# Chapter 1

## A tour of the NEURON simulation environment

## Modeling and understanding

Modeling can have many uses, but its principal benefit is to improve understanding. The chief question that it addresses is whether what is known about a system can account for the behavior of the system. An indispensable step in modeling is to postulate a *conceptual model* that expresses what we know, or think we know, about a system, while omitting unnecessary details. This requires considerable judgment and is always vulnerable to hindsight and revision, but it is important to keep things as simple as possible. The choice of what to include and what to leave out depends strongly on the hypothesis that we are studying. The issue of how to make such decisions is outside the primary focus of this book, although from time to time we may return to it briefly.

The task of building a *computational model* should only begin after a conceptual model has been proposed. In building a computational model we struggle to establish a match between the conceptual model and its computational representation, always asking the question: would the conceptual model behave like the simulation? If not, where are the errors? If so, how can we use NEURON to help understand why the conceptual model implies that behavior?

## Introducing NEURON

NEURON is a simulation environment for models of individual neurons and networks of neurons that are closely linked to experimental data. NEURON provides tools for conveniently constructing, exercising, and managing models, so that special expertise in numerical methods or programming is not required for its productive use. Applications of NEURON in research and education include << update list >>

In the following pages we introduce NEURON by going through the development of a simple model from start to finish. This will require us to consider each of these steps:

1. State the question that we are interested in
2. Formulate a conceptual model
3. Implement the model in NEURON
4. Instrument the model, i.e. attach signal sources and set up graphs
5. Set up controls for running simulations
6. Save the model with instrumentation and run controls

7. Run simulation experiments
8. Analyze results

Since our aim is to provide an overview, we have chosen a simple model that illustrates just one of NEURON's strengths: the convenient representation of the spread of electrical signals in a branched dendritic architecture. We could do this by writing instructions in NEURON's programming language `hoc`, but for this example we will employ some of the tools that are provided by its graphical user interface. Later chapters examine `hoc` and the graphical tools for constructing models and managing simulations in more detail, as well as many other features and applications of the NEURON simulation environment (e.g. complex biophysical mechanisms, neural networks, analysis of experimental data, model optimization, customization of the user interface).

## 1. State the question

The scientific issue that motivates the design and construction of this model is the question of how synaptic efficacy is affected by synaptic location and the anatomical and biophysical properties of the postsynaptic cell. This has been the subject of many experimental and theoretical studies [Andreasen, 1998 #359][Bernander, 1994 #360][Carnevale, 1997 #228][Cook, 1999 #327][Hoffman, 1997 #264][Jack, 1983 #90][Jaffe, 1999 #322][Magee, 1999 #357][Rall, 1989 #200][Schwindt, 1997 #358][Spruston, 1994 #77][Stuart, 1998 #301].

## 2. Formulate a conceptual model

Most neurons have many branches with irregularly varying diameters and lengths (Fig. 1.1 A), and their membranes are populated with a wide assortment of ionic channels that have different ionic specificities, kinetics, dependence on voltage and second messengers, and spatial distributions. Scattered over the surface of the cell may be hundreds or thousands of synapses, some with a direct effect on ionic conductances (which may also be voltage−dependent) while others act through second messengers. Synapses themselves are far from simple, often displaying stochastic and use−dependent phenomena that can be quite prominent, and frequently being subject to various pre− and postsynaptic modulatory effects. Given all this complexity, we might well ask if it is possible to understand anything without understanding everything. From the very onset we are forced to decide what to include and what to omit.

Suppose we are already familiar with the predictions of the basic ball and stick model [Rall, 1977 #108][Jack, 1983 #90], and that experimental observations motivate us to ask questions such as: How do synaptic responses observed at the soma vary with synaptic location if dendrites of different diameters and lengths are attached to the soma? What happens if some parts of the cell have active currents, while others are passive? What if a neuromodulator or shift of the background level of synaptic input changes membrane conductance?

Then our conceptual model might be similar to the one shown in Fig. 1.1 B. This model includes a neuron with a soma that gives rise to an axon and two dendritic trunks, and a single excitatory synapse that may be located at any point on the cell.

