OpenWorm Documentation

Release 0.5

OpenWorm

October 17, 2014

Contents

Contents:

CHAPTER 1

Introduction to OpenWorm

1.1 Welcome

OpenWorm is an open source project and open science community dedicated to creating the world's first whole organism in a computer, a *C. elegans* nematode, via bottom-up "systems biology" computational modeling. It is an association of highly motivated scientists, engineers, coders, and curious citizens from around the world who believe in open science and open access.

1.2 Mission/Vision

The complexity of computational neuroscience and biology make it extremely difficult to sort through the myriad facts, data, and biological processes that are uncovered on a daily basis by researchers around the world.

OpenWorm believes that the challenges of solving brain simulation, even for the simplest of model organisms, require open access and collaborative solutions.

OpenWorm is actively working to achieve its goal - creating the world's first virtual organism in a computer - by:

- bringing together highly motivated scientists and engineers in the open space
- pushing away all the red tape by taking open science to the extreme
- fostering growth of a completely open computational biology community

1.3 Goal

Our main goal is to build the world's first virtual organism– an in silico implementation of a living creature– for the purpose of achieving an understanding of the events and mechanisms of living cells. Our secondary goal is to enable, via simulation, unprecedented in silico experiments of living cells to power the next generation of advanced systems biology analysis, synthetic biology, computational drug discovery and dynamic disease modeling.

1.4 Navigating OpenWorm

We've created this documentation to help orient you to the different locations on the web where OpenWorm material is found and where contributions can be made.

The *modeling approach page* explains how we have broken down this problem and what steps we are currently taking.

The *resources page* has a gallery of content that has been produced by the project, including simulation engines, visualization environments, and data sets.

There are a lot of additional questions you may have about the project. We have assembled a *frequently asked questions (FAQ)* document to help you. You may also wish to use the search feature in our documentation.

1.5 Contributing to OpenWorm

To start off the process, please give us some information about yourself on this form. We recommend as well that you sign up to this mailing list and peruse the archives to get a sense of what is going on.

Then, please check out a recent orientation overview, and browse *our project list* to understand the different areas where work is happening. To put the projects in context, you will find it useful to read more about the *big picture idea of the modeling approach* we are taking.

If you are interested in a specific programming language, check out links to issues specifically for python or c++. We also have a lot of Java and Javascript/HTML/CSS development going on as part of Geppetto. See the list of issues here.

If you have questions about specific things you find, please post them to the list.

More information about the process of making a contribution is *available on our community page*

While the heart of OpenWorm is computational modeling, we are always looking for people with talents beyond programming to contribute. Are you a graphic designer, writer, PR specialist or simply someone with a love of science and expertise to share? Please reach out to us at info@openworm.org to discuss opportuntities with OpenWorm.

CHAPTER 2

Project Background

2.1 History

Established in January 2011 and in just 2 years of activity, OpenWorm has built a community of highly-motivated and highly-skilled individuals and coordinated their work. The community has produced regular scientific publications making extensive use of scientific research published through open access, proving the validity of the open science approach taken.

More information is available on *the past history of releases the project has done*

2.2 Why do this?

There has never been a scientific result in biology or neuroscience that is inconsistent with the idea that brains are 100% physical matter. We tend to forget, but our brains are tissues just like our lungs and heart are. If I could stick an electrode in your brain right now I could record activity from your neurons that corresponds with your thoughts. The problem is that scientists can't understand what all that activity means yet.

Scientists can make models of sending a rocket to land on the surface of Mars, but not a model of all the activity of your brain. Scientists lack well-agreed upon models of complex neuronal activity because such models are hard to produce.

Complex neuronal activity is the result of countless interactions between molecules happening inside, between, and around neurons. Because of all the interactions that make up complex neuronal activity, you need to gather up a lot of information to make models of it. Also because of all those interactions, you need sophisticated software platforms to manage all that information properly.

We are building a simulation platform to prove it is possible to make good models of complex neuronal activity, starting with a digital worm. We are making the simulation platform open source because we believe anyone should be able to use it to understand how neurons and cells work.

2.3 Why c. elegans?

In the field of neuroscience, one of the simplest organisms that are studied is the c. elegans. It only has 302 neurons, has a very consistent lifecycle, and is well studied. Its whole body has only 1000 cells total. With those 1000 cells it solves basic problems of feeding, mate-finding, predator and toxin avoidance using a nervous system driving muscles on a body in a complex world.

The cells in its body work together to produce its behavior. Instead of starting with the behavior and building a simple system to capture it, we are starting with making models of the individual cells and their interactions. If we do this

correctly so that the cells act on each other as they do in the real organism, we will have a much more realistic model than we would get trying to go straight to the behavior.

This seems to us the only sensible starting point to creating a true biological simulation that captures enough details and has enough constraints to approximate real biology. Simulating a single cell that doesn't move (like a yeast cell) isn't going to provide us enough of a foundation to build up to more complex organisms by itself. If we can't accomplish a simulation at this humble scale, we'll never be able to do it at the massive scale of the human brain. The technology that would come out of this endeavor would be applicable to much more complex organisms down the road.

2.4 On models

Models are the cornerstone of science. Tools like algebra and calculus and newtonian mechanics and computer spreadsheets were advances because we could plug numbers into equations and get answers out that told us something about the world.

Unfortunately, neuroscience has few predictive models for how nervous systems work.

We are starting by building a full simulation of a small biological system with a reasonable number of parts. We are focused on capturing as much of the rich detail of that biological system as possible.

2.5 Concepts

2.5.1 Top-down simulation

Our first instincts when looking at a system we want to simulate is to come up with a list of its obvious features and then try to pick the simplest means of simulating it. In the case of obvious top-down ways to model a worm, one might capture the fact that it bends in a sinusoidal pattern as a good starting point, and begin implementing sine and cosine functions that can capture this.

There is an important place for this kind of simulation, but we have found that one rapidly runs into limitations of generalization. The model that worked great for crawling no longer works for turning around. The simplest thing possible is added to the model to make it work for turning around, but soon there is another aspect to capture, and then another. Soon, the model is a series of hacks that become increasingly brittle.

Instead of a pure top-down approach, we employ a balanced top-down, bottom-up approach, with a greater emphasis on the bottom up.

2.5.2 Bottom-up simulation

Biology teaches us that when it comes to understanding how animals work, understanding the behavior of cells is critical. Our bodies are made up of between 40 and 100 trillion cells, and it is these cells working together that make up everything we are and do. Of particular interest are the cells in the brain and larger nervous system, that are responsible for our thoughts, creativity and feelings.

Today, science has barely scratched the surface of how to make best use of the enormous power of computers to create models of cellular activity. Scientists have not yet placed computer models of cells at the center of biology.

A "bottom-up" simulation, in this case, is an attempt to model the individual cells in the organism, giving them behaviors which, when combined together, produce the outward behavior of the entire organism. This is as opposed to building the organism without consideration for individual cells to start with, and adding cells in later.

In reality, we always have to do some bottom-up simulation along with top-down simulation, in order to make progress. But in general and where possible, we view what we are doing as focused on simulating cells first.

2.5.3 Multi-algorithm integration

Just as mathematics has played a crucial role in the description of physics, mathematicians have approached the field of biology with the goal of describing biological activity more precisely. Generally speaking, this means that if it happens inside a biological organism, there should be a set of equations that can explain how it works. A great deal of creativity goes into coming up with such equations.

Once equations have been determined, computers are great at calculating them once they have been turned into algorithms. Algorithms become the computer's way of handling a bunch of equations.

The challenge is that there are a lot of equations that are necessary to fully specify how cellular activity works. A recent whole cell model of a relatively simple cell came up with 32 algorithms composed of many more equations and a ton of data.

The consequence of this from an engineering perspective is, in order to simulate complex living systems, we need software that is flexible enough to let us assemble the algorithms we need in just the right ways. We call this "multi-algorithm integration".

2.5.4 Model optimization

There are a lot of aspects of the *c. elegans* that we will not be able to measure directly for a while based on experimental limitations. These are "free parameters". The conventional wisdom on modeling is to minimize the number of free parameters as much as possible. Sometimes, the large number of free parameters are used as an argument to avoid making computational simulations.

