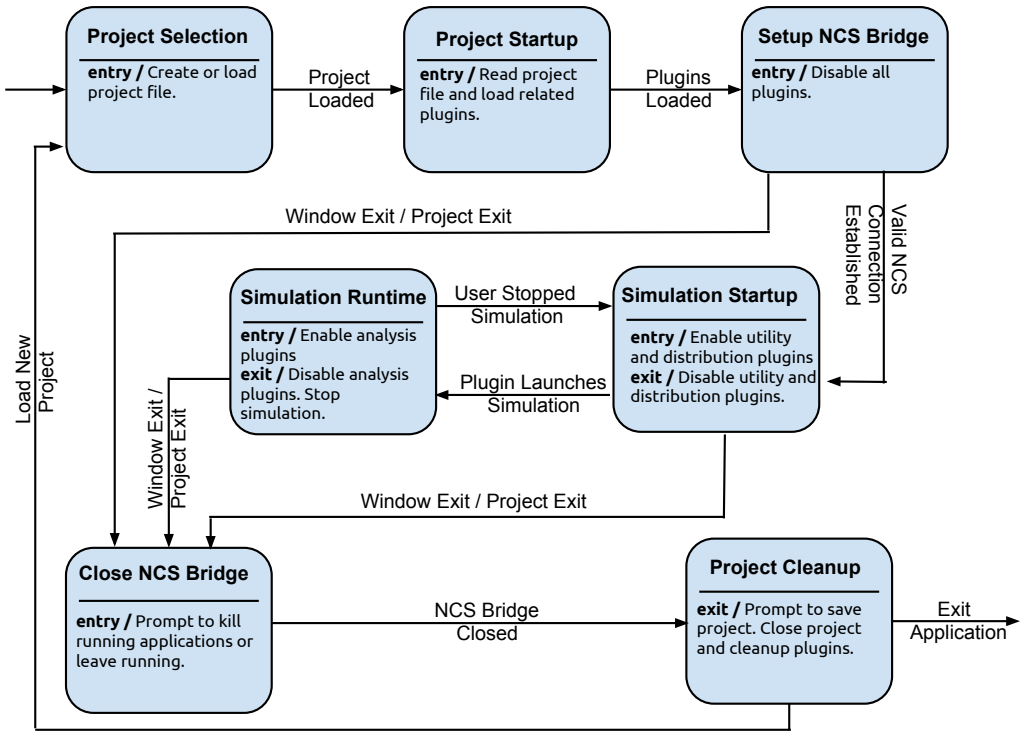

Figure 2: A state diagram that describes the operation of NCV.

data recieved by the network interface, transfers this data to buffer memory on the GPU, and gathers information from the user interface to decide how the simulation topology is rendered. To incorporate the data recieved from the simulation, the visualization maps the received data to a buffer of user specified color values which is sampled for each element during rendering. This allows a large collection of data to be represented intuitively and also allows users to specify custom color configurations that best suit their unique needs. Much like the design of the core application, the visualization plugin is developed with portability and efficiency in mind. It uses the OpenGL 3.3 specification to offload graphics computation to the GPU and utilizes many components of the modern OpenGL feature set to optimize rendering efficiency. As a result, the visualization widget can render simulations in the tens of thousands of neurons with reliable framerates on modest laptops, and render simulations of over 100,000 neurons on more capable desktop computers. On a laptop with an NVIDIA 610M, a model with 1,000 neurons and 10,000 synapses can be displayed at more than 50 FPS. On an i7 desktop with dual NVIDIA 580s, a model with 1,000,000 neurons and 100,000,000 synapses can be displayed at greater than 20 FPS.

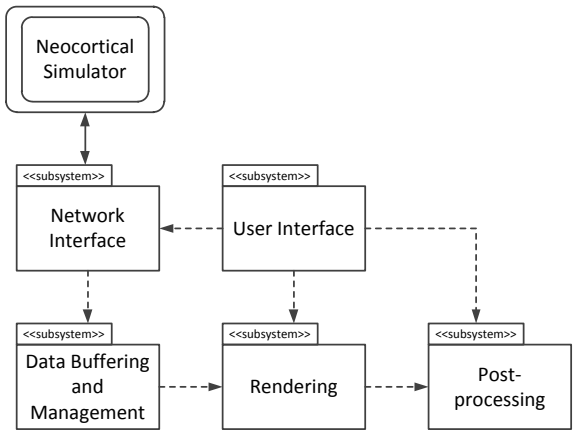

Figure 3: The architecture diagram of NCV's visualization component, showing the main modules and the relationships between them. Also included is the network connection to the external simulator.