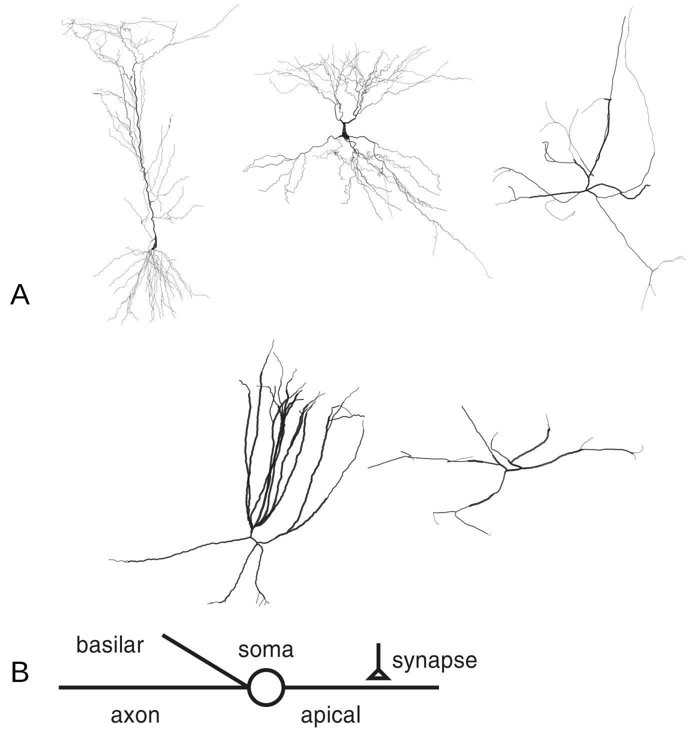


Figure 1.1. A. Clockwise from top left: Ca1 and Ca3 pyramidal neurons (from D.A. Turner); calbindin−, parvalbumin−, and calretinin−positive interneurons (from A.I. Gulyás). B. Our conceptual model neuron. The conductance change synapse can be located anywhere on the cell.

Although deliberately more complex than the prototypical ball and stick, the anatomical and biophysical properties of our model are much simpler than the biological original (Table 1.1). The axon and dendrites are simple cylinders, with uniform diameters and membrane properties along their lengths. The dendrites are passive, while the soma and axon have Hodgkin−Huxley sodium, potassium, and leak currents, and are capable of generating action potentials [Hodgkin, 1952 #129]. A single synaptic activation causes a localized transient conductance increase with a time course described by an alpha function

$$g_s(t) = \begin{cases} 0 \quad for \quad t < t_{act} \\ g_{max} \dfrac{(t - t_{act})}{\tau_s} \, e^{-\frac{(t-t_{act})}{\tau_s}} \quad for \quad t \ge t_{act} \end{cases} \qquad \text{Eq. 1.1}$$

where $t_{act}$ is the time of synaptic activation, and $g_s$ reaches a peak value of $g_{max}$ at $t = \tau_s$ (see Table 1.2 for numeric values of parameters). This conductance−increase mechanism is just slightly more complex than the ideal current sources used in many theoretical studies [Jack, 1983 #90][Rall, 1977 #108], but it is still only a pale imitation of any real synapse [Bliss, 1973 #364][Castro−Alamancos, 1997 #367][Ito, 1989 #363][Thomson, 1997 #366].

**Table 1.1. Model cell parameters**

| | **Length µm** | **Diameter µm** | **Biophysics** |
|---|---|---|---|
| **soma** | 30 | 30 | HH $g_{Na}$, $g_K$, and $g_{leak}$ |
| **apical dendrite** | 600 | 1 | passive with Rm = 5,000 Ω $cm^2$, $E_{pas}$ = −65 mV |
| **basilar dendrite** | 200 | 2 | same as apical dendrite |
| **axon** | 1000 | 1 | same as soma |

Cm = 1 µf / $cm^2$

cytoplasmic resistivity = 100 Ω cm

Temperature = 6.3 ºC

**Table 1.2. Synaptic mechanism parameters**

| | |
|---|---|
| $g_{max}$ | 0.05 µS |
| $\tau_s$ | 0.1 ms |
| $E_s$ | 0 mV |

## 3. Implement the model in NEURON

With a clear picture of our model in mind, we are ready to express it in the form of a computational model. We could do this by writing instructions in `hoc`, NEURON's programming language, but for this example we will employ some of the tools that are provided by NEURON's graphical user interface.

We begin with the CellBuilder, a graphical tool for constructing and managing models of individual neurons. At this stage, we are not considering synapses, stimulating electrodes, or simulation controls. Instead we are focussing on creating a representation of the continuous properties of the cell. Even if we were not using the CellBuilder but instead were developing our model entirely with `hoc` code, it would probably be best for us to follow a similar approach, i.e. specify the biological attributes of the model cell separately from the specification of the instrumentation that we will use to exercise the model. This is an example of the programming strategy of "divide and conquer," in which a large and complex problem is broken into smaller, more tractable steps.

