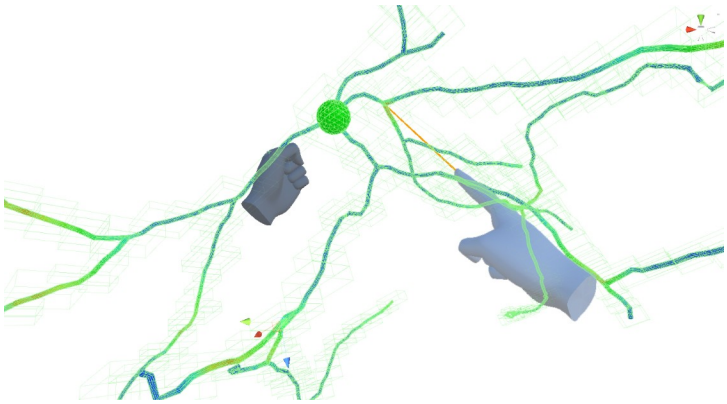
Neuro-VISOR

VISOR (Virtual Interactive Simulation Of Reality) is a research and software theme developed by Temple University's Center for Computational Mathematics and Modeling (C2M2), College of Science and Technology. The overarching long-term vision is to produce novel concepts and software that enable efficient immersed virtual reality (VR) visualization and real-time interaction with simulations of real-world processes described via principled mathematical equations. Unlike traditional high performance computing (HPC) applications, the philosophy of VISOR is that (a) the simulation runs while it is visualized in a virtual environment, and (b) the simulation continues even when the user affects and/or modifies the system state or its conditions.

Neuro-VISOR focuses on applications in computational neuroscience. Specifically, it provides a pipeline that (a) retrieves a wire-frame neuron geometry file from the public neuron database NeuroMorpho, and (b) generates a surface mesh from it. This mesh is then (c) visualized in VR, while an efficient numerical method approximates the Hodgkin-Huxley model on the given wire-frame neuron geometry. Finally, while the running simulation is fed to the VR environment in real time, (d) the user can interact with the surface mesh and affect the simulation while it is running via several methods outlined under controls.



A key use case of this framework is that is can rapidly accelerate scientific discovery by providing an immediate and very intuitive feedback to the user about how changes to the simulated system affect the system's behavior. This insight, obtained from the simple and fast models used for VISOR, can then enable the computational scientist to devise significantly more targeted (non-interactive) simulations on large-scale HPC clusters of more complex models. In addition, in the context of neuroscience, the immersed 3D environment provides a more intuitive way to navigate and comprehend complex neuron geometries than traditional visualizations on computer screens.

The Neuro-VISOR software can be run in a VR version (currently only for Oculus), as well as via a desktop version (without VR headset). While the desktop version does not provide the 3D visualization of the VR version, it does provide most of the same interactive real-time simulation capabilities.

code quality C

License

We use a modified version of the GNU LPGL v3. Our license can be found in LICENSE.txt

Research Team

This project is produced at the Center for Computational Mathematics and Modeling (C2M2) at Temple University.

Project lead: Dr. Benjamin Seibold, Dr. Gillian Queisser

Researchers and developers: Craig Fox (in), Stephan Grein (in), Bogdan Nagirniak, James Rosado (in), Jacob Wells (in), Noah Williams.

Code Documentation

We have code documentation generated using Doxygen. The completeness of this documentation is dependent on code commenting, so there may be gaps and imperfections. If you notice issues with this documentation, please report it to jacob.wells@temple.edu.

Other code

We maintain a separate project for generating neuron grids.

We have another project containing additional custom attributes which proved useful during development.

Connect with us

We have a blog where we go into finer detail about some of our past and current code solutions. While far from complete, we feel that this blog still has lots of interesting details about the areas of the project that it does cover.

Inquiries about the project can be made to seibold@temple.edu.

Performance issues and code-related questions can be sent to jacob.wells@temple.edu

Use Guide

Recommended hardware

- **Oculus Rift, Rift S, or Quest Head Mounted Display (HMD)** - This project is currently tested on the Oculus Rift S and the Oculus Quest. It was previously developed on the original Rift headset. This project was developed using the Oculus SDK. We have not been able to run this project on other headsets, but have been working to make it more cross-compatible.
- We do not have rigorously tested minimum computer specs. We have listed our hardware setup below for convenience. Oculus has provided a list of VR-ready computers, but we do not necessarily endorse any computer recommended by them.

