

Warning Concerning Copyright Restrictions

The Copyright Law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted materials.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be used for any purpose other than private study, scholarship, or research. If electronic transmission of reserve material is used for purposes in excess of what constitutes "fair use," that user may be liable for copyright infringement.

Warning Concerning Copyright Restrictions

The Copyright Law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted materials.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be used for any purpose other than private study, scholarship, or research. If electronic transmission of reserve material is used for purposes in excess of what constitutes "fair use," that user may be liable for copyright infringement.

University of Nevada, Reno

Design and Implementation of a Neocortical Visualization Tool

A thesis submitted in partial fulfillment of the
requirements for the degree of
Bachelor of Science in Computer Science and the Honors Program

by

Justin E. Cardoza

Dr. Frederick Harris, Thesis Advisor

May 2013

**UNIVERSITY
OF NEVADA
RENO**

THE HONORS PROGRAM

We recommend that the thesis
prepared under our supervision by

Justin E. Cardoza

entitled

Design and Implementation of a Neocortical Visualization Tool

be accepted in partial fulfillment of the
requirements for the degree of

BACHELOR OF SCIENCE

Frederick Harris, Ph.D., Faculty Mentor

Honors Program Representative

Tamara Valentine, Ph.D., Director, **Honors Program**

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 Neurocortical Simulator developed by the Brain Computation Laboratory at UNR, allowing users to build models, distribute simulations across the available hardware, and view the current network state in 3D. It 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.

Acknowledgements

This software project was developed as a team effort by Justin Cardoza, Alexander Jones, Denver Liu, and Paul Bretos in CS 425 *Software Engineering* and CS 426 *Senior Projects in Computer Science*. The group was advised by Dr. Fred Harris, Dr. Sergiu Dascalu, Dr. Laurence Jayet Bray, and Roger Hoang.

I would also like to thank Devyani Tanna and Tor Loken of the UNR Brain Lab for their help at many stages of the project.

Finally, I would like to thank my wonderful family for their constant support, understanding, and encouragement throughout my senior project and my education.

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

Table of Contents

Abstract.....	i
Acknowledgements.....	ii
Table of Contents.....	iii
List of Tables.....	iv
List of Figures.....	v
1 Introduction.....	1
2 Background.....	3
3 Requirements and Use Cases.....	4
3.1 Requirements.....	4
3.1.1 Functional Requirements.....	4
3.1.2 Non-functional Requirements.....	5
3.2 Use Cases.....	6
3.3 Requirement Traceability Matrix.....	9
4 Architecture.....	10
4.1 High Level Architecture.....	10
4.2 Detailed Architecture.....	12
5 Design.....	14
5.1 Network Component.....	14
5.2 Rendering Pipeline.....	16
5.3 User Interface.....	17
6 Results.....	18
6.1 Simulator Connection and Validation.....	19
6.2 Cluster Specification Editor.....	21
6.3 NCS Leaky Integrate-and-Fire Simulation Launcher.....	23
6.4 Izhikevich Simulation Launcher.....	24
6.5 Visualizer Interface.....	25
7 Conclusion and Future Work.....	29
7.1 Conclusion.....	29
7.2 Future Work.....	29
8 Glossary.....	31
Bibliography.....	35

List of Tables

Table 3.1: Level one functional requirements.....	4
Table 3.2: Level two functional requirements.....	5
Table 3.3: Level three functional requirements.....	5
Table 3.4: Non-functional requirements.....	5

List of Figures

Figure 3.1: Requirement Traceability Matrix for NCV.....	9
Figure 4.1: High-level architecture diagram.....	10
Figure 4.2: Detailed architecture diagram.....	12
Figure 5.1: Network component initialization.....	14
Figure 5.2: The attribute update process.....	15
Figure 5.3: Graphics data management.....	16
Figure 5.4: The standard OpenGL programmable pipeline.....	16
Figure 5.5: Simple initial interface design with two toolbars.....	17
Figure 5.6: Updated interface design with more controls.....	17
Figure 6.1: The simulator connection screen.....	19
Figure 6.2: The remote NCS connection interface.....	20
Figure 6.3: The cluster editor interface panel.....	21
Figure 6.4: The LIF-model launching interface.....	23
Figure 6.5: The Izhikevich model launching interface.....	24
Figure 6.6: The visualizer interface showing a running simulation.....	25
Figure 6.7: The visualizer interface with the customization panel showing.....	26
Figure 6.8: Selecting a group of elements by dragging the mouse.....	27
Figure 6.9: The "Selection" section of the customization panel.....	28

Chapter 1 Introduction

The primary overall goal of this project is to provide a means for neuroscientists to intuitively visualize computational brain models. The intended users are, for the most part, neuroscientists with relatively little technical knowledge. However, some computer scientists may also use the software, especially in collaboration with neuroscientists. Neocortical View, or NCV, represents a major breakthrough in how scientists can analyze and interact with computational brain models. Previously, if information was required on any part of a model, the 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.

Because of its potentially varying user base and runtime environment, Neocortical View has been built from the ground up with multiple computing platforms in mind. It has been tested on Microsoft Windows and Ubuntu Linux, and should also run on Intel-based Macintosh computers with minimal changes. The application was written using the cross-platform Qt framework for user interface elements, threading capabilities, and network communications. Qt is also used for communication between the various modules of the application through the signal/slot mechanism. OpenGL 4 is used for portable, high-performance graphics, and the open source libssh library is used for connecting to remote hosts in a computing cluster.

Neocortical View is a significant improvement over the minimal user interaction previously provided by NCS. It provides visual tools for managing NCS simulations in

new intuitive ways, and its capabilities will only grow with time. The Brain Lab hopes to expand the application to include model construction, and eventually, direct manipulation of the simulation as it runs. Although it will require significant changes to the simulator, functionality will also be added to allow the user to connect to an already running simulation. The combination of NCS and NCV is a powerful one, and continued development will ensure that they stay at the forefront of real-time computational neuroscience for years to come.

The rest of this thesis has the following structure: Chapter 2 is an overview of the background for the project, and covers the simulator software as well as some basic concepts from neuroscience. Chapter 3 will go into detail on the requirements of the visualizer application and some use cases that model important interactions between it and other entities. In Chapter 4, the architecture of the application is presented at a high level. A more detailed, but still abstract, design overview is given in Chapter 5. The bulk of Chapter 6 focuses on the user interface, its capabilities, and how the application interacts with the simulator. Chapter 7 discusses future plans for the software and concludes the thesis. Finally, a brief glossary of terms and bibliography follow at the end.

Chapter 2 Background

The Neocortical Simulator, or NCS for short [2], is a software package for simulating parts of the brain in real time. NCS can use both Central Processing Units (CPUs) and the Graphics Processing Units (GPUs) found on modern NVIDIA graphics cards to perform its computations. The use of GPUs allows NCS to run effectively on relatively few computers; whereas most traditional CPU-only simulators would require potentially thousands of machines for a mid-size simulation, NCS can run medium to large simulations on a handful of machines.

NCS simulations consist of virtual neurons, which are analogous to the organic neurons found in a real brain, and the connections between those neurons, as well as any external stimuli to be applied to the neurons. NCS is currently capable of running simulations using the well known Leaky Integrate-and-Fire neuron model or the Izhikevich neuron model introduced by Eugene M. Izhikevich in 2003 [5]. Other neuron models can be added through a robust plugin architecture [4].

Results from NCS are generated using “reports,” which can be one of two types, attached to the attributes of the neurons and connections in the simulation. Until recently, the only report type supported was based on file output: the simulator would write data to a text file, and when it finished running, scientists could then access the data but were on their own when it came to interpreting it. A new report type based on real-time network streams has been introduced this year, which allows other applications to receive updates from the simulator as the simulation state changes. This has been used in the past for experiments where a neural network controls the actions of a virtual robot [1].

Chapter 3 Requirements and Use Cases

3.1 Requirements

Requirements for the application are split into two categories: functional requirements directly describe its behavior, while non-functional requirements describe constraints on the system. The requirements are also sorted by “level”: level 1 requirements are essential to the application, level 2 requirements are desirable but not immediately necessary, and level 3 requirements are mostly plans for future development.

3.1.1 Functional Requirements

The functional requirements for NCV start below with the level one requirements, i.e. the essentials that need to be finished this year:

R01	NCV will visualize neurons in the simulation window.
R02	NCV will visualize axons between the neurons in the simulation window.
R03	NCV will allow the user to navigate through the scene.
R04	NCV will allow the user to select neurons to focus on.
R05	NCV will allow the user to deselect neurons.
R06	NCV will allow the user to adjust the size of model elements.
R07	NCV will allow the user to customize the colors used for rendering.
R08	NCV will allow the user to connect to an external simulation.
R09	NCV will allow the user to disconnect from the server.
R10	NCV will receive data about model structure.
R11	NCV will receive data about neuron firings.
R12	NCV will receive data about neuron voltages.

Table 3.1: Level one functional requirements.