In this case, we have to make do with what we have and make some good educated guesses about the free parameters. There is a mathematical discipline that helps us do that known as optimization. For our purposes, you can think of this as generating many different versions of a model, each version with slightly different parameters, and then measuring if the model produces good results. If a model produces better results by changing the parameters in a particular way, you try to keep changing the parameters in that way and see if you get even better results. In this way, roughly speaking, optimization techniques enable scientists to turn a problem of lack of data into a problem that a computer can address using brute force calculations.

2.5.5 NeuroML

NeuroML is an XML (Extensible Markup Language) based model description language that aims to provide a common data format for defining and exchanging models in computational neuroscience. The focus of NeuroML is on models which are based on the biophysical and anatomical properties of real neurons. (Wikipedia). NeuroML is known as an open standard, because its means of describing a model is publicly available for others to improve upon.

CHAPTER 3

OpenWorm Modeling Approach

Our main goal is to build the world's first virtual organism– an *in silico* implementation of a living creature– for the purpose of achieving an understanding of the events and mechanisms of living cells. Our secondary goal is to enable, via simulation, unprecedented in silico experiments of living cells to power the next generation of advanced systems biology analysis, synthetic biology, computational drug discovery and dynamic disease modeling.

Contents

In order to achieve these goals, we first began with an informal cartoon representation of a breakdown of cell types and various biological processes that the worm has. Here is a representation of a small subset of those processes, where arrows show the causal relationship between processes and cell types:

This picture is purposefully drawn with an underlying loop of causal relationships, and is partially inspired by the work of Robert Rosen. The decision to focus on any one loop is arbitrary. Many different loops of causal relationships could be plucked out of the processes that underly the worm. However, focusing on first dealing with the loop that deals with behavior in the environment and through the nervous system has several advantages as a starting point:

- Crawling behavior of worms is relatively easy to measure
- The 302 neurons responsible for the behavior are well mapped
- Enables the study of the nervous system as a real time control system for a body
- Provides the model with a minimum core to which other biological processes and cell types can be added.

Having chosen one loop to focus on first, we can now re-define the problem as how to construct an acceptable neuromechanical model. There have been other attempts to do this in the past and there are some groups currently working on the problem using different approaches (e.g. Cohen, Lockery, Si Elegans).

Our approach involves building a 3D mechanical model of the worm body and nervous system, tuning the model using model optimization techniques, validating the model using real data, and ensuring the model is reproducible by other labs by exposing it through a web-based simulation engine.

3.1 Closing the loop with neuromechanical modeling

While our ultimate goal is to simulate every cell in the *c. elegans*, we are starting out by building a model of its body and environment, its nervous system, and its muscle cells.

To get a quick idea of what this looks like, check out the CyberElegans prototype. In this movie you can see a simulated 3D c. elegans being activated in an environment. Similar to the CyberElegans model, its muscles are located around the outside of its body, and as they turn red, they are exerting forces on the body that cause the bending to happen.

These steps are outlined in blue text of the figure below:

Our end goal for the first version is to complete the entire loop. We believe that this is the most meaningful place to begin because it enables us to study the relationship between a nervous system, the body it is controlling, and the environment that body has to navigate. We also believe this is a novel development because there are no existing computational models of any nervous systems that complete this loop. For an excellent review of the current state of research on this topic, check out Cohen & Sanders, 2014

When we first started, our team in Novosibirsk had produced an awesome prototype. We recently published an article about it. If you watch the movie that goes along with the prototype, you can see the basic components of the loop above in action:

Here muscle cells cause the motion of the body of the worm along the surface of its environment.

Inside the worm, motor neurons are responsible for activating the muscles, which then makes the worms move. The blue portions of the loop diagram above are those aspects that are covered by the initial prototype. We are now in the process of both adding in the missing portions of the loop, as well as making the existing portions more biologically realistic, and making the software platform they are operating on more scalable.

3.2 Components

In order to accomplish this vision, we have to describe the different pieces of the loop separately in order to understand how to model them effectively. This consists of modeling the body within an environment, the neurons, and the muscle cells.

3.2.1 Body and environment

One of the aspects of making the model more biologically realistic has been to incorporate a 3d model of the anatomy of the worm into the simulation.

To get a quick idea of what this looks like, check out the latest movie. In this movie you can see a simulated 3D c. elegans being activated in an environment. Its muscles are located around the outside of its body, and as they turn red, they are exerting forces on the body that cause the bending to happen.

In turn, the activity of the muscles are being driven by the activity of neurons within the body.

More detailed information is available on the *Sibernetic project page*.

Having a virtual body now allows us to try out many different ways to control it using signals that could arise from neurons. Separately, we have been doing work to create a realistic model of the worm's neurons.

3.2.2 Neurons

This is a much more faithful representation of the neurons and their positions within the worm's body.

Our computational strategy to model the nervous system involves first reusing the c. elegans connectome and the 3D anatomical map of the c. elegans nervous system and body plan. We have used the NeuroML standard (Gleeson et al., 2010) to describe the 3D anatomical map of the c. elegans nervous system. This has been done by discretizing each neuron into multiple compartments, while preserving its three-dimensional position and structure. We have then defined the connections between the NeuroML neurons using the c. elegans connectome. Because NeuroML has a well-defined mapping into a system of Hodgkin-Huxley equations, it is currently possible to import the "spatial connectome" into the NEURON simulator (Hines & Carnevale 1997) to perform *in silico* experiments.

To start getting some practical experience playing with dynamics that come from the connectome, we have simplified it into a project called the 'connectome engine' and integrated its dynamics into a Lego Mindstorms EV3 robot. You can see a movie of this in action.

More information about working with the data within it and other data entities can be found *on the data representation project page*.

These neurons must eventually send signals to muscle cells.

3.2.3 Muscle cells

We have started our process of modeling muscle cells by choosing a specific muscle cell to target:

More information about working with the data within it and other data entities can be found *on the data representation project page*.

Once the body, neurons, and muscles are represented, we still have a lot of free parameters that we don't know. That's what leads us to the need to tune the model.

3.3 Tuning

The way we make the model biophysically realistic is to use sophisticated mathematics to drive the simulation that keep it more closely tied to real biology. This is important because we want the model to be able to inform real biological experiments and more coarse-grained, simplified mathematics falls short in many cases.

Specifically for this loop, we have found that two systems of equations will cover both aspects of the loop, broadly speaking:

As you can see, where the two sets of equations overlap is with the activation of muscle cells. As a result, we have taken steps to use the muscle cell as a pilot of our more biologically realistic modeling, as well as our software integration of different set of equations assembled into an algorithmic "solver".

These two algorithms, Hodgkin-Huxley and SPH, require parameters to be set in order for them to function properly, and therefore create some "known unknows" or "free parameters" we must define in order for the algorithm to function at all. For Hodgkin-Huxley we must define the ion channel species and set their conductance parameters. For SPH, we must define mass and the forces that one set of particles exert on another, which in turn means defining the mass of muscles and how much they pull. The conventional wisdom on modeling is to minimize the number of free parameters as much as possible, but we know there will be a vast parameter space associated with the model.

To deal with the space of free parameters, two strategies are employed. First, by using parameters that are based on actual physical processes, many different means can be used to provide sensible estimates. For example, we can estimate the volume and mass of a muscle cell based on figures that have been created in the scientific literature that show its basic dimensions, and some educated guesses about the weight of muscle tissue. Secondly, to go beyond educated estimates into more detailed measurements, we can employ model optimization techniques. Briefly stated, these computational techniques enable a rational way to generate multiple models with differing parameters and choose those sets of parameters that best pass a series of tests. For example, the conductances of motor neurons can be set by what keeps the activity those neurons within the boundaries of an appropriate dynamic range, given calcium trace recordings data of those neurons as constraints.

If you'd be interested to help with tuning the model, please check out the *Optimization project page*.

3.4 Validation

In order to know that we are making meaningful scientific progress, we need to validate the model using information from real worms. The movement validation project is working with an existing database of worm movement to make the critical comparisons.