Dell Precision 5820 Tower	
CPU	Intel Xeon W-2125
GPU	NVIDIA GeForce GTX 1080
RAM	32 GB

Recommended software

- Unity 2019.4.26f1 (LTS)
- Windows 10 64-bit - This program may be functional on other operating systems, but its performance is not guaranteed. The emulator functions have been tested on Ubuntu 16.04 LTS.
- Relevant HMD software and drivers
- We use the Oculus Desktop 2.38.4 package, but are planning to move to Unity's XR Plugin system soon.
- Git version >= 2.7.0
- Git LFS version >= 2.10.0
- Note: Git users (versions < 2.23.0) should clone the repository by using `git lfs clone`. All other should use `git clone`. For users of older git versions this remedies the problem that every LFS versioned file will ask the user for their password.
- A pre-commit hook will block commits made with inappropriate versions of Git, Git LFS, or Unity.

Users should install the hooks by calling `./install_git_hooks.sh` from the root directory after cloning.

- git lfs: `git lfs env` in a terminal/console.
- git: `git --version`
- Unity: see UnityEditor

Starting the program

Make sure that your HMD is set up and that you have gone through the first-time setup. Ideally you have tested several other programs on your headset before using this software.

1. Clone project to any location (NOTE: Downloading the zip archive of this Unity Project is problematic, as GitHub does not currently support archiving files versioned with Git LFS. Downloading static build releases still works as intended).
2. Ensure the correct version of the Unity Editor is installed
3. Open project in Unity and open Assets/Scenes/MainScene
4. Place desired neuron cell files in `Assets/StreamingAssets/NeuronalDynamics/Geometries`. Any cells found in this directory will be available at runtime.
5. Start the application by pressing play in the editor.
6. Upon startup, the user is placed in a model of C2M2's lab. The application will detect a VR headset and launch in VR/keyboard emulator mode automatically.
7. Our control scheme is outlined below. Try moving around and looking at your hands.

Controls

Oculus touch controller naming conventions can be found here. An instructional video demonstrating the controls can be found here. The term 'Hand Trigger' and 'Grip Trigger' are used analogously throughout the project, as are the terms 'Index Trigger' and 'Front Trigger'.

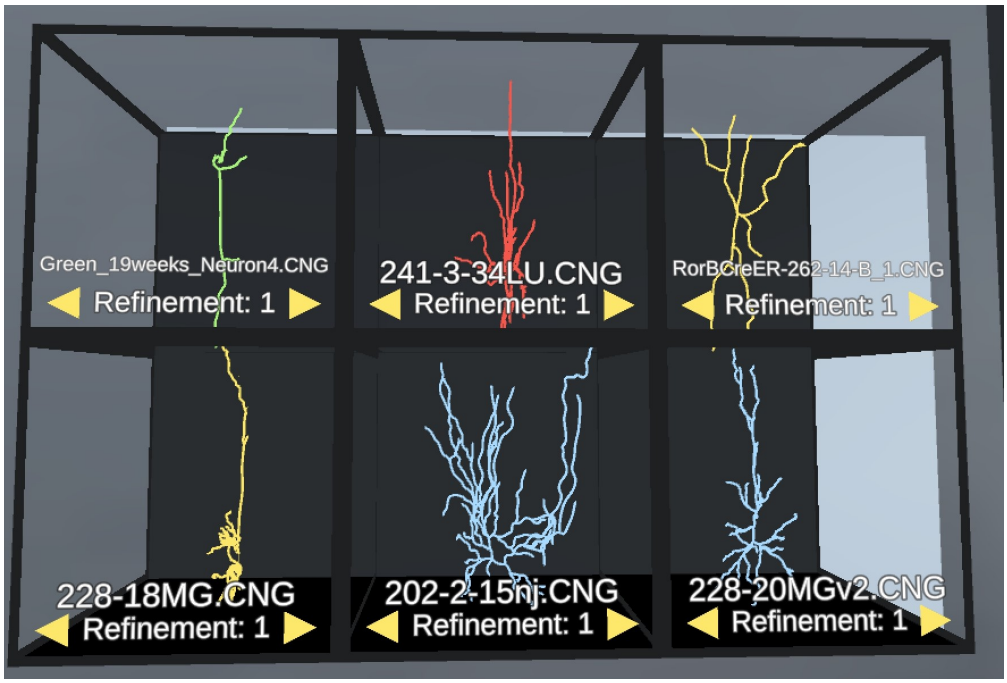
Action	Rift Touch Controllers	Keyboard/Mouse
Toggle raycasting	(P) Button one (A/X)	Always on
Grab	(H) Hand trigger	Not supported
Interact	Raycasting + (P/H) Index trigger	(P/H) Left mouse button