The implementation of these is fairly solid at this point, so development will likely focus on level two and three requirements next:

R13	NCV will allow the user to save, edit, and open existing models.
R14	NCV will allow the user to take screen captures after the simulation is over.
R15	NCV will allow the user to zoom in and out on different anatomical levels.
R16	NCV will show users the speed and delay of connections.
R17	NCV will display different colors for synaptic strengths.
R18	NCV will report voltage, current, and fire count.
R19	NCV will allow users to make anatomical changes on existing models.
R20	NCV will allow users to build models from scratch.

Table 3.2: Level two functional requirements.

R21	NCV will be able to connect to a live simulation.
R22	NCV will allow user to capture the screen during a simulation.
R23	NCV will allow users to focus on column, layers, or specific groups of neurons.

Table 3.3: Level three functional requirements.

3.1.2 Non-functional Requirements

There will, of course, also be constraints on how NCV can fulfill these requirements. Some of the constraints come in the form of non-functional requirements:

N01	NCV will be a cross-platform application.
N02	NCV will be implemented using Qt and OpenGL 4.0 in C++.
N03	NCV will use TCP to facilitate network connections.
N04	NCV will operate on large datasets.
N05	NCV will effectively partition neuron data into separate regions for selection.
N06	NCV will render as close to simulation time as possible.

Table 3.4: Non-functional requirements.

3.2 Use Cases

UC01 - Initialize Connection: Upon starting the application, the user will be prompted to initiate a connection to the external simulation. The user will be asked to enter critical information regarding the communication between the simulation and visualization such as port number and Job ID. These fields can be saved for later use if specified by the user.

UC02 - Reconnect: If the user previously disconnected the visualization from the currently running simulation, the connection can be reestablished via the GUI.

UC03 - Disconnect: If for any reason the user would like to disconnect from the running simulation and halt updating the visualization, the user can do so by clicking a button in the GUI.

UC04 - Receive Firings: Once the initial update parameters have been set up for neuron firings, update packets will be received on a previously determined network port. Each packet will contain information for some of the neurons in the simulation as determined by the networking protocol. As each packet is received, the firing data will be copied into memory used by the graphical portion of the application, and the next time the screen is updated, the neurons that have fired since the last update will be highlighted.

UC05 - Receive Structure: After initiating a network request to read the structure of the simulation, the structure will be received over that same network connection. The application will then replace whatever structure is currently being displayed with the new data, and that visualization will show up on screen.

UC06 - Receive Voltages: If voltage updates are currently being received, each voltage update packet will contain floating-point voltage values for some of the neurons.

These voltage values will be copied into memory used by the graphics code, and the next time the screen is updated, the new voltage values will be displayed in the form of different colors for the neurons if the appropriate display mode is set.

UC07 - Save Model: The user will have the ability to save the model currently being analyzed so that it may be loaded for future sessions if desired.

UC08 - Load Model: The user can load any previously saved model to the visualizer so that it may continue to be rendered and analyzed.

UC09 - Create Model: There will be an option to create a model from scratch. The user can populate all the attributes so that test cases can be created and reviewed as a way to predict certain events or to gain knowledge about an area that may not currently be available.

UC10 - Modify Model: A previously saved or created model can be modified so that the user can input custom attributes and parameters to be rendered.

UC11 - Create Screenshot: Users will be able to take screenshots of the current session and save them for future review and analysis.

UC12 - Customize Coloring: The user can choose a custom color configuration for the neurons and synapses using an RGB color scale. Several ranges can be selected for different colors and the neuron and synapse colors can be either linked or have their own color table.

UC13 - Camera Control: Camera movement includes rotating the viewpoint and navigating through the scene. The camera can be controlled by both the mouse and keyboard. By using the keyboard, the user can navigate through the 3D environment and with the mouse, the user can rotate the camera on a 3D axis.

UC14 - Single Selection/Deselection: When the user clicks on a neuron using the left mouse button, that neuron is selected and highlighted, and its information is presented in the right side of the screen. When the user clicks again on a previously selected neuron, or clicks in an empty region, that neuron is deselected, it is no longer highlighted, and its detailed information is hidden.

UC15 - Group Selection/Deselection: The user can select multiple neurons by supplying a selection rectangle. The selection rectangle will be determined from the position of the mouse when the user first clicked the along with the position of the mouse when the user released the button. The rectangle will be shown on screen to aid in selection. If the user clicks in a free space on the screen, all selected neurons will be deselected.

UC16 - Successive Selection/Deselection: If there is already at least one neuron selected, the user can select more neurons by clicking on them and holding the Shift key. If there is already at least one neuron selected, the user can deselect one neuron at a time by clicking on them and holding the Shift key.

UC17 – Scroll Zoom: The size of the rendered neurons can be enlarged (scroll down on the mouse wheel) or shrunk down (scroll up on the mouse wheel).

UC18 - Render Scene: NCV will render all the neurons and connections using the OpenGL graphics pipeline. Rendering will happen when data is downloaded from the network, or when a user has loaded, modified, or created their own models to render.

3.3 Requirement Traceability Matrix

		USE CASES																	
		UC01	UC02	UC03	UC04	UC05	UC06	UC07	UC08	UC09	UC10	UC11	UC12	UC13	UC14	UC15	UC16	UC17	UC18
REQUIREMENTS	R01																		X
	R02																		X
	R03													X					
	R04														X	X	X		
	R05														X	X	X		
	R06																	X	
	R07												X						
	R08	X																	
	R09			X															
	R10	X	X			X													
	R11	X	X		X														
	R12	X	X				X												
	R13							X	X	X	X								
	R14											X							
	R15																	X	
	R16																		X
	R17												X						
	R18																		X
	R19									X									
	R20									X									
	R21	X	X																
	R22											X							
	R23	X	X																X
	R24														X	X	X	X	

Figure 3.1: Requirement Traceability Matrix for NCV.

Chapter 4 Architecture

4.1 High Level Architecture

At a high level, Neocortical View is broken up into three main modules: the user interface, the rendering pipeline, and the network component.

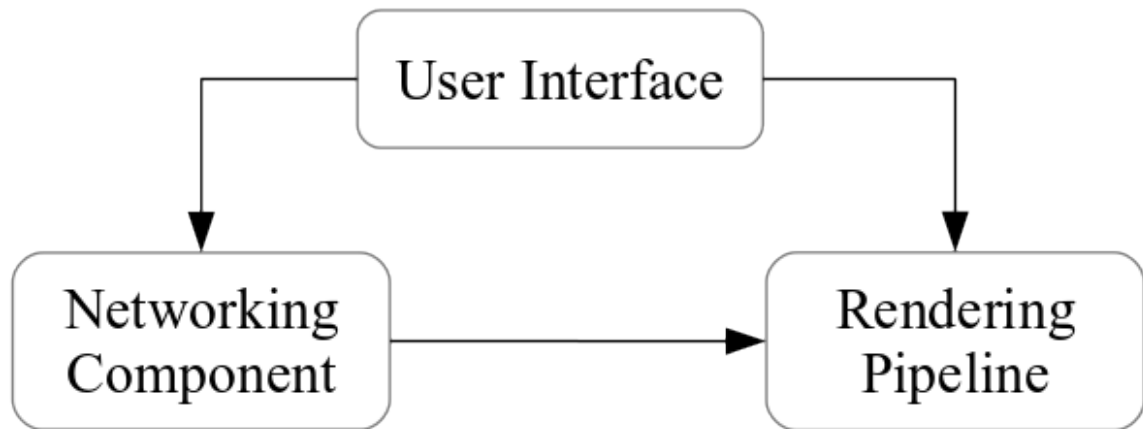


Figure 4.1: High-level architecture diagram.

The networking component is responsible for communicating with the simulator. It handles launching remote simulations and receiving state update information as the simulation runs. Even a locally run simulation will use a network socket to pass information back to NCV, but this type of connection will be handled efficiently under most operating systems by using a local pipe or other IPC method to bypass the network hardware entirely.

The rendering pipeline manages the current data set being rendered, as well as keeping track of any custom rendering parameters the user has set. This module also performs basic tasks to manage the point of view and mediates between the Central Processing Unit and the graphics card, ensuring that the latest information is always

displayed on screen.

The user interface allows the user to start and stop simulations, control the parameters of the current simulation, and interact with the simulator and associated tools in various ways. It is the portion of the application that is most visible and directly accessible to the user, and it gives the user the ability to control aspects of the other modules' behaviors.