The CellBuilder makes it easier for us to create a model of a neuron by allowing us to specify its architecture and biophysical properties through a graphical interface. When we are satisfied with the specification, the CellBuilder will generate the `hoc` code for us. Once we have a model cell, we will be ready to use other graphical tools to attach a synapse to it and plot simulation results (see **4. Instrument the model** below).

The images in the following discussion were obtained under MSWindows; the appearance of NEURON under UNIX, Linux, and MacOS is quite similar.

### *Start NEURON and bring up a* CellBuilder

To start NEURON under UNIX or Linux, just type `nrngui` on the command line and skip the remainder of this paragraph. Under MSWindows, bring up the NEURON program group (i.e. use Start / Program Files / NEURON) and select the nrngui item (Fig. 1.2 A). If you are using MacOS, open the folder where you installed NEURON and double click on the neuron application (Fig. 1.2 B).

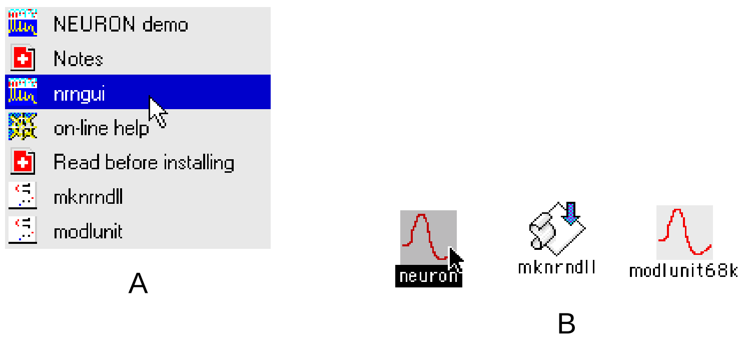


Figure 1.2. Starting NEURON from MSWindows (A) and MacOS (B).

You should now see the NEURON Main Menu (Fig. 1.3), which offers a set of menus for bringing up graphical tools for creating models and running simulations. To get a CellBuilder just click on Build, scroll down to the CellBuilder item, and release the mouse button.

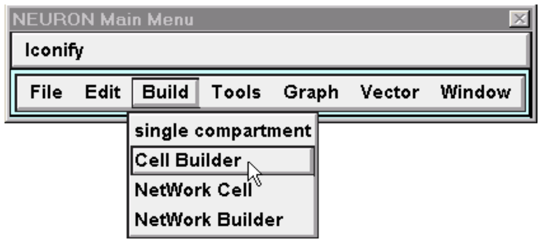


Figure 1.3. Using the NEURON Main Menu to bring up a CellBuilder.

Across the top of the CellBuilder is a row of radio buttons and a checkbox, which correspond to the sequence of steps involved in building a model cell (Fig. 1.4). Each radio button brings up a different page of the CellBuilder, and each page provides a view of the model plus a graphical interface for defining properties of the model. The first four pages (Topology, Subsets, Geometry, Biophysics) are used to create a complete specification of a model cell. On the Topology page, we will set up the branched architecture of the model and give a name to each branch, without regard to diameter, length, or biophysical properties. We will deal with length and diameter on the Geometry page, and the Biophysics page is where we will define the properties of the membrane and cytoplasm of each of the branches.

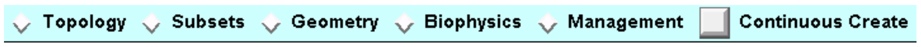


Figure 1.4. Top panel of the CellBuilder

The Subsets page deserves special comment. In almost every model that has more than one branch, two or more branches will have at least some biophysical attributes that are identical, and there are often significant anatomical similarities as well. Furthermore, we can almost always apply the d_lambda algorithm for compartmentalization throughout the entire cell (see below). We can take advantages of such regularities by assigning shared properties to several branches at once. The Subsets page is where we group branches into subsets, on the basis of shared features, with an eye to exploiting these commonalities on the Geometry and Biophysics pages. This allows us to create a model specification that is compact, efficient, and easily understood.

***Enter the specifications of the model cell***

● Topology

We start by using the Topology page to set up the branched architecture of the model. As Fig. 1.5 shows, when a new CellBuilder is created, it already contains a branch ("section") that will serve as the root of the branched architecture of the model. This root section is initially called "soma," but we can rename it if we desire (see below).