Action	Rift Touch Controllers	Keyboard/Mouse
Scale Object	Grab + Thumbstick up/down	N/A
Move camera	Walk around!	WASD
Rotate camera	Look around!	(H) Left-Ctrl + move mouse cursor

(P)	Press button once
(H)	Hold button down

Selecting a cell

1. A cell previewer stands against the whiteboard near the window. It attempts to render the 1D mesh of any neuron `.vrn` cell file archives found in



`Assets/StreamingAssets/NeuronalDynamics/Geometries`. Three example cells are included with this repo. Several more cells can be found here.

2. Enable raycast mode in VR. The hand with raycast mode enabled should be constantly pointing forward.

3. With raycast mode enabled, hover over a cell preview window by pointing at it. A blue guide line should be drawn between your pointer finger and the preview window. Continue hovering to see more information about the cell, or press the Interact button to load the cell and launch solve code. The guide line should turn orange while pressing/holding. The geometry should render in the middle of the room, scaled to fit within the room and appearing in a uniform color. A color scale should now be displayed on the whiteboard.

Simple cell interaction

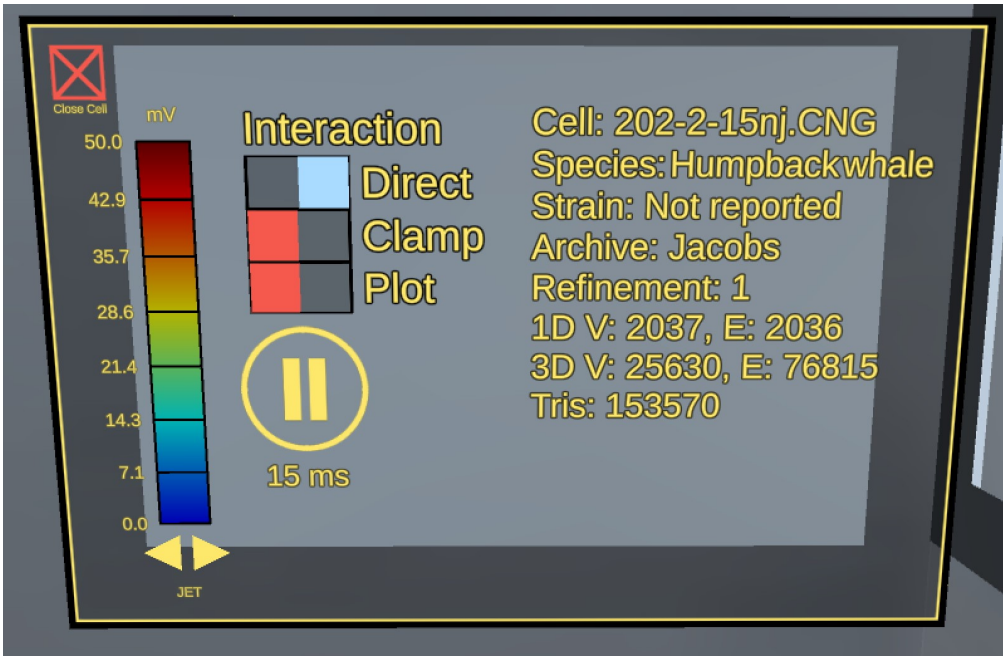
Grabbing

The cell can be grabbed by hovering your hand over the 3D geometry and pressing the hand trigger on that hand's controller. Once grabbed, you can move and rotate cells freely.