4.2 Detailed Architecture

A more detailed architecture diagram also takes into account the interconnect between the networking module and the rendering pipeline, as well as breaking down the graphics functionality into a more realistic two-part process:

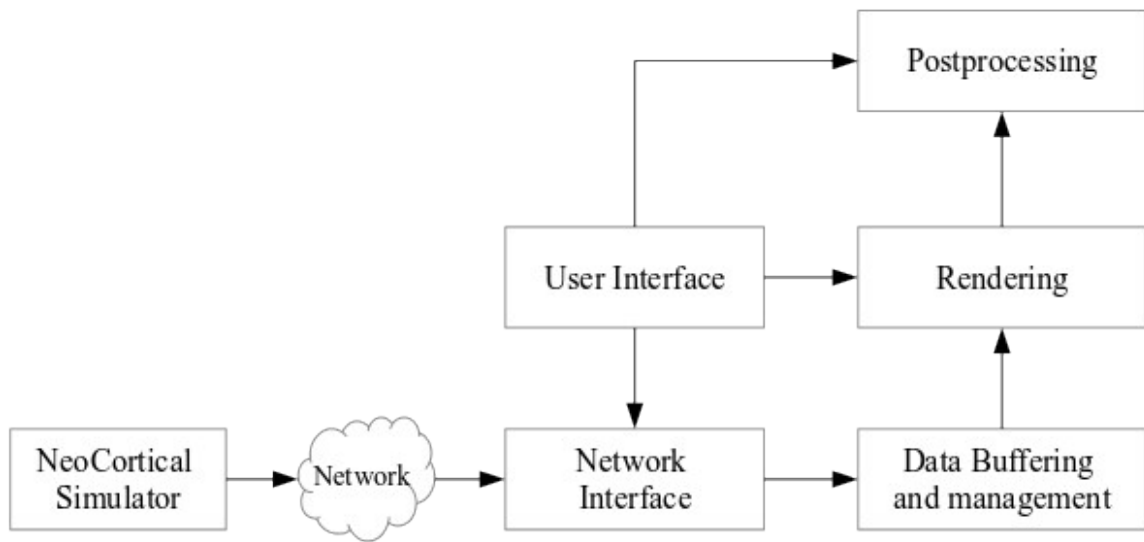


Figure 4.2: Detailed architecture diagram.

The main differences are the separation of the rendering and postprocessing stages in the graphics pipeline and the addition of an intermediate data management layer. In the rendering module, the scene geometry is specified and tessellated as needed before being passed on to the postprocessing module, which affects the final appearance of each object and filters geometry on a pixel-by-pixel basis.

Bridging the gap between the networking and graphics modules, the data buffering and management layer organizes the incoming information from the network into a format that is useful for displaying to the user. This includes making sure that the indices used by the simulator are mapped to the optimized indices used by the rendering

code on the graphics card.

Finally, it is good to note that this more detailed view of the system architecture also includes a view of the external source of data used by NCV, namely, the simulator. NCS is represented by another block; not much detail is required since it is outside of the application being analyzed. It is connected to the networking module through an abstract computer network which is represented by the standard “cloud” icon, signifying that the exact implementation of the network is unimportant.

Chapter 5 Design

After the high-level architecture of the application was decided, designs were created to describe the behavior and organization of the various components, as well as the internal interfaces that tie them together.

5.1 Network Component

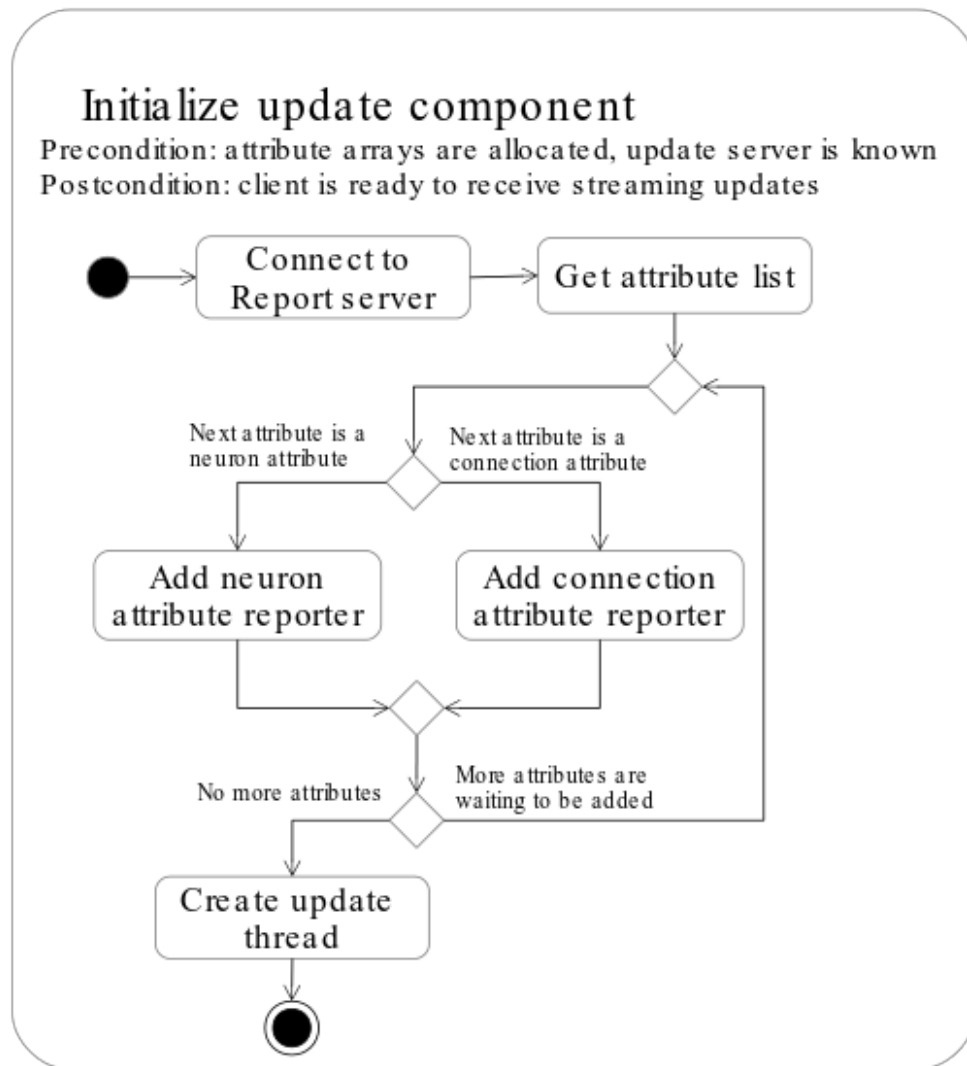


Figure 5.1: Network component initialization.

This UML activity diagram details the initialization process performed when the

network component is started. It automatically enumerates all of the attributes that the rendering pipeline will need access to and requests those attributes from the simulator. Once the simulator is connected, attributes can be updated asynchronously as described by this activity diagram:

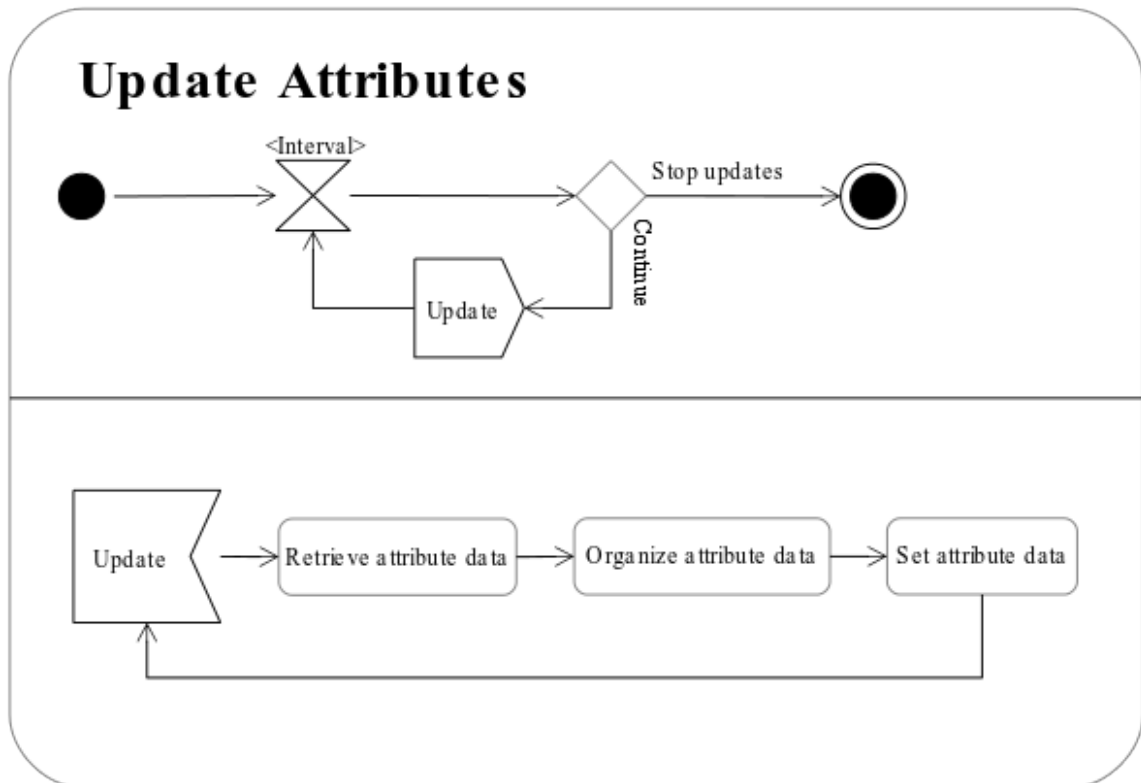


Figure 5.2: The attribute update process.

This two-part process is used to ensure thread safety, since the attribute update loop runs in a dedicated thread to avoid blocking the user interface. Qt provides safe thread-to-thread communication through signals and slots as long as the receiving thread has an event loop. The update interval can be set by sending a signal to another separate slot in the network update manager.