The main goal of the Movement Validation team is to finish a test pipeline so the OpenWorm project can run a behavioural phenotyping of its virtual worm, using the same statistical tests the Schafer lab used on their real worm data.

More detailed information is available on the *Movement validation project page*.

3.5 Reproducibility

In order to allow the world to play with the model easily, we are engineering Geppetto, an open-source modular platform to enable multi-scale and multi-algorithm interactive simulation of biological systems. Geppetto features a built-in WebGL visualizer that offers out-of-the-box visualization of simulated models right in the browser. You can read about architectural concepts and learn more about the different plug-in bundles we are working on.

The *project page for Geppetto* has information about getting involved in its development with OpenWorm.

CHAPTER 4

OpenWorm Projects

4.1 NeuroMechanical Modeling - Sibernetic

Contents

While our ultimate goal is to simulate every cell in the c. Elegans, we are starting out by building a model of its body, its nervous system, and its environment. Sibernetic is the home of the C++ code base that implements the core of the model. We have implemented an algorithm called Smoothed Particle Hydrodynamics (SPH) to simulate the body of the worm and its environment using GPUs. This algorithm has been initially worked out in C++ (with OpenGL visualization).

To get a quick idea of what this looks like, check out the latest movie. In this movie you can see a simulated 3D c. elegans being activated in an environment. Its muscles are located around the outside of its body, and as they turn red, they are exerting forces on the body that cause the bending to happen.

4.1.1 Previous accomplishments

- Physics tests
- Initial worm crawling

4.1.2 Current roadmap

Electrofluid Paper

We are writing a manuscript focusing on the work we have to implement SPH in the project and apply it to muscle cells and the worm body. @vellamike, @a-palyanov and @skhayrulin are taking the lead on this,

The proposal is to do this after the Sibernetic proof of concept worm wiggling is complete.

4.1.3 Issues list

All issues related to the Sibernetic code base can be found on GitHub.

4.1.4 Associated Repositories

Repository	Description	Language
Smoothed-Particle-Hydrodynamics	**The Sibernetic code base containing the 2014 version of the worm body model,** a C++ implementation of the Smoothed Particle Hydrodynamics algorithm customised for the OpenWorm project.	C++
ConfigurationGenerator	Generation start scene configuration for PCI SPH solver	JavaScript
CyberElegans	Circa 2010 Neuromechanical model of C. Elegans	C++

4.2 Geppetto Simulation Engine

Contents

In order to allow the world to play with the model easily, we are engineering Geppetto, an open-source modular platform to enable multi-scale and multi-algorithm interactive simulation of biological systems. Geppetto features a built-in WebGL visualizer that offers out-of-the-box visualization of simulated models right in the browser. You can read about architectural concepts and learn more about the different plug-in bundles we are working on.

Geppetto, is written in Java and leverages technologies like OSGi, Spring Framework, OpenCL and Maven.

Geppetto's frontend is written using THREE.js and WebGL. Back-end / front-end communication happens via JSON messages through WebSocket.

The engine runs on on Eclipse Virgo WebServer deployed on an Amazon Elastic Compute Cloud Linux instance.

4.2.1 Previous accomplishments

- Past releases of Geppetto

4.2.2 Current roadmap

STORY: Worm wiggling in the browser

As a user, I want to see the proof of concept sibernetic worm in my web browser so that anyone around the world can play with it.

Practically, this means porting the proof of concept scene into Geppetto.

STORY: Interactive worm wiggling in browser

As a user, I want to be able to see a visualization of the proof of concept worm wiggling in my web browser and be able to perturb it in a manner that causes the wiggling to change in a realistic manner.

This milestone suggests interactivity via Geppetto. The kind of perturbation is not defined yet– ideally we should aim for the simplest kind we can think of that gives the user an interface to make modifications.

4.2.3 Issues list

The issues related to Geppetto are distributed across different repositories.

Issues related to general functionalities that need to be added to support the OpenWorm simulation are found here.

Issues related to the platform in general are found here.

Ultimately every module of Geppetto has issues of its own, see the list of repositories below.

The issues are so splitted to allow capturing different granularities, having both issues that reflect what macro functionalities need to be added in the OpenWorm and Geppetto repository and having detailed close-to-the-code bugs in the individual repositories.

4.2.4 Associated Repositories

Repository	Description	Language
org.gepetto	Geppetto Main Bundle and packaging	Java
org.geppetto.solver.sph	PCI SPH Solver bundle for Geppetto	Java
org.geppetto.simulator.jlems	jLEMS based simulator for Geppetto	Java
org.geppetto.model.neuroml	NeuroML Model Bundle for Geppetto	Java
org.geppetto.core	Geppetto core bundle	Java
org.geppetto.frontend	Geppetto frontend bundle - Web application	Java
org.geppetto.testbackend	Geppetto test backend for Geppetto	Java
org.geppetto.simulator.sph	SPH Simulator bundle for Geppetto	Java
org.geppetto.simulation	Generic simulation bundle for Geppetto	Java
org.geppetto.model.sph	PCI SPH Model Bundle for Geppetto	Java
org.geppetto.samples	Sample simulations for Geppetto	Descriptive
org.geppetto.templatebundle	Template bundle	Java

4.3 Movement Validation

Contents

In order to know that we are making meaningful scientific progress, we need to validate the model using information from real worms. The movement validation project is working with an existing database of worm movement to make the critical comparisons.

The main goal of the Movement Validation team is to finish a test pipeline so the OpenWorm project can run a behavioural phenotyping of its virtual worm, using the same statistical tests the Schafer lab used on their real worm data.

4.3.1 Previous accomplishments

- All code necessary to reproduce Ev Yemini's Nature Methods paper was obtained in October 2013. Jim has stored it in the MRC_wormtracker_GUI repo.
  - This is in addition to the SegWorm repo, although we will be merging them.
  - It has code to generate features from measurements.
- A movement validation GitHub repository was started specifically with the goal of developing

  the infrastructure to validate model worm movements against real worms.

4.3.2 Current roadmap

STORY: Build a test suite for the simulation from WormBehavior database

As a scientist or developer, I want to be able to run a test suite against the simulation that will show me how close the model is to real data.

In order for a model to demonstrate scientific value, it has to make falsifiable predictions. The target data to be able to predict will be drawn from the WormBehavior database. This milestone will involve working with these data, creating a code base that can compare movement output from the simulation with ground truth from the database and produce an accuracy score.

This story breaks down the epic to predict behavior from the WormBehavior database

EPIC: Correctly predict 80% of wild type (N2) behavior in WormBehavior database

This epic is to have a simulation that can demonstrate it can predict (and therefore reproduce) 80% of the data collected about the N2 worm in the WormBehavior database. This means building a training set and a test set that are kept separate from each other, using the training set to tune up the model, then generating predictions, and comparing them against the test set, and doing some cross-validation).

This epic focuses on an output of simulation performance rather than the means of implementation, so any way to achieve this epic is welcome.

More information on next steps is available in a recent progress report.

4.3.3 Issues list

All issues related to movement validation can be found on GitHub

4.3.4 Associated Repositories

Repository	Description	Language
movement_validation	A test pipeline that allows us to run a behavioural phenotyping of our virtual worm running the same test statistics the Shafer lab used on their worm data.	Python
SegWorm	SegWorm is Matlab code from Dr. Eviatar Yemini built as part of the WormBehavior database (http://wormbehavior.mrc-lmb.cam.ac.uk/)	Matlab

4.4 Optimization engine

Contents

The Optimization engine uses optimization techniques like genetic algorithms to help fill gaps in our knowledge of the electrophysiology of *C. elegans* muscle cells and neurons.

These two algorithms, Hodgkin-Huxley and SPH, require parameters to be set in order for them to function properly, and therefore create some "known unknows" or "free parameters" we must define in order for the algorithm to function at all. For Hodgkin-Huxley we must define the ion channel species and set their conductance parameters. For SPH, we must define mass and the forces that one set of particles exert on another, which in turn means defining the mass of muscles and how much they pull. The conventional wisdom on modeling is to minimize the number of free parameters as much as possible, but we know there will be a vast parameter space associated with the model.