5 User Interface

Upon launching the application, the user is prompted to create a new project or load an existing one. When creating a new project, the user must specify a project name and what plugins that project will need. The

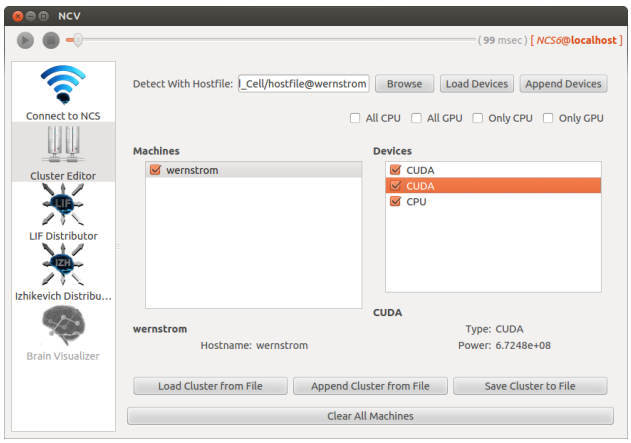

Figure 4: The main window allows users to select the active plugin by interacting with the plugin launcher.

user is then presented with the main application window shown in Figure 4. Docked on the top of the application window is the home toolbar which informs the user of pertinent information about the currently connected NCS installation such as the version and host system. When a simulation is running, this toolbar also lets the user start, pause, and end the running simulation as well as set the interval at which updates from the simulator are received. Present below the toolbar is a widget which contains a plugin launcher and a canvas for displaying the currently selected plugin.

The plugin launcher enables a user to show a plugin's dialog by clicking on the label associated with it. As all plugins require a connection to a built installation of NCS, the plugin launcher contains a built-in widget, indicated by the "Connect to NCS" label, for establishing such connections. The dialog shown after clicking this label validates a local or remote installation of NCS and reports to the main application when a bridge to a valid installation has been established. The main application window then reports information regarding the NCS installation on the home toolbar and activates all plugins in the launcher that require a bridge to NCS. NCV contains several built-in plugins which, given a bridge to NCS, assist in the setup and distribution of simulations. The Cluster Editor plugin allows a user to detect available hardware present among specified machines and generate custom hardware configurations for use in simulation distribution. Simulation launching plugins like the built-in LIF and Izhikevich Launchers allow the user to distribute simulations utilizing specific neuron modeling techniques such as Leaky Integrate-and-Fire or Izhikevich modeling. When a plugin has launched a simulation, it reports to the main application that a simulation is running and the plugin launcher enables all plugins that satisfy the analysis interface so that they may help users analyze the data reported by the simulation.

The built-in visualization plugin is the featured analysis tool provided in NCV. The visualization plugin uses the topology generated by a running simulation to render the current model in 3D. The visualization widget, shown in Figure 5, contains a visualization canvas which houses the rendering context, in addition to a tool panel for customizing the rendering of the brain. The tool panel lets the user select a reported attribute to display for both neurons and connections in addition to providing customization options for how values of these attributes are represented. Attributes with discrete states, such as whether or not a neuron is firing, are given distinct color values for each state; attributes that occupy a continuous range, such as the voltage of a neuron, are mapped to a continuous gradient of color.

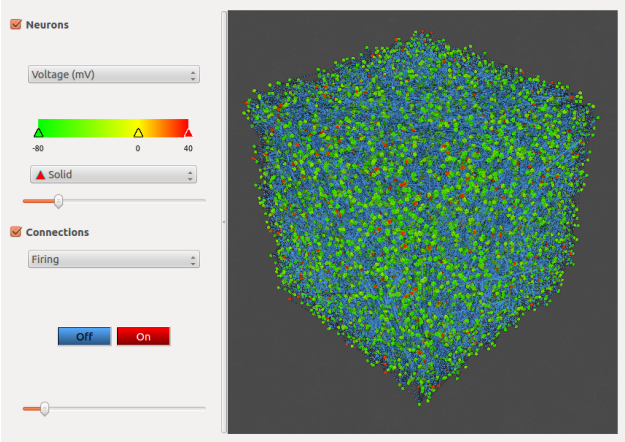

Figure 5: The visualization plugin represents the brain topology in 3D and represents data reported by the simulation through colorization of brain elements.