5.2 Rendering Pipeline

The rendering pipeline is highly optimized for performance with large numbers of objects. To minimize data transfers, it stores geometry data in fixed memory buffers on the graphics card, and it stores attribute data in other buffers more suited for quickly changing values. It also uses a technique known as *instancing* to reduce CPU load. Instancing allows the program to send one command to draw a group of objects, rather than sending a separate command for each object.

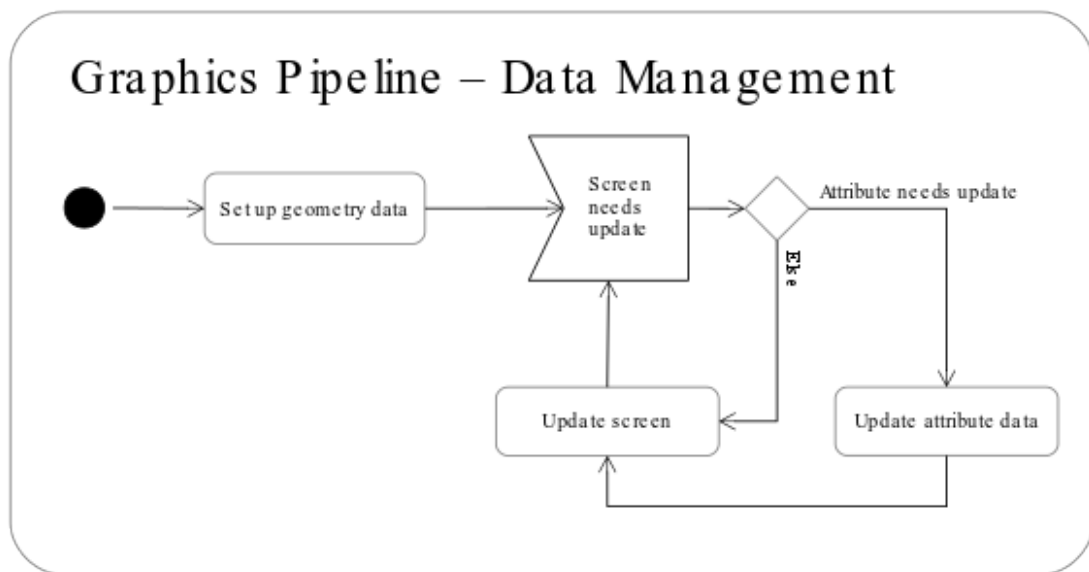


Figure 5.3: Graphics data management.

Once the required data is available on the graphics card and a display update has been requested, the standard OpenGL graphics process on the graphics card takes over.

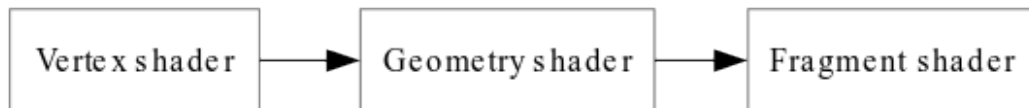


Figure 5.4: The standard OpenGL programmable pipeline.

5.3 User Interface

NCV's user interface has evolved greatly since the beginning of the project. Initial designs were very simple, and the interface was gradually refined as new requirements became apparent.

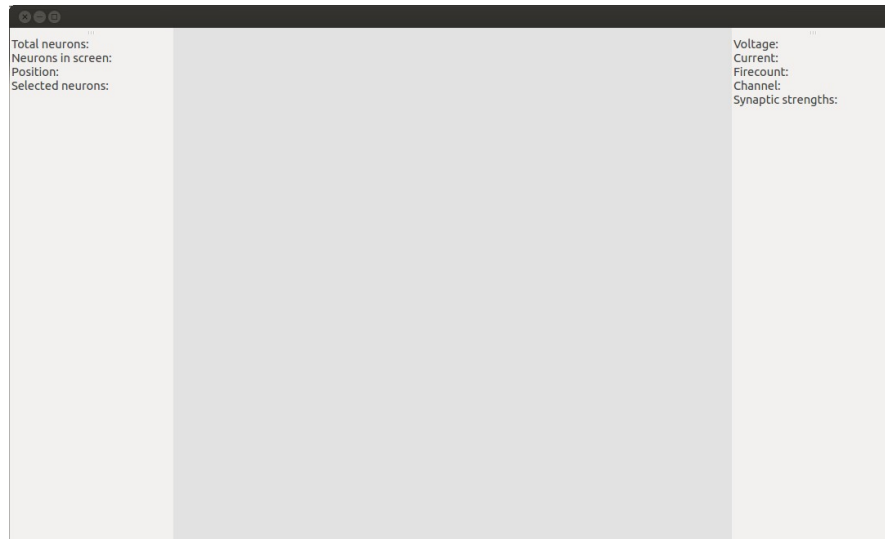


Figure 5.5: Simple initial interface design with two toolbars.

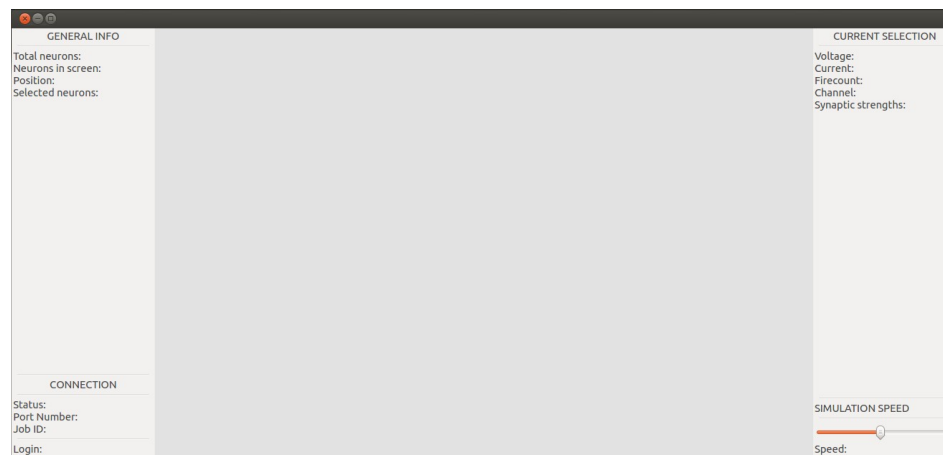


Figure 5.6: Updated interface design with more controls.

The current advanced user interface implemented in the final version will be discussed in greater detail in the next section.

Chapter 6 Results

Neocortical View can currently run a simulation on the local machine's CPU, connect to it, and update the display with the state of the simulation as it runs. It provides an intuitive graphical user interface for performing various tasks such as simulation launching, verification of simulator files, cluster specification editing, and visual simulation analysis. The user interface is modular and extensible, and plugins loaded at runtime can add new panels to the toolbar as needed in order to give easy access to their own custom interfaces.

The next few pages will detail the appearance, features, functionality, and uses of the project. The focus is on the user interface since that is the core aspect of the application, but details are also given on internal functionality which affects how the user interacts with it.

6.1 Simulator Connection and Validation

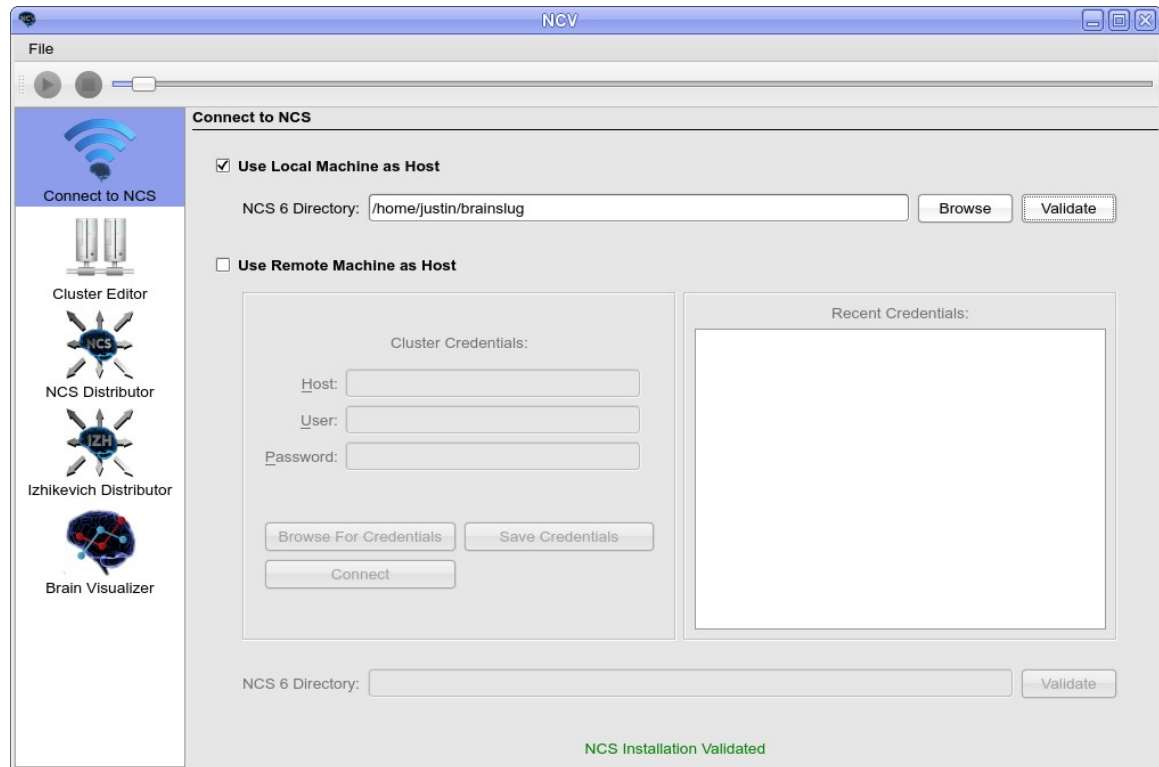


Figure 6.1: The simulator connection screen.