To deal with the space of free parameters, two strategies are employed. First, by using parameters that are based on actual physical processes, many different means can be used to provide sensible estimates. For example, we can estimate the volume and mass of a muscle cell based on figures that have been created in the scientific literature that show its basic dimensions, and some educated guesses about the weight of muscle tissue. Secondly, to go beyond educated estimates into more detailed measurements, we can employ model optimization techniques. Briefly stated, these computational techniques enable a rational way to generate multiple models with differing parameters and choose those sets of parameters that best pass a series of tests. For example, the conductances of motor neurons can be set by what keeps the activity those neurons within the boundaries of an appropriate dynamic range, given calcium trace recordings data of those neurons as constraints.

4.4.1 Previous accomplishments

- Genetic algorithms applied to tuning muscle cell models

4.4.2 Current roadmap

STORY: Muscle Cell model output closely matches that of real data

We will show that we have built a model of C. elegans muscle cell that matches data recorded from the nematode muscle cell. In part, we will use techniques of model optimization to fill in gaps in the model parameter space (deduce unmeasured parameters). The main technical challenge is tuning muscle cell passive properties and building a larger data set (more cell recordings).

bionet: training C. elegans with a specialized genetic algorithm

The C. elegans connectome is a neural network wiring diagram that specifies synaptic neurotransmitters and junction types. It does not however quantify synaptic connection strengths. It is believed that measuring these must be done in live specimens, requiring emerging or yet to be developed techniques. Without the connection strengths, it is not fully known how the nematode's nervous system produces sensory-motor behaviors.

Bionet is an attempt to compute the connection strengths that produce desired sensory-motor behaviors. This is done by a hybrid genetic algorithm that trains a large space of 3000+ weights representing synapse connection strengths to perform given sensory-motor sequences. The algorithm uses both global and local optimization techniques that take advantage of the topology of the connectome. An artificial worm embodying the connectome and trained to perform sensory-motor behaviors taken from measurements of the actual C. elegans would then behave realistically in an artificial environment. This is an important step toward creating a fully functional artificial worm. Indeed, knowing the artificial weights might cast light on the actual ones.

Using the NEURON simulation tool as a fitness evaluation function, the pharyngeal neuron assembly has been trained to produce given activation patterns, reducing activation differences from more than 50% to less than 5%. Looking ahead, training worm locomotion behaviors using Movement Validation measurements as models will allow the neural network to drive the Sibernetic body model realistically.

4.4.3 Issues list

none

4.4.4 Associated Repositories

Repository	Description	Language
HeuristicWorm		C++
bionet	Artificial neural network for training C. elegans behaviors	C++

4.5 Data Collection and Representation

Contents

There is not a single data source for our simulation; in fact one of our unique challenges is coming up with new ways to work out how to integrate multiple data sets together. On this page you can read about how different dataset are used in the model.

Being an integrative model, OpenWorm utilizes different datasets, each with different file formats and interfaces to the model. There is no master representation of all the data incorporated into the model, instead our aim is to keep the model open to be able to cope with different data structures.

Consider the connectomics data. There are different useful ways to mine this data set. For example, a NetworkX representation of the connectome as a complex graph enables questions to be asked about first and second nearest neighbors of a given neuron. In contrast, an RDF semantic graph representation is useful for reading and writing annotations about multiple aspects of a neuron, such as what papers have been written about it, multiple different properties it may have such as ion channels and neurotransmitter receptors and so on. A NeuroML representation is useful for answering questions about model morphology and simulation parameters. Lastly, a Blender representation is a full 3D shape definition that can be used for calculations in 3D space.

Using these different representations separately leads to ad hoc scripting for for each representation. This presents a challenge for data integration and consolidation of information. An ongoing development of the project is to create a unified data access layer (see PyOpenWorm below), which enables different representations to become encapsulated into an abstract view. This allows the user to work with objects related to the biological reality of the worm. This has the advantage that the user can forget about which representation is being used under the hood.

Here is a list of some of the data sets that we have used so far:

- The Virtual Worm (3D atlas of C. elegans anatomy)
- The c. elegans connectome (wiring diagram of neurons)
- Cell list of c. elegans
- Ion channels used by c. elegans
- Database of Worm behavioral phenotypes

Currently our work on data collection and representation is divided among four subprojects:

- NeuroML Connectome
- Data Visualization
- PyOpenWorm Unified Data Access Layer
- Muscle cell integration

Below you can find information about each subproject, see the project's current roadmap and access the associated data repositories

A lot of data about *c. elegans* is integrated into the model. In this project, we work on what forms we should put these data in to best leverage them for building the model.

4.5.1 NeuroML Connectome

Our computational strategy to accomplish this involves first reusing the *C. elegans* connectome and the 3D anatomical map of the *C. elegans* nervous system and body plan. We have used the NeuroML standard (Gleeson et al., 2010) to describe the 3D anatomical map of the c. elegans nervous system. This has been done by discretizing each neuron into multiple compartments, while preserving its three-dimensional position and structure. We have then defined the connections between the NeuroML neurons using the c. elegans connectome. Because NeuroML has a well-defined mapping into a system of Hodgkin-Huxley equations, it is currently possible to import the "spatial connectome" into the NEURON simulator (Hines & Carnevale 1997) to perform in silico experiments.

Previous accomplishments

- Building the C Elegans NeuroML file

Current roadmap

Updated NeuroML connectome model

The NeuroML connectome model provides a framework for multi-compartmental modeling of the c. elegans nervous system. We are continuing to refine this to include more and more information that is known about the anatomy and dynamics of the nervous system in order to reach ever-improving biological realism.

- Create sample NeuroML connectome output
- Remove Glutamate_GJ etc in neuroConstruct project
- Create or reuse a NeuroML description of c. elegans motor neuron synapses

Issues list

All issues related to working with data, and doing research can be found on GitHub.

Associated Repositories

Repository	Description	Language
CElegansNeuroML	NeuroML based C elegans model, contained in a neuroConstruct project	Java
Blender2NeuroML	Conversion script to bring neuron models drawn in Blender into NeuroML format	Python
NEURONSimData	Graphing voltage data from NEURON sims of C. elegans conectome	Python

4.5.2 Data Visualization

With the ever increasing capacity to collect data about biological system, the new challenge is to understand what these dataset tell us about the system. The computational neuroscience community is developing a range of methods to extract knowledge from these datasets. One approach the accomplish this task is to represent the data visually. Our team has already produced the OpenWorm browser for web and iOS, which makes it easy to visually study the anatomy of the the worm.

Previous accomplishments

- OpenWorm browser
- OpenWorm browser iOS
- Hive Plots visualizations of connectome

Current roadmap

- Create a D3 implementation of the C. elegans connectome HivePlot

Issues list

All issues related to working with data, and doing research can be found on GitHub.

Associated Repositories

Repository	Description	Language
wormbrowser	The Worm Browser – a 3D browser of the cellular anatomy of the c. elegans	Javascript
openwormbrowser-ios	OpenWorm Browser for iOS, based on the open-3d-viewer, which was based on Google Body Browser	Objective-C
data-viz	Repository for scripts and other code items to create web-based visualizations of data in the project	Python

4.5.3 PyOpenWorm Unified Data Access Layer

We have consolidated a lot of data about the worm into a python library that creates a unified data access layer called PyOpenWorm. Documentation for PyOpenWorm *is available online <http://pyopenworm.readthedocs.org/en/latest/intro.html>_.*

Previous accomplishments

- Building the original OpenWorm database
- Initial release of PyOpenWorm

Current roadmap

- Finalize remaining issues for PyOpenWorm version alpha0.5
- Document Neuron Ion Channels: Types
- Document Ion channels: Research Claims

Issues list

All issues related to working with data, and doing research can be found on GitHub. Additionally, the PyOpenWorm project has its own issues list and a waffle board for easier observation of what is going on.