The visualization canvas renders neurons as cubes and connections between neurons as rectangular pipes. Although this representation is a dramatic simplification of true biological brain structure, it allows large neural networks to be visualized efficently and navigated fluidly. Subsections of the brain can be analyzed by left clicking the mouse on the visualization canvas and dragging a selection rectangle over the desired geometry. Selected brain elements are then visually emphasized over other elements and can be isolated entirely so that only the elements of interest are rendered. As seen in Figure 6, the brain topology can be examined in finer granularity by hovering the mouse over an individual element to examine the current value for all reported attributes.

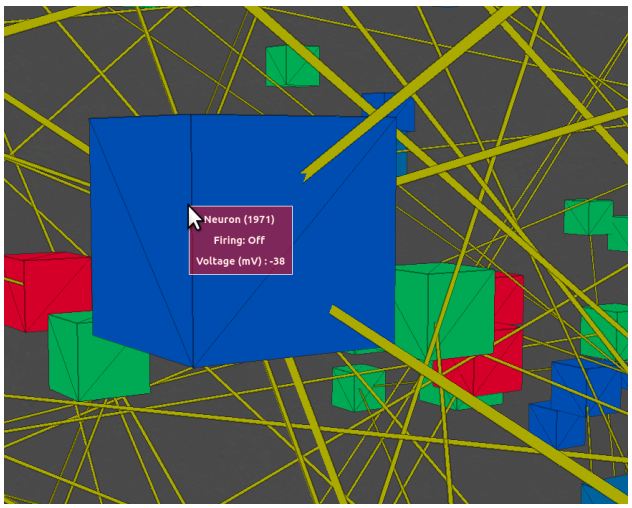

Figure 6: The visualization plugin allows users to examine individual brain elements by hovering over them.

6 Conclusion

NCV provides a wealth of features that make running simulations with NCS easier. By abstracting the powerful facilities of NCS from command line operations into easy-to-use graphical interfaces, it makes NCS accessible to neuroscientists with limited technical expertise. The visualization component allows for powerful analysis of the simulation state in real time, giving users the ability to customize how simulation data is represented. NCV also provides an open source plugin framework which can be used by developers or technically inclined neuroscientists to easily extend NCV to support new features. This framework is fully compatible with the NCS plugin architecture, allowing its plugins to be abstracted for users of NCV. The built-in features of NCV expose the power of NCS, and the plugin framework ensures that further expansions to the simulator can be incorporated.

7 Future Work

Although NCV already contains a great deal of core functionality, there are still many areas in which the application can be expanded. While NCS has defined file formats for brain topology and simulation procedures, NCV could benefit from abstracting their creation into a simple graphical interface. By implementing a drafting plugin, users could visually construct, combine, and populate sections of a brain model in 3D and intuitively describe simulation behavior.

Currently, NCV launches a simulation and immediately connects to it. In the future, additional functionality could be added to NCV to connect to and analyze previously running simulations.

The visualization plugin renders the low level brain topology of a running simulation. One of the most interesting changes would be to subdivide the brain model into groupings. If the visualization plugin could present labeled groupings of neurons and synapses, users could identify the areas of the model that are of interest and focus on them. Additionally, it would reduce the amount of geometry rendered and result in lower computation time.

Acknowledgements

This work was supported in part by a grant from the U.S. Office of Naval Research (N000140110014).

References

[1] J. Arlow and I. Neustadt. *UML 2 and the unified process: practical object-oriented analysis and design*. Addison-Wesley Professional, 2005.

[2] R. Drewes. Brainlab: A toolkit to aid in the design, simulation, and analysis of spiking neural networks with the NCS environment. Master's thesis, University of Nevada, Reno., May 2005.

[3] R. V. Hoang, D. Tanna, L. C. Jayet Bray, S. M. Dascalu, and F. C. Harris, Jr. A Novel CPU/GPU Simulation Environment for Large-Scale Neural Modeling. Submitted, 2013.

[4] E. M. Izhikevich. Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, 14(6):1569–1572., 2003.

[5] A. Jones, J. Cardoza, D. Liu, L. Bray, B. Bryant, S. Dascalu, S. Louis, and F. Harris Jr. A Software Package for Visualizing Complex, Distributed Neural Networks. To Appear, BMC Neuroscience, 2013.

[6] I. Sommerville. *Software Engineering*. Addison-Wesley, Harlow, England, 9 edition, 2010.

[7] T. P. Trappenberg. *Fundamentals of Computational Neuroscience*. Oxford University Press, USA., Second edition, 2010.

[8] C. Wilson, P. Goodman, and F. C. Harris, Jr. Implementation of a biologically realistic parallel neocortical-neural network simulator. In *Proceedings of the Tenth SIAM Conference on Parallel Processing for Scientific Computing*, Portsmouth, VA., March 2001.