The first step in launching a simulation is to connect to and validate a copy of the Neocortical Simulator using the interface shown above. A local installation of NCS can be used for small models, in which case the simpler portion of the interface can quickly be used in just a few clicks.

For larger models that require a computing cluster to simulate, the more complex remote interface must be used. This requires the user to log in using SSH credentials, connect to the remote machine, find where NCS is stored on that machine, and then validate it.

The following example shows how the remote connection interface might be used to connect to a cluster through a main host machine using manually entered credentials:

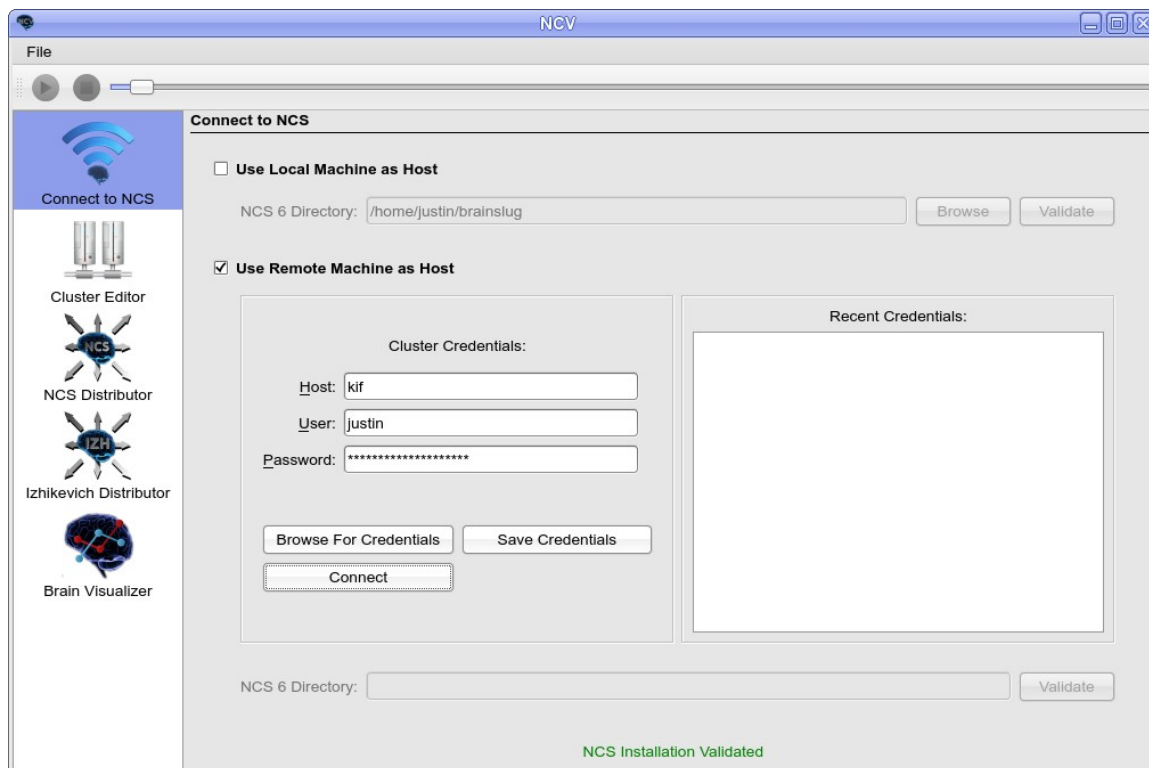


Figure 6.2: The remote NCS connection interface.

The host name should be the “master machine” in the cluster, responsible for distributing the simulation to the others and managing its execution. The user will need to have a user account on each machine in the cluster, but for validating the NCS installation, only an account on the master machine is required. Credentials can be entered manually or loaded from a file, and the interface gives the option to save the current credentials for later use.

6.2 Cluster Specification Editor

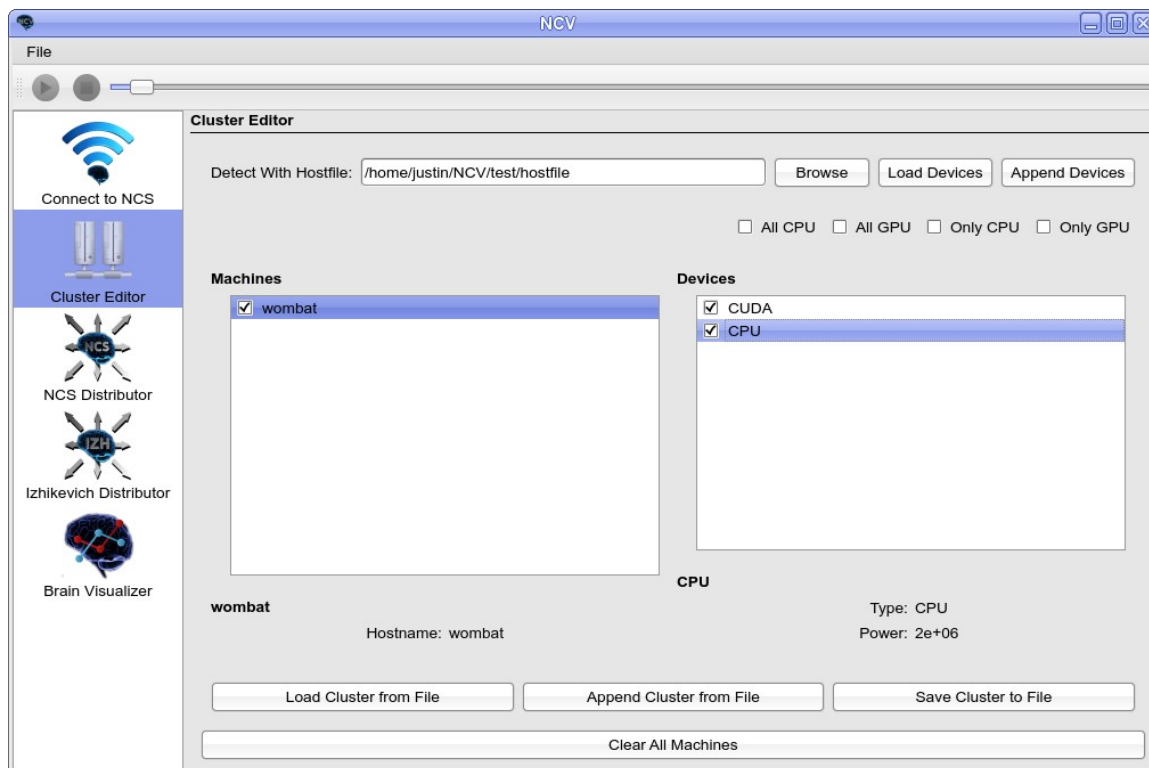


Figure 6.3: The cluster editor interface panel.

NCS needs a cluster specification file to tell it what hardware it should use for its computations. If the user does not already have a cluster specification file, they can use this interface to determine where the simulator should run. It uses a “hostfile,” a text file containing a list of hostnames, to scan the network for other computers; once the other computers are found, the interface can list the available hardware for each one. The hardware types currently supported are the most common in standard PC hardware: CPUs, or Central Processing Units, and CUDA devices, which are graphics cards that can perform general-purpose computations.

After the user has selected the appropriate hardware for NCS to use, they can save the cluster specification file for use in launching a simulation. It is also possible to load

an existing cluster specification file for editing and then save the modified version once the necessary changes have been made.

For editing purposes, the interface provides a checkbox for each computer and each piece of hardware. A computer can be added to or removed from the simulation by checking or unchecking its checkbox in the left pane, and similarly, an individual hardware device can be added to or removed from the simulation by checking or unchecking its checkbox in the right pane. All hardware of a certain type can be added or removed using the checkboxes above the two main panes. An estimate of the total computational power of a hardware device is displayed below when its entry in the right pane is selected.