Associated Repositories

Repository	Description	Language
PyOpenWorm	Unified, simple data access library for data & facts about c. elegans anatomy	Python

4.5.4 Muscle Cell Integration

Because the muscle cell is driven both by an electrical model and a mechanical model, it is a focus of integration between different algorithms. Previously we have created a separate repository for the muscle model that is an adaptation of the work by *Boyle & Cohen, 2008 <http://www.comp.leeds.ac.uk/jboyle/JordanBoyle_files/extended.pdf>*. We have an approximately working version implemented in NEURON and are porting this to be fully NeuroML2 compliant.

Current roadmap

- Update NeuroML2 version of muscle model to match Neuron version
- Create or reuse a NeuroML description of C. elegans motor neuron synapses
- Sync channel descriptions with Muscle model standalone
- Find neuroreceptors and synaptic information for MDL08 muscle cell
- Secondary mechanical/electrophysiological muscle cell integration

Issues list

All issues related to working with the muscle model, can be found on GitHub.

Associated Repositories

Repository	Description	Language
muscle_model	model of c.elegans muscle in NEURON / Python	Python

4.6 Community Outreach

Contents

The effort to build the OpenWorm open science community is always ongoing.

- Outreach via Social Media
- Documenting our progress
- Journal clubs

You can find out more about our OpenWorm community *on another page*.

4.6.1 Previous accomplishments

- Past Journal clubs
- Media attention
- Attracting contributors
- Attracting supporters

4.6.2 Current roadmap

- Create documentation and package to allow others to play with the model optimization code
- Have a parallel Spanish version of the site
- Mention the call for C++ programmers to assist with Sibernetic on the Get Involved page

4.6.3 Issues list

All issues related to help with documentation. can be found on GitHub.

4.6.4 Associated Repositories

Repository	Description	Language
org.openworm.website	OpenWorm Website	Python
OpenWorm	Project Home repo for OpenWorm Wiki and Project-wide issues	Matlab
openworm_docs	Documentation for OpenWorm	

The project is currently laid out into six major areas shown below:

- Neuromechanical modeling with Sibernetic
- Geppetto Simulation Engine

- Movement validation
- Optimization engine
- Data collection and representation
- Community outreach

NeuroMechanical Modeling - Sibernetic

While our ultimate goal is to simulate every cell in the c. Elegans, we are starting out by building a model of its body, its nervous system, and its environment. Sibernetic is the home of the C++ code base that implements the core of the model. We have implemented an algorithm called Smoothed Particle Hydrodynamics (SPH) to simulate the body of the worm and its environment using GPUs. This algorithm has been initially worked out in C++ (with OpenGL visualization).

To get a quick idea of what this looks like, check out the latest movie. In this movie you can see a simulated 3D c. elegans being activated in an environment. Its muscles are located around the outside of its body, and as they contract, they exert forces on the surrounding fluid, propelling the body forward via undulatory thrust. In this model, the neural system is not considered and patterns of muscle contraction are explicitly defined.

More detailed information is available on the *Sibernetic project page*.

Geppetto Simulation Engine

In order to allow the world to play with the model easily, we are engineering Geppetto, an open-source modular platform to enable multi-scale and multi-algorithm interactive simulation of biological systems. Geppetto features a built-in WebGL visualizer that offers out-of-the-box visualization of simulated models right in the browser. You can read about architectural concepts and learn more about the different plug-in bundles we are working on.

More detailed information is available on the *Geppetto project page*.

Movement validation

In order to know that we are making meaningful scientific progress, we need to validate the model using information from real worms. The movement validation project is working with an existing database of worm movement to make the critical comparisons.

The main goal of the Movement Validation team is to finish a test pipeline so the OpenWorm project can run a behavioural phenotyping of its virtual worm, using the same statistical tests the Schafer lab used on their real worm data.

More detailed information is available on the *Movement validation project page*.

Optimization engine

The Optimization engine uses optimization techniques like genetic algorithms to help fill gaps in our knowledge of the electrophysiology of *C. elegans* muscle cells and neurons.

More detailed information is available on the *Optimization project page*.

Data Collection and Representation

A lot of data about *C. elegans* is integrated into the model. In this project, we work on what forms we should put these data in to best leverage them for building the model.

More detailed information is available on the *Data representation project page*.

Community Outreach

The effort to build the OpenWorm open science community is always ongoing.

More detailed information is available on the *Community project page*.

CHAPTER 5

OpenWorm Community

This page contains information intended to help individuals understand what steps to take to make contributions to OpenWorm, how to join OpenWorm meetings, how to interact with the community online, and how to become an OpenWorm core member.

Contents

5.1 An Opening Note

Feeling lost? Not uncommon in open source projects. In fact, there are whole papers describing the kinds of problems you may be having and some proposed solutions. Help us make helping you easier by reaching out to us to ask for help!

5.2 Contribution Best Practices

Once you have identified an issue you want to work on from *a particular project*, please announce your intention to helping out on the mailing list and by *commenting on the specific GitHub issue*.

5.2.1 Using OpenWorm repos on GitHub

Making a contribution of code to the project will first involve *forking one of our repositories*, making changes, committing them, creating a pull request back to the original repo, and then updating the appropriate part of documentation.

An alternate way to contribute is to create a new GitHub repo yourself and begin tackling some issue directly there. We can then fork your repo back into the OpenWorm organization at a later point in order to bring other contributors along to help you.

More details on best practices using OpenWorm repos on GitHub are available *on a separate page*.

5.2.2 Creating organizing documents

Another great way to contribute is by *organizing ideas or documentation or proposals via a Google doc*, and then sharing the link on our mailing list.

To contribute documentation and materials to the OpenWorm Google Drive, log into your Gmail account and click on this link.

All documents located in the OpenWorm folder is viewable to the public. Comments can be added to both text documents and spreadsheets. In order to edit existing documents or to add a new document, you will need to be added to the folder. You can request access by email your Google ID to info@openworm.org.

OpenWorm Docs

Taking notes as Google docs

It is very useful to create notes and progress reports as the result of meetings as Google docs. Docs should be shared publicly with view and comment access.

An effective progress report should contain the following information:

- Meeting title
- Attendees
- Date
- Goal being worked on (link back to doc page describing project)
- Previous accomplishments
- Recent progress towards goal
- Next Steps
- Future Steps

An example of an effective progress report is available online.

Once the document is shared, it should be announced on the mailing list.

Creating proposals as Google docs

To gather public comment on a direction for the project, it is often effective to create a proposal as a world-editable Google Doc. Once your document is created and shared, it should be announced on the mailing list.

An example of an effective proposal is available online

5.2.3 Contributing to the OpenWorm documentation

The OpenWorm documentation is a searchable repository of knowledge we have assembled to help new users get oriented to the different areas of the project. When new contributions are made, it is important that they are incorporated into the appropriate part of the documentation.

When they are ready to consume by the general public, simulation engines, visualization environments, and data sets should be added to the *resources page*.

Information about the goals, progress, and roadmap of current or proposed projects should be added to the *projects page*.

The docs use rst format. This kind of markup is a bit verbose and unforgiving in its syntax compared to other languages, but it is convenient for publishing documentation to the ReadTheDocs service directly from the GitHub repo, so we use it.

The 'master outline' for the top level is in index.rst. The 'toctree' directive in this file sets up what is on the sidebar. This assumes that files with the names under the toctree are present in the same directory as index.rst. Under this, the next level of hierarchy is determined by section headers. In the projects page we've used a hidden toctree in the file, which is creating the next level of hierarchy in the sidebar. In that toctree, you can see an example of referencing the underlying directory structure (e.g. 'Projects/worm-movement').

Changes that appear in GitHub will automatically trigger a hook that will cause the documentation on ReadTheDocs to become rebuilt and pushed onto the site. There are different versions of the documentation that are explained below.

OpenWorm Documentation Versions

Multiple versions of the documentation are enabled via GitHub branches. The content that appears as 'latest' online corresponds to what is in the master branch in the repo. This content should be dynamic and a space for adding stuff boldly.

The content that appears as a numbered version, like 0.5 corresponds to what is in the branch named 0.5 in the repo. This content should be considered stable and not updated lightly.