Resizing

While grabbing a cell, hold the thumbstick up or down on the hand that is being used to grab the cell to resize the cell in world space. Note: this does not affect the environment of the solver code: the cell can be scaled freely in world space without affecting the stored vertex positions of the 1D or 3D meshes.

Board info and controls



A large user interface is spawned upon selecting a cell. This board contains useful static information about the cell.

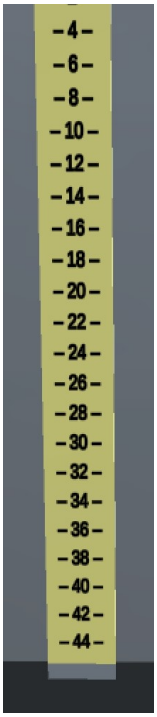
The board contains a color table that shows the current color gradient applied to the cell. The voltage value corresponding to different color values is diosplayed to the left of the color table. The user can change the value corresponding to the top or bottom of the color scale by hovering their finger cursor over either number and holding up/down on the joystick (or the up/down arrow key). The user can also change the color scheme by interacting with the arrows beneath the color table.

The board contains a subpanel for selecting the type of interaction. The user can select any of these toggles to change the effect of directly interacting with the surface. See direct, clamp, and plot interactions for explanations of each.

The board also contains a play/pause button. The user can use this to pause solver code at the current time step. Beneath is displayed the current simulation time.

Ruler controls

A ruler is spawned with every cell geometry. The ruler can be used to understand the length scale of the cell in its local space. While grabbing the ruler, resize it by moving the thumbstick up or down on the hand that is being used to grab the ruler.

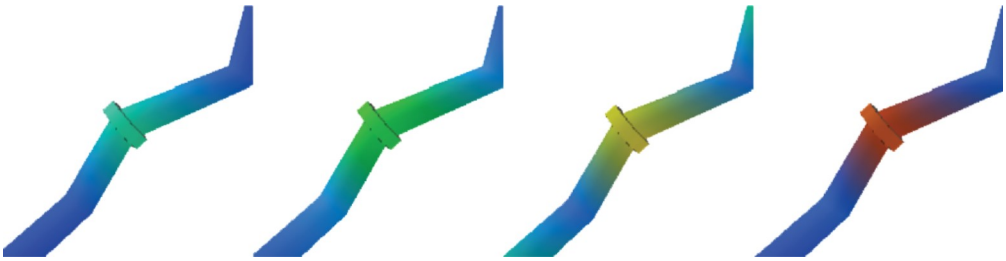


The measurements on the ruler will adapt to the world-space size of the geometry so that it can always act as a translator between the size of the cell in the user's space and the local length scales of the neuron.

Direct cell interaction

With raycast mode enabled, point at the surface of the geometry. A blue line should be drawn between your pointer finger and the surface of the geometry. Tap the geometry from up close, or press the Interact button from a distance to directly alter simulation value at the nearest 1D vertex to the point of interaction. The guide line should turn orange upon pressing, and the surface of the geometry should change color to reflect the affected potential at the nearest 1D vertex on the geometry.

Clamp cell interaction



Clamps can be used to continuously alter the value of a single vertex on the 1D mesh.

Enable Clamp Mode

With the cell loaded and raycast mode enabled, press the `Clamp Mode` button on the whiteboard. "Finger clamps" should appear on both of the user's pointer fingers whilst raycasting. The finger clamps should appear as cylinders on the user's pointer finger.

With clamp mode enabled, interacting with the surface of the cell should place a cylindrical potential clamp on the cell near the point of interaction. The clamp will snap to the nearest 1D vertex to the 3D point of interaction, and will effect that 1D vertex on the 1D mesh.

Toggling a clamp

Enable or disable the clamp by tapping it from up close or pressing the Interact button while pointing at it from a distance. The clamp is gray when it is disabled. It's color when enabled should reflect the clamp's current potential value.