6.3 NCS Leaky Integrate-and-Fire Simulation Launcher

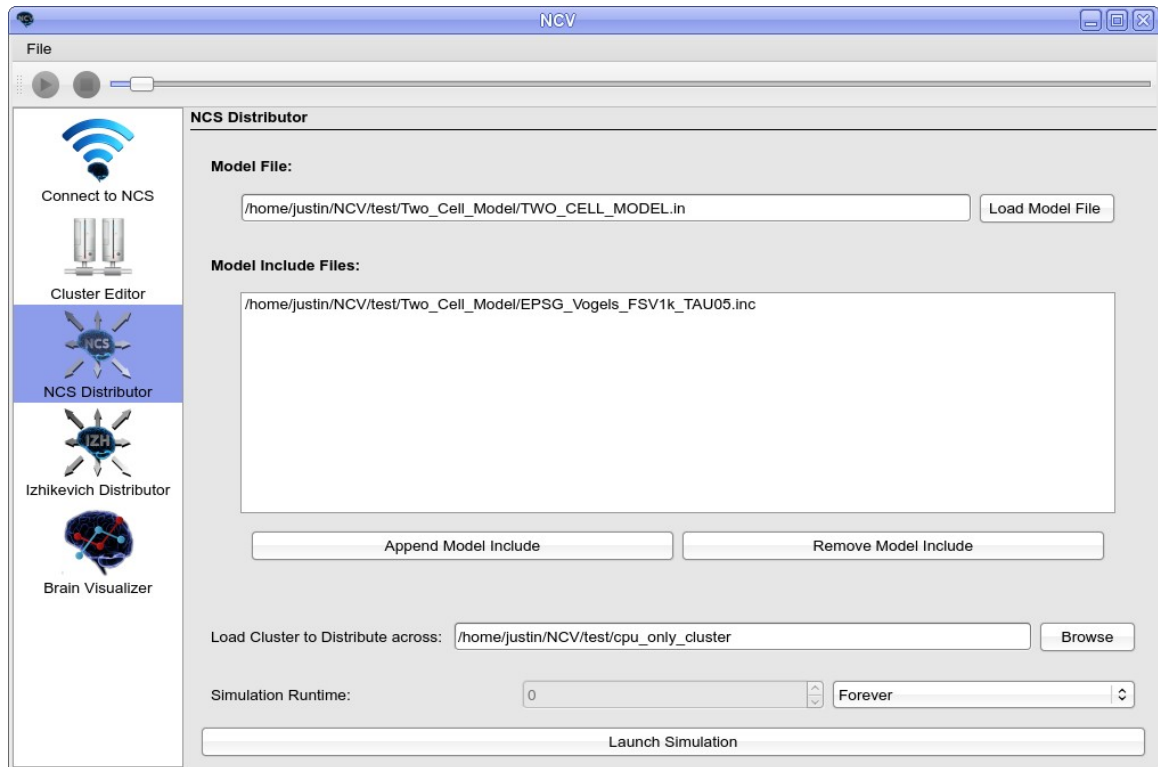


Figure 6.4: The LIF-model launching interface.

To launch a simulation using the NCS-specific implementation of Leaky Integrate-and-Fire neurons, Neocortical View provides the above interface. The user must choose a model file, any include files the model depends on, and a cluster specification file before launching the simulation. If a cluster specification file is not available, one can be generated using the cluster editor interface which is also part of Neocortical View.

Finally, before clicking the “Launch Simulation” button, the user must specify a duration for the simulation. The duration can be in milliseconds, seconds, minutes, hours, or days, or the user can specify that the simulation should run “forever” (until it is stopped).

6.4 Izhikevich Simulation Launcher

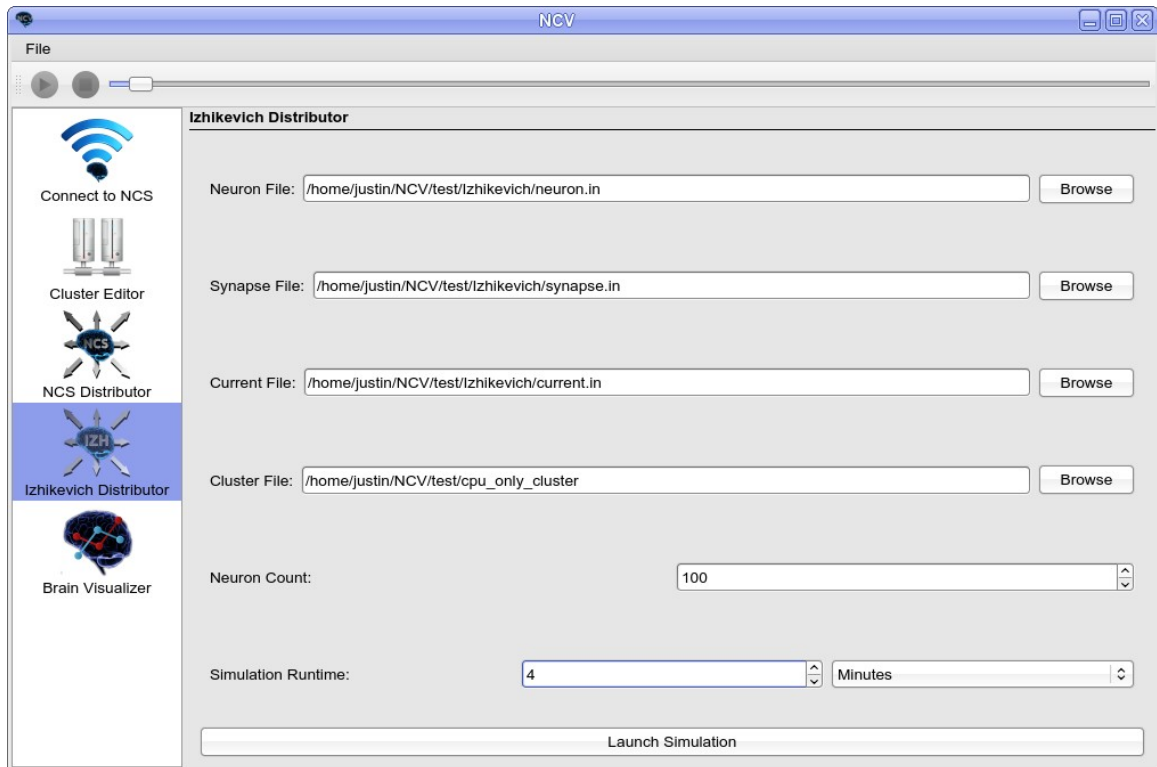


Figure 6.5: The Izhikevich model launching interface.

To run a simulation based on Izhikevich neurons, more input files are required: a separate file is required for the neurons, the synapses, and the input currents. As with Leaky Integrate-and-Fire simulations, a cluster specification file is also required to determine what hardware NCS will run its computations on. A simulation duration is also required before the simulation can be launched.

6.5 Visualizer Interface

As the view into the actual visualization of the model, the visualizer interface is perhaps the most important part of the application that the user will interact with.

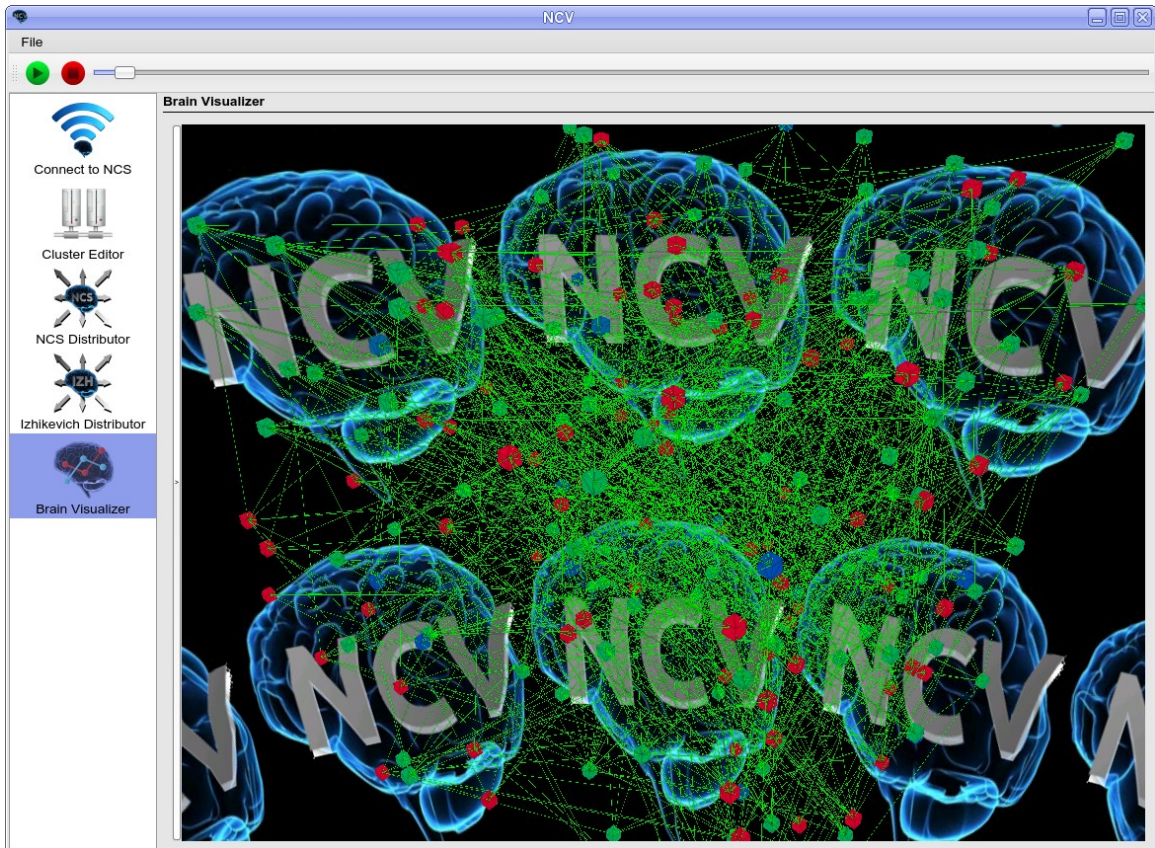


Figure 6.6: The visualizer interface showing a running simulation.