Keeping a division between latest and the versioned documentation is important for several reasons:

- *Latest* acts as a staging area - ReStructuredText is often touchy in terms of formatting – it is easy towrite something before ensuring that it formats properly. We don't want those warts exposed to the public so having an extra layer of review by checking the page on *latest* first is valuable.
- URL Stability - content in *latest* is easy to update. Pages can be moved or deleted easily, breaking URLs that we have given out. If we make sure not to move pages around on the versioned docs, we can sustain URLs
- Versions should correspond to major releases of the project as a whole, which happen approximately every six months. As the project naturally evolves, the versioned docs provide a motivation for the entire documentation to be re-evaluated as a whole.

The recommended best practice when updating the documentation is that if your changes fix bugs with the documentation that don't involve moving pages, renaming pages, or deleting pages, then check them in first to latest. Then on a regular basis the changes can be evaluated to be back applied to the most recent version. If your changes add new

projects or new content, or update a documentation page with the results of new events, keep this in latest and it will get rolled into the next version.

5.2.4 Guest Blog Post

We love hearing about what members are of the OpenWorm community are doing. If you have something to share, contact us at info@openworm.org to discuss.

5.2.5 Journal Clubs

Every few months an academic journal article comes along we can't resist talking about. We host a journal club where we invite scientists to present on the paper and to host a discussion about it, hopefully with some of the article authors.

You can see past journal clubs we have conducted online.

If you have an idea for a good journal club, please post the suggestion on our mailing list.

5.2.6 Coding Standards

It is recommended to follow the PEP8 Guidelines. For contributions of Python code to OpenWorm repositories. Compliance can be checked with the pep8 tool and autopep8

5.3 Meetings

5.3.1 Team meetings

We have a regular meeting of the team that is building applications every two weeks. We also currently schedule an ad-hoc data team meeting about every 3-4 weeks. The events are on our community calendar. The events are streamed live when they occur and an archive of the meeting videos and the minutes are kept online.

5.3.2 Working meetings

Contributors are encouraged to meet with each other on a regular basis to advance areas of the project they need interaction on.

5.3.3 IRC channel for OpenWorm

We're trying to reboot an IRC channel for OpenWorm. Check it out!

5.3.4 Scheduling meetings

We like using the Doodle service for scheduling meetings. This makes it easy to find times to meet across various time zones. Once a meeting is scheduled, we will often create a Google+ event to track it and remind everyone it is occurring.

5.4 Interactions

5.4.1 Mailing Lists

There are two Google Groups in connection with OpenWorm. We suggest joining both lists to stay current, introduce yourself to the project, and participate in ongoing discussions. Simply login with you Gmail username and click on "Join Group" for each list.

This list is for general updates and announcements related to the project.

This list is for high-volume type technical discussions, day-to-day communications, and questions related to the OpenWorm project.

5.4.2 Google Plus

Follow us on OpenWorm Google+

Click on the "Follow" button to be a part of the OpenWorm community on Google+.

If you need more help with Google+, check out the handy guide put out by Google.

5.4.3 YouTube

View our YouTube channel

Want to get notified when new content goes live? Subscribe to the channel by clicking on the "subscribe" button while logged in to your Google account.

Playlists

- Status Updates - Biweekly updates from the OpenWorm team.
- Journal Clubs - Like journal clubs that meet in person, the OpenWorm journal clubs use discuss new discoveries, tools and resources related to neuroscience, *C. elegans*, computational biology and open source science. Journal clubs are posted to social media in advance for any to watch and recordings then become available on YouTube. *Learn more about our journal clubs.*
- Data Team meetings - *Learn more about our team meetings.*
- Real *C. elegans*
- Building Blocks

5.4.4 Twitter

Follow our Twitter feed

Want to tag OpenWorm on a tweet? Use @openworm and share the love.

5.4.5 Blog

Our blog is hosted in Tumblr.

Interesting in being a guest on our blog? We love hearing about what members of the OpenWorm community are doing. If you have something to share, contact us at info@openworm.org to discuss.

5.5 Membership

More information about the membership policy is *available on a separate page*.

CHAPTER 6

Using OpenWorm Resources

This page describes content that has been created by the project for use by the public. Currently we make simulation engines, visualization environments, and data sets available.

Contents

6.1 Simulation engines

6.1.1 Sibernetic

Sibernetic is the code base that currently implements the crawling model. Sibernetic is a C++ / Python code base by Palyanov, Khayrulin and Vella that has been created expressly for the purpose of doing research and building the model quickly.

More information on running Sibernetic is available online.

The *project page for Sibernetic* has information about getting involved with its development.

6.1.2 Geppetto

Geppetto is a generic multi-algorithm integration platform written in Java and HTML5 by Cantarelli, Idili, Martinez and Khayrulin whose goal is to enable the world to play with simulations via their web browser, dramatically reducing the barrier to entry. We are currently working to port the functionality in Sibernetic into Geppetto, which would

transform the experience of seeing the model from looking at a YouTube video to being able to play and interact with the model in 3D.

More information on running Geppetto is available online.

The *project page for Geppetto* has information about getting involved in its development with OpenWorm.

6.1.3 Connectome Engine and Lego Mindstorms robot

To start getting some practical experience playing with dynamics that come from the connectome, we have simplified it into a project called the 'connectome engine' and integrated its dynamics into a Lego Mindstorms EV3 robot. You can see a movie of this in action.

This is currently in the process of being written up.

6.1.4 CyberElegans

When we first started, our team in Novosibirsk had produced an awesome prototype of a neuromechanical c. elegans model which they called 'CyberElegans'. We recently published an article about it. If you watch the movie that goes along with the prototype, you can see the basic components of the loop above in action:

Here muscle cells cause the motion of the body of the worm along the surface of its environment.

Inside the worm, motor neurons are responsible for activating the muscles, which them makes the worms move. The blue portions of the loop diagram above are those aspects that are covered by the initial prototype. We are now in the process of both adding in the missing portions of the loop, as well as making the existing portions more biologically realistic, and making the software platform they are operating on more scalable.

You can download the binary for the CyberElegans (Windows only)

This code base is not currently in active development.

6.2 Visualization Environments

6.2.1 Connectome Browser

The Connectome browser, created by the team at the Open Source Brain, is a way to explore the NeuroML connectome produced by the project. You can investigate the current settings of the dynamics of each neuron, and by clicking "selection mode" you can click on individual neurons to see their synaptic partners in 3D. This is built from the Virtual Worm Blender files
docs.google.comuc?authuser=0id=0B$_t3mQaA - HaMek5wb0trd00wVFUexport = downloadrevid = 0B_t3mQaA - HaMWk1$

6.2.2 WormBrowser (HTML5 and iOS)

Explore the c. elegans in 3D! The WormBrowser is an interactive virtual experience of browsing the C. elegans worm anatomy. This is built from the Virtual Worm Blender files
docs.google.comuc?authuser=0id=0B$_t3mQaA - HaMdkMzaUI3VWVtOG8export = downloadrevid = 0B_t3mQaA - HaMT$

Source code for the web version and an iOS version are available online. We don't currently have active development happening with either, but if you are interested in helping with the iOS code base, here's a walkthrough of how to get started with the codebase.

6.3 Data sets

6.3.1 OpenWorm Database

An web version of the OpenWorm database can be browsed online.

More information about working with the data within it and other data entities can be found *on the data representation project page*.

6.3.2 C. elegans NeuroML model in neuroConstruct

The NeuroML conversion of the Virtual Worm Blender files has been imported into a neuroConstruct project. *This page* provides instructions for obtaining the latest version of neuroConstruct, getting the latest CElegans project and generating/visualizing the cells and connections.

More information about working with the data within it and other data entities can be found *on the data representation project page*.

6.3.3 OpenWorm Spreadsheet data

We keep a publicly accessible archive of data sets that we have come across and adapted on Google Drive. We are currently in the process of consolidating these data into the OpenWorm database. More information about working with the data within it and other data entities can be found *on the data representation project page*.