Changing a clamp's power

While the clamp is enabled, the clamp's power can be changed by pointing at it, holding the Interact button, and moving the controller's joystick up or down.

Destroying a clamp

Clamps can be destroyed by holding the Interact button briefly while pointing at the clamp, and then releasing the Interact button.

Group clamp controls, clamp highlighting

Users can place many clamps on the geometry. Clamps cannot be placed too near to each other, but there is no set limit to the number of clamps that can be placed.

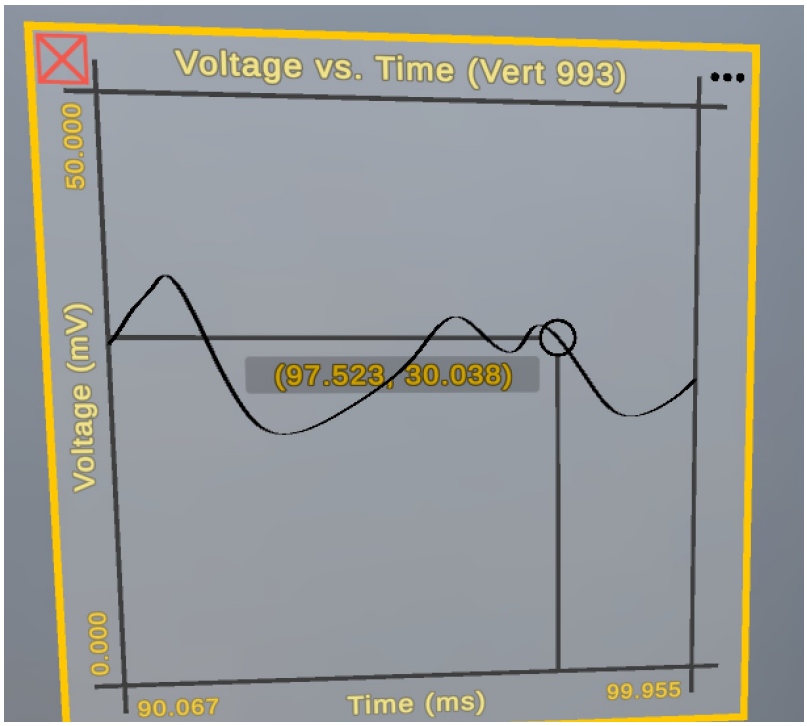
In addition to each clamp being individually interactable, the "finger clamps" mentioned in Enable Clamp Mode are also interactable, and act as controllers for all existing clamps. Any control that can be done to a single clamp (toggle, alter power, destroy), can be done to all existing clamps at once by interacting with the finger clamp in the same manner.

The user can highlight all clamps with a red sphere to clarify their position to the user. This is particularly useful on complicated geometries with many clamps attached. While pointing at the finger clamp and holding the Interact button, hold down the hand trigger to highlight all existing spheres.

Point-plotter interaction

The user is able to spawn line graphs that are attached to specific 1D vertices on the neuron cell. These graphs will show the voltage at that 1D point over time.

To spawn a graph panel, the user simply needs to toggle "Plot" on the board UI. At this point the user can raycast onto the surface of the mesh and press the index trigger (or click with the mouse) in order to spawn a graph attached to the nearest 1D vertex. Panels are grabbable and resizeable in the same way that the cell or the ruler is.



The user can hover over the graph-plane with their raycasting finger or the mouse in order to spawn a cursor at the point of hovering. The cursor will display the exact value of the graph at that point. The user can then press the index trigger (or click on the mouse) to lock the cursor at that position. Clicking again will free the cursor.

The number of samples in the graph can be altered by opening the "more info" panel (top-right corner of the graph), hovering over "number of samples", and holding up or down on the joystick (or holding the up or down arrows on the keyboard).

Standalone Builds

A standalone build for Windows 64-bit can be downloaded from the "Releases" section. Standalone builds can be run as executable files without the need for the Unity editor and will improve performance. Builds can be made by anybody with the Unity project, but we have made "approved" builds with relevant build information.

Within a static build, custom .vrn cell archives can be placed within `virtual-reality_Data\StreamingAssets\NeuronalDynamics\Geometries` to be run within the application.