This panel of the application displays the topology of the currently running simulation, updates the colors of the neurons and connections based on their attributes and the ranges selected by the user, and gives options for customizing the appearance of the visualization. In the image above, the neurons which are currently firing are shown in a different color than the ones that are not, and the neurons that are about to fire are shown in another range of colors based on the voltage level within each cell.

If needed, a second side panel can be shown to offer further controls that the user

can use to customize how the model is rendered on screen. When the customization panel is not needed, it can easily be hidden, and if it is needed again, it can easily be brought back.

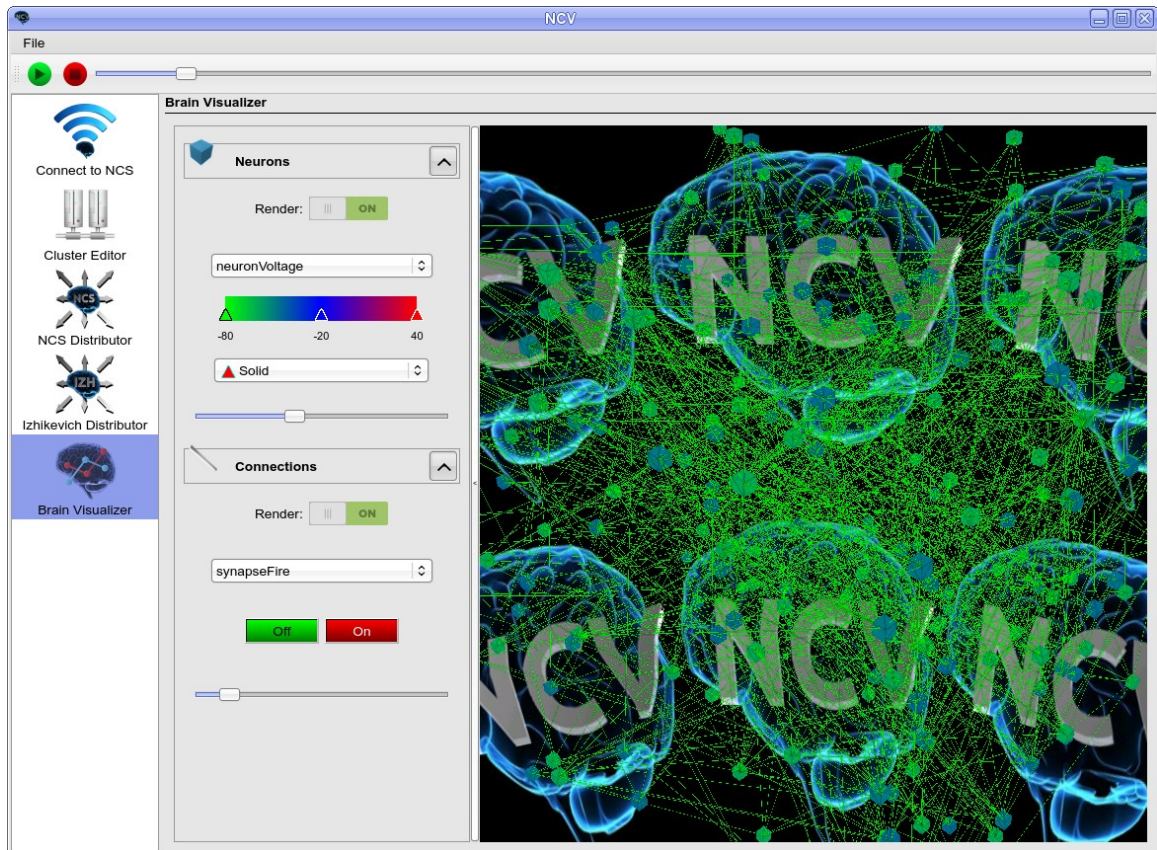


Figure 6.7: The visualizer interface with the customization panel showing.

From the customization panel, the user can select which attributes to display, change the color gradient ranges, and determine whether to render neurons, connections, both, or neither. The sizes of both element types on screen are also configurable by sliding controls.

Another section of the customization panel, the “Selection” section, only appears if there are elements of the simulation currently selected. Selecting elements is fairly straightforward: by clicking on a neuron or connection with the mouse, the user can

select a single element, and by clicking and dragging, the user can select a group all at once.

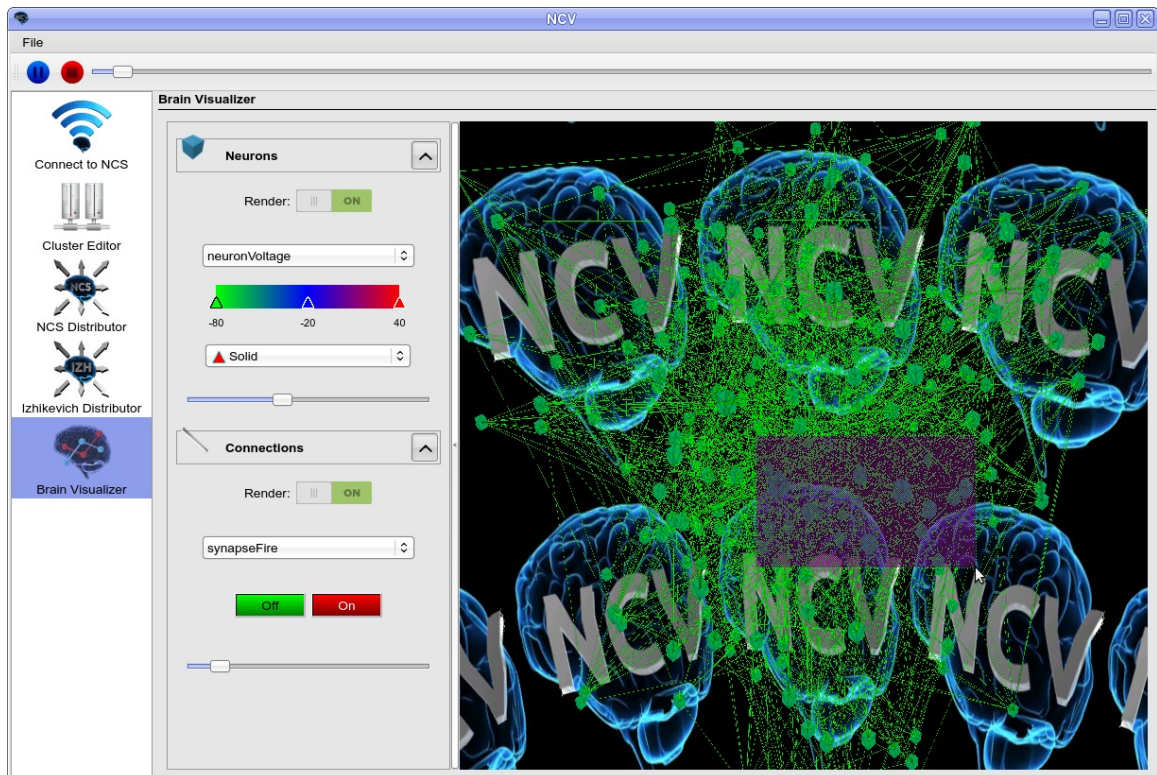


Figure 6.8: Selecting a group of elements by dragging the mouse.

While the user is dragging the mouse to select elements, a translucent rectangle is drawn over the visualization to show where the selection will take place. When the user releases the mouse cursor, the rectangle disappears and the selected elements are highlighted. In addition, the section labeled “Selection” on the customization panel appears, and the user can interact with it to further customize how the current selection is displayed and treated.