CHAPTER 7

Frequently Asked Questions

Contents

7.1 OpenWorm general

7.1.1 Why C. elegans?

The tiny worm C. elegans is by far the most understood and studied animal with a brain in all of biology. It was the first multi-cellular organism to have its genome mapped. It has only ~1000 cells and exactly 302 neurons, which have also been mapped as well as its "wiring diagram" making it also the first organism to have a complete connectome produced. This part gets particularly exciting for folks interested in artificial intelligence or computational neuroscience.

Three different Nobel prizes have been awarded for work on this worm, and it is increasingly being used as a model for better understanding disease and health relevant to all organisms, including humans. When making a complex computer model, it is important to start where the data are the most complete.

7.1.2 What does the real worm do?

It has all sorts of behaviors! Some include:

- It finds food and mates
- It avoids toxins and predators
- It lays eggs
- It crawls and there are a bunch of different crawling motions

7.1.3 Do you simulate all that?

We've started from a cellular approach so we are building behavior of individual cells and we are trying to get the cells to perform those behaviors. We are starting with simple crawling. The main point is that we want the worm's overall behavior to emerge from the behavior of each of its cells put together.

7.1.4 So say the virtual organism lays eggs. Are the eggs intended to be new, viable OpenWorms, or is fertilization not a goal?

Right now we aren't addressing the egg laying or development capacity, however, the worm does have the best known developmental history of any organism so it would be really interesting to work on a computational development model.

7.1.5 Does it need to know how to be a worm to act like a worm?

The "logic" part comes from the dynamics of the neurons interacting with each other. it is a little unintuitive but that's why makes up how it "thinks". So we are simulating those dynamics as well as we can rather than instructing it what to do when. Of course that will require a good mechanical model of how CE muscles respond to stimulation.

7.1.6 Given all that we DON'T know about c. elegans (all the various synaptic strengths, dynamics, gap junction rectification, long-range neuromodulation, etc.), how do you know the model you eventually make truly recapitulates reality?

All models are wrong, some models are useful :) We must have the model make a prediction and then test it. Based on how well the model fits the available data, we can quantify how well the model recapitulates reality.

We are currently evaluating the database behind a recent paper on C. elegans behavioral analysis, which resides here, as the standard we will use to test the model's external behavior. More on this here.

As an analogy to what we are aiming for, we are inspired by the work of the Covert lab in the creation of a whole cell simulation that predicts phenotype from genotype at 80% accuracy. This is just a single cell model, but it has the same challenges of high complexity and fundamental understanding gaps that must be bridged via good assumptions.

7.1.7 Is there only one solution to all those variables in the connectome that will make a virtual c. elegans that resembles a real one, or are there multiple?

It is very likely to be multiple, given what we know about the variability of neuronal networks in general. One technique to deal with this is to generate multiple models that work and analyze them under different conditions. What we are after is the solution space that works (see Fig 6 for an example), rather than a single solution. That said, it is extremely likely that the solution space is much smaller than the complete space of possibilities.

7.1.8 Why not start with simulating something simpler? Are nematodes too complex for a first go at whole organism simulation?

Nematodes have been studied far more than simpler multi-cellular organisms, and therefore more data exist that we can use to build our model. We would need to get, for example, another connectome and another anatomical 3D map whereas in C. elegans they already exist. The community of scientists using c. elegans as their model organism is much larger than communities that studying simpler multi-cellular organisms, so the effect of the community size also weighed in on the decision.

7.1.9 When do you think the simulation will be "complete", and which behaviors would that include?

Completion is a functional standard – so it is complete when it fits all available data about worm behavior. Today, the gold standard for available data about worm behavior is encapsulated in the WormBehavior database, described here. More information from the paper.

At the moment we are focusing on integrating an electrophysiological simulation of the nervous system with a elastic matter and fluid dynamics simulation for how the body of the worm interacts with the environment. You can read more about this here

Once the simulation of the nervous system is driving the physics-enabled body of the worm around a simulated petri dish, it will be comparable to the WormBehavior database. The degree of overlap between the simulated worm and the behavior of real worms will be very interesting to see – we are very curious to find this out!

7.1.10 Currently, what are your biggest problems or needs?

To make this project move faster, we'd love more help from motivated folks. Both programmers and experimentalists. We have a lot we want to do and not enough hands to do it. People who are skeptical about mammal whole-brain simulations are prime candidates to be enthusiastic about whole-worm simulations. Read more about ways to help on our website.

7.1.11 Where I could read about your "to do's?"

We have a set of [high level milestones](https://github.com/openworm/OpenWorm/issues/milestones) for the modeling direction we are taking up on GitHub. We also have [a master board of all is-

sues](https://waffle.io/openworm/openworm) across all our GitHub repositories that show a bunch of programming tasks we are working on.

7.1.12 How do I know which issues are safe to work on? How do I know I won't be stepping on any toes of work already going on?

The high-volume mailing list is the organizing mechanism of first resort when determining these questions. If you are interested in helping with an issue but don't know if others are working on it, search on the list, and if you don't see a recent update, post on the list and ask. The mechanism of second resort is to ask a question in the comment thread of the GitHub issue. All contributors are advised to report on their effort on the mailing list or on the GitHub issue as soon as they start working on a task in order to let everyone know. As much as possible we avoid doing work that don't get exposed through one or both of these mechanisms.

In general, you won't step on any toes though – multiple people doing the same thing can still be helpful as different individuals bring different perspectives on tasks to the table.

7.1.13 Do you all ever meet up somewhere physically?

Subsets of us meet frequently, and there has been one meeting of the core OpenWorm team in Paris in July 2014. We use Google+ hangout to meet face to face virtually every two weeks.

7.2 OpenWorm simulation and modeling

7.2.1 What is the level of granularity of these models (ie. cells, subcellular, etc.), and how does that play out in terms of computational requirements?

In order to make this work we have to make use of abstraction in the computer science sense, so something that is less complex today can be swapped in for something more complex tomorrow. This is inherent in the design of the simulation engine we are building

Right now our model of the electrical activity neurons is based on the Hodgkin Huxley equations. The muscles and the physical body of the worm are governed by an algorithm known as "smoothed particle hydrodynamics." So our initial complexity estimates are based on asking how much CPU horsepower do we need for these algorithms.

7.2.2 What's the data source for your computer simulation of the living worm?

There is not a single data source for our simulation; in fact one of our unique challenges is coming up with new ways to work out how to integrate multiple data sets together. Here is a list of some of the data sets that we have used so far:

- The Virtual Worm (3D atlas of C. elegans anatomy)
- The c. elegans connectome (wiring diagram of neurons)
- Cell list of c. elegans
- Ion channels used by c. elegans
- Database of Worm behavioral phenotypes

7.2.3 Has there been previous modeling work on various subsystems illustrating what level of simulation is necessary to produce observed behaviors?

There have been other modeling efforts in C. Elegans and their subsystems, as well as in academic journal articles. However, the question of "what level of simulation is necessary" to produce observe behaviors is still an open question.

7.2.4 How are neurons simulated today?

There are a number of neuronal simulators in use, and we have done considerable amount of work on top of one in particular, the NEURON simulation environment.

There are a wide variety of ways to simulate neurons, as shown in figure two of Izhikevich 2004.

7.2.5 What does a neuronal simulator do?

It calculates a system of equations to produce a read out of the changing membrane potential of a neuron over time. Some simulators enable ion channel dynamics to be included and enable neurons to be described in detail in space (multi-compartmental models), while others ignore ion channels and treat neurons as points connected directly to other neurons. In OpenWorm, we focus on multi-compartmental neuron models with ion channels.

7.2.6 What is the connection between the basic properties of C. elegans neurons and human neurons?

C.elegans neurons do not spike (i.e. have action potentials), which makes them different from human neurons. However, the same mathematics that describe the action potential (known as the Hodgkin-Huxley model) also describe the dynamics of neurons that do not exhibit action potentials. The biophysics of the neurons from either species are still similar in that they both have chemical synapses, both have excitable cell membranes, and both use voltage sensitive ion channels to modify the electrical potential across their cell membranes.