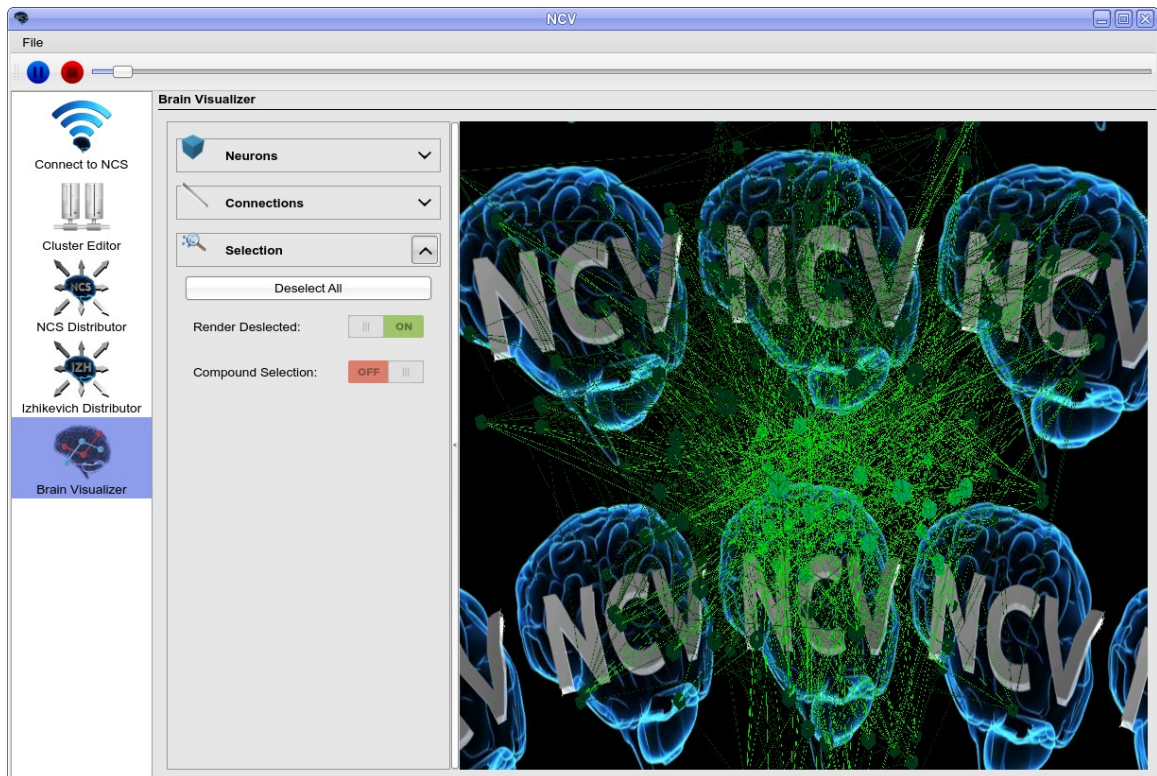


Figure 6.9: The "Selection" section of the customization panel.

Using the special “Selection” section of the customization panel, the user can deselect all, turn on and off rendering of unselected elements, or enable compound selection mode. The “Deselect All” button clears the current selection and returns the display to the normal view. Turning off the rendering of unselected elements allows the user to only see what they have selected, giving them the opportunity to focus on one part of the simulation. Finally, compound selection mode allows the user to add and subtract elements or groups from the current selection so that more complex-shaped groups than just rectangles can be selected at one time.

Chapter 7 Conclusion and Future Work

7.1 Conclusion

In its first year alone, this project has grown into a viable alternative to the analysis methods previously used for NCS simulations. Although there are still many places where additional functionality would greatly benefit users, NCV is an excellent start so far to bridging the gap between people and computers in computational neuroscience. The interface it provides already goes a long way toward providing a comprehensive simulation analysis and manipulation suite and its capabilities will grow with time to become even more useful.

Neocortical View represents a shift in how neuroscientists analyze and interact with computational brain simulations. It literally changes the way people look at computer-based brain models, and that's a good thing. By reducing the need for tedious, technically involved, low-level interpretation of raw simulation data, it lets scientists focus on the problems they are really trying to solve.

7.2 Future Work

Neocortical View will, of course, keep expanding after this year. Some of the plans for future work include adding the ability to connect to already running simulations, support for more simulation types, and new options for manipulating the simulation itself as it runs.

Having the ability to connect to running simulations will greatly increase the versatility of Neocortical View and make it more useful in certain situations. One particularly useful scenario is where a large, long-running simulation is maintained on a

server or cluster in a laboratory on campus, and scientists can connect to it from various locations to check on its current state and progress. This feature will be added at some point due to the resulting gain in utility, but it will require some restructuring of the simulator to ensure the correct data about each simulation is retained and transmitted to NCV when needed.

Support for additional simulation types will be useful as neuroscientists push the boundaries of existing computational neuron models. Researchers at the UNR Brain Lab are currently working to integrate other third-party simulators with NCS, which would significantly increase the possibilities for different simulation types. NCV's plugin architecture should make adding this fresh functionality a much easier process.

One manipulation method that is of particular interest is known as “lesioning,” the process of severing connections between neurons while the simulation is running. Scientists will then observe how the behavior of the rest of the model changes. This would require significant restructuring of both NCS and NCV.

In addition to improvements to the software itself, written publications are a significant part of the future work planned for this year. A paper proposal detailing the user interface has been accepted to CNS 2013 taking place in Paris, France in July [3]. At least one other conference paper is planned for later in the year.

Chapter 8 Glossary

Axon – A fiber-like projection from a neuron that conducts signals away from it toward other cells.

Culling - The process of removing objects that cannot be seen. These objects can either be outside of the current view or occluded by a nearer object.

Dendrite – A projection of a neuron that conducts signals from other cells toward it.

Ethernet - A family of networking hardware technologies which are used as the backbone of local area networks. Common varieties of Ethernet enable data transfer of up to a gigabit per second, or one billion bits every second.

Fragment – A unit of data that describes a “potential pixel”; all the information required to generate a pixel is included within it, as well as other metadata to inform the decision of whether or not to generate a pixel.

Graphical User Interface (GUI) – A method of interacting with software that involves visual cues and metaphors. GUIs have often historically relied heavily on mouse input, but most also use some sort of keyboard for text entry.

Instancing - A graphical process for rendering multiple objects in a scene from a single set of geometry. This geometry is duplicated across the scene with different attributes (color, size, position, etc.) to describe a large scene with minimal overhead.

Interpolation – The estimation of data values that would lie in between known values. This is often used to sample from a discrete data set as if it were continuous.

Local Area Network (LAN) - A computer network that covers a relatively small geographic area (e.g. a single office building) and typically can support very fast data transfers compared to Internet transfer speeds.

Neocortex - A part of the outer layer of the brain responsible for higher level functions in mammals such as sensory perception, conscious thought, and language.

Neuron - An electrically excitable cell that processes and transmits information through electrical and chemical signals.

OpenGL - An open source graphics library for rendering in 2D and 3D which provides platform-independent access to hardware acceleration for fast rendering. It is widely considered a standard for portable graphics applications.

Port - While used in various contexts in computing, a “port” in networking is a number that identifies a destination for network data within the currently running software on a computer system. An application can open a port, and then send and/or receive information over a network using that port.

Qt – A cross-platform set of libraries and development tools which abstracts away the different ways of performing basic tasks on different systems. It is most well known for user interface design, although it also provides facilities for networking, threading, file I/O, and much more.

Rasterization – The process of creating fragment data that approximates the appearance of shapes such as triangles or lines.

Shader – A short computer program that runs on a Graphical Processing Unit (GPU). In the context of 3D rendering, a shader might transform vertex coordinates between reference frames or calculate lighting parameters for parts of objects in the scene.

Soma - The main cell body of a neuron.

Synapse - A structure within the brain that permits a neuron to pass chemical or electrical signals to another cell.

Transmission Control Protocol (TCP) - One of the main mid-level network protocols used on the Internet. It treats data as an uninterrupted stream and offers guarantees about reliability and order of data transfer: it can recover from some network errors, resending data if necessary, and different pieces of data are always received in the same order in which they were sent.

Vertex - A point in space that describes a corner of a surface.

Bibliography

- [1] Laurence C. Jayet Bray, Sridhar R. Anumandla, Corey M. Thibeault, Roger V. Hoang, Philip H. Goodman, Sergiu M. Dascalu, Bobby D. Bryant, and Frederick C. Harris Jr. “Real-time human-robot interaction underlying neurorobotic trust and intent recognition.” *International Joint Conference on Neural Networks*. Volume 32, pages 130-137. August 2012.

- [2] Romain Brette, Michelle Rudolph, Ted Carnevale, Michael Hines, David Beeman, James M. Bower, Markus Diesmann, Abigail Morrison, Philip H. Goodman, Frederick C. Harris Jr., Milind Zerpe, Thomas Natschläger, Dejan Pecevski, Bard Ermentrout, Mikael Djurfeldt, Anders Lansner, Olivier Rochel, Thierry Vieville, Eilif Muller, Andrew P. Davison, Sami El Boustani, and Alain Destexhe. “Simulations of networks of spiking neurons: A review of tools and strategies.” *Journal of Computational Neuroscience*. Volume 23, Issue 3. Dec 2007.

- [3] Alexander Jones, Justin Cardoza, Denver J. Liu, Laurence C. Jayet Bray, Bobby Bryant, Sergiu M. Dascalu, Sushil J. Louis, and Frederick C. Harris, Jr. “A Software Package for Visualizing Complex, Distributed Neural Networks.” *BMC Neuroscience*, To Appear.

- [4] Nathan M. Jordan, Kimberly B. Perry, Nitish Narala, Laurence C. Jayet Bray, Sergiu M. Dascalu, and Frederick C. Harris Jr. “Design and Implementation of an NCS-NeuroML Translator.” *ISCA International Conference on Software Engineering and Data Engineering*. Jun 2012.

- [5] Eugene M. Izhikevich. “Simple Model of Spiking Neurons.” *IEEE Transactions on Neural Networks*. November 2003.