7.2.7 What is the level of detail of the wiring diagram for the non-neuron elements?

There is a map between motor neurons and muscle cells in the published wiring diagram. There isn't much of a wiring diagram that touches other cell types beyond that. There is an anatomical atlas for where they are located. And you can work out the influence between cells based on molecular signals (known as peptides).

7.2.8 How much new electrophysiological data will the project need to achieve its goals?

We are hoping that we get neuron by neuron fast calcium imaging of a lot of neurons.

7.2.9 How will the parameters of the neurons be inferred from calcium imaging?

Basically we will use model optimization / genetic algorithms to search the parameter space for parameters that are unknown.

7.2.10 What are you using genetic algorithms in OpenWorm for?

Because there are a lot of unknowns in the model, we use genetic algorithms (or more generally model optimization techniques) to help us generate many of possible models to match experimental data and then pick the ones most likely to be correct. Here's a paper that describes a process like this.

7.2.11 What will the fitness function be?

Here are some examples

7.2.12 How do you plan to extend its methods from single neurons to multiple neurons?

This project is all about biting off small workable pieces of the problem. The plan there is to chain this method. We are starting from a muscle cell whose example electrophysiology we have. Then we will approximate the six motor neurons synapsing onto it based on what we know about its ion channels and whatever more we can gather based on calcium imaging.Then we will be exploring how to tune the combined system of the single muscle cell with the 6 motor neurons connected to it as a network and radiate outwards from there.

7.2.13 Do you need a connectome for these gap junctions as well or should an accurate enough cell model suffice?

The gap junctions are included in the C. elegans connectome.

7.2.14 What's the main differences between the single and multi-compartment models?

Single compartment models lack sufficient detail to capture the detailed shape of the neuron or muscle, which has been shown to influence the dynamics of the cell as a whole. Basically, only multi-compartment models get close enough to be comparable to real biology.

7.2.15 What is NeuroML and what does it represent?

An introduction to NeuroML is available on their website. In short, it is an XML based description of biological descriptions of neurons.

7.2.16 How is excitation and inhibition in neurons handled in OpenWorm?

Inhibition and excitation will be handled via synapses. Different neurotransmitters and receptors are encoded in our model of the nervous system. Some of those include Glutamate "excitatory" and GABA "inhibitory." We have encoded information about the neurons in the OpenWorm NeuroML spatial connectome

7.2.17 How do I run the NeuroML connectome?

Get the connectome NeuroML project that contains it and load it up in NeuroConstruct. Install the NEURON simulation environment and set the path to NEURON's bin directory containing nrniv within neuroConstruct's menu

(Settings->General Preferences and Project Defaults). After generating cell positions (easiest to do this with the PharyngealNeurons_inputs configuration), go to the export tab, the NEURON subtab, and press 'create hoc simulation'. Once this is completed the button will stop being greyed out and the 'Run simulation' button will be available. Clicking this should kick off the simulation run. Once this is completed, the output from the simulation should tell you that results are available in a directory named 'Sim_XX' where XX will be a number. Go back to the Visualisation tab and click 'View Prev Sims in 3D..." Click on the box with the 'Sim_XX' name that applies to the simulation run you did and press 'Load Simulation' at the bottom. Then at the bottom of the Visualisation screen click 'Replay' and the 'Replay simulation'. For PharyngealNeurons_inputs, the color changes will be subtle, but they will be happening.

7.2.18 I generated positions for the connectome in NeuroConstruct and tried to export to NEURON but it said NEURON was not found!

Double check that you have set the path to NEURON's **bin** directory containing nrniv within neuroConstruct's menu (Settings->General Preferences and Project Defaults). Just pointing to the root where the bin directory is located will **NOT** work.

7.2.19 How does the NemaLoad project relate to OpenWorm?

We both want to see the c. elegans reverse engineered as a means of understanding nervous systems. We've met a few times and David Darlymple contributes to the project and on the mailing list. We have a different approach right now, but they are complementary and could be unified down the road. Both projects have a lot of up front development work that we are doing now, us mainly in software and integrating data that already exists and David in building an ambitious experimental set up to collect a never-before-gathered data set.

7.2.20 What is SPH?

Smoothed Particle Hydrodynamics. More information is available online.

7.2.21 What are you doing with SPH?

We are building the body of the worm using particles that are being driven by SPH. This allows for physical interactions between the body of the worm and its environment.

7.3 OpenWorm code reuse

7.3.1 What are LEMS and jLEMS?

LEMS (Low Entropy Model Specification) is a compact model specification that allows definition of mathematical models in a transparent machine readable way. NeuroML 2.0 is built on top of LEMS and defines component types useful for describing neural systems (e.g. ion channels, synapses). jLEMS is the Java library that reads, validates, and provides basic solving for LEMS. A utility, jNeuroML, has been created which bundles jLEMS, and allows any LEMS or NeuroML 2 model to be executed, can validate NeuroML 2 files, and convert LEMS/NeuroML 2 models to multiple simulator languages (e.g. NEURON, Brian) and to other formats.

7.3.2 What is OSGi and how is it being used?

OSGi is a code framework that is at the heart of Geppetto. One of the basic underpinnings of object-oriented programming is that code modules should have low coupling– meaning that code in one part of your program and code in another part of your program should minimize calling each other. Object oriented languages like Java help to enable programs to have low coupling at compile time, but it has been recognized that in order to have true modularity, the idea of low coupling needed to be extended through to run-time. OSGi is a code framework in Java that does this. With OSGi, code modules can be turned on and off at run-time without need for recompile. This provides for an extremely flexible code base that enables individual modules to be written with minimal concern about the rest of the code base.

This matters for OpenWorm as we anticipate many interacting modules that calculate different biological aspects of the worm. So here, each algorithm like Hodgkin Huxley or SPH can be put into an OSGi bundle in the same way that future algorithms will be incorporated. Down the road, this makes it far more likely for others to write their own plugin modules that run within Geppetto.

7.3.3 What is Spring and how is it being used?

Spring is a code framework being used at the heart of Geppetto. It enables something called 'dependency injection', which allows code libraries that Geppetto references to be absent at compile time and called dynamically during run-time. It is a neat trick that allows modern code bases to require fewer code changes as the libraries it depends on evolves and changes. It is important for Geppetto because as it increasingly relies on more external code libraries, managing the dependencies on these needs to be as simple as possible.

7.3.4 What is Tomcat and how is it being used?

Tomcat is a modern web server that enables Java applications to receive and respond to requests from web browsers via HTTP, and Geppetto runs on top of this. It has no OSGi functionality built into it by itself, that's what Virgo adds.

Geppetto implements OSGi via a Virgo server which itself runs on top of Tomcat. It is a little confusing, but the upshot is that Geppetto avoids having to build components like a web server and focus only on writing code for simulations.

7.3.5 What is Virgo and how is it being used?

Virgo is a web server that wraps Tomcat and uses OSGi as its core framework, and Geppetto runs on top of this. On top of the code modularity framework that OSGi provides, Virgo adds the ability to receive and respond to requests from web browsers via HTTP. It is important for Geppetto because it is a web-based application.

7.3.6 What is Maven and how is it being used?

Maven is a dependency management and automated build system for Java that is used by Geppetto to keep track of all the libraries it uses. If you are familiar with Make files, Maven provides a more modern equivalent in the form of a project object model file, or pom.xml. Whereas Spring is a library that appears in source code, Maven operates external to a code base, defining how code will get built and what libraries will be used. Maven enables external code libraries to be downloaded from the internet upon run time, which helps to avoid the bad programming practice of checking all your libraries into version control repositories.

It is important for OpenWorm because as Geppetto increasingly relies on other code libraries, we need easy ways to manage this.

7.4 OpenWorm links and resources

7.4.1 Do you have a website?

http://openworm.org

7.4.2 Where can I send my inquiries about the project?

info@openworm.org

7.4.3 Where can I find the "worm browser"?

http://browser.openworm.org

7.4.4 How do I join the public mailing list?

More info here: http://www.openworm.org/contacts.html

7.4.5 Where are downloads located?

http://www.openworm.org/downloads.html