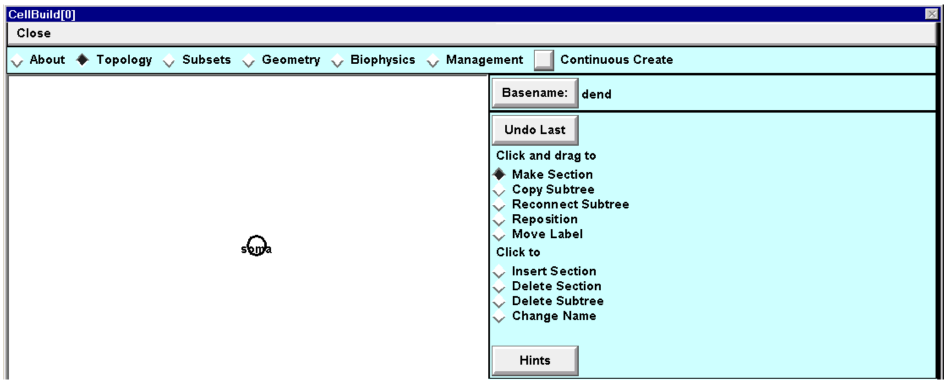
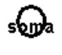


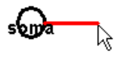
Figure 1.5. The Topology page. The left panel shows a simple diagram of the model, which is called a "shape plot." The right panel contains many functions for editing the branched architecture of a model cell.

The Topology page offers many functions for creating and editing individual sections and subtrees. We can make the section that will become our apical dendrite by following the steps presented in Fig. 1.6. Repeating these actions a couple more times (and resorting to functions like Undo Last, Reposition, and Delete Section as needed to correct mistakes) gives us the basilar dendrite and axon.

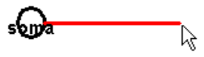
Figure 1.6. Making a new section. Verify that the Make Section radio button is *on*, and then perform the following steps.



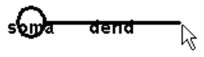
Place the cursor near one end of an existing section.



Click to start a new section. One end of the new section will automatically attach to the nearest end of an existing section; the other end is tethered to the cursor while the mouse button is held down.



Drag to the desired length and orientation.



Release the mouse button.

Our model cell should now look like Fig. 1.7. At this point some minor changes would improve its appearance: moving the labels away from the sections so they are easier to read (Fig. 1.8), and then renaming the apical and basilar dendrites and the axon (Figs. 1.9 and 10). The final result should resemble Fig. 1.11.

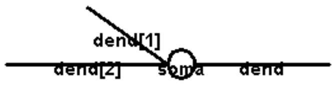
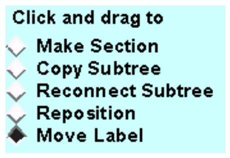
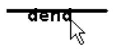


Figure 1.7. The model after all sections have been created.

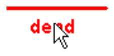
Figure 1.8. To change the location of a label,



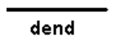
click on the Move Label radio button,



then click on the label,

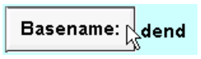


drag it to its new position,

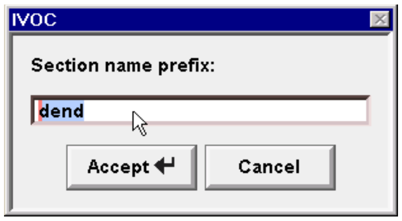


and release the mouse button.

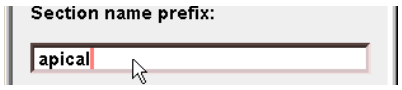
Figure 1.9. Preparing to change the name of a section. Each section we created was automatically given a name based on "dend." To change these names, we must first change the base name as shown here.



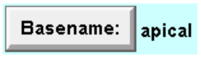
Click the Basename button.



This pops up a Section name prefix window.

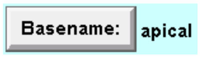


Click inside the text entry field of this new window, and type the desired name. It is important to keep the mouse cursor inside the text field while typing; otherwise keyboard entries may not have an effect.

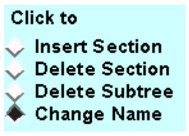


After the new base name is complete, click on the Accept button. This closes the Section name prefix window, and the new base name will appear next to the Basename button.

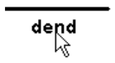
Figure 1.10. Changing the name of a section.



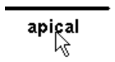
First make sure that the base name is what you want; if not, change the base name (see Fig. 1.9).



Click the Change Name radio button.



Place the mouse cursor over the section whose name is to be changed.



Click the mouse button to change the name of the section.

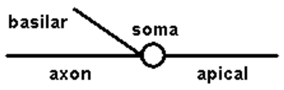


Figure 1.11. The shape plot of the model with labels positioned and named as desired.

## ● Subsets

As mentioned above, The Subsets page (Fig. 1.12) is for grouping sections that share common features. Well−chosen subsets can save a lot of effort later by helping us create very compact specifications of anatomical and biophysical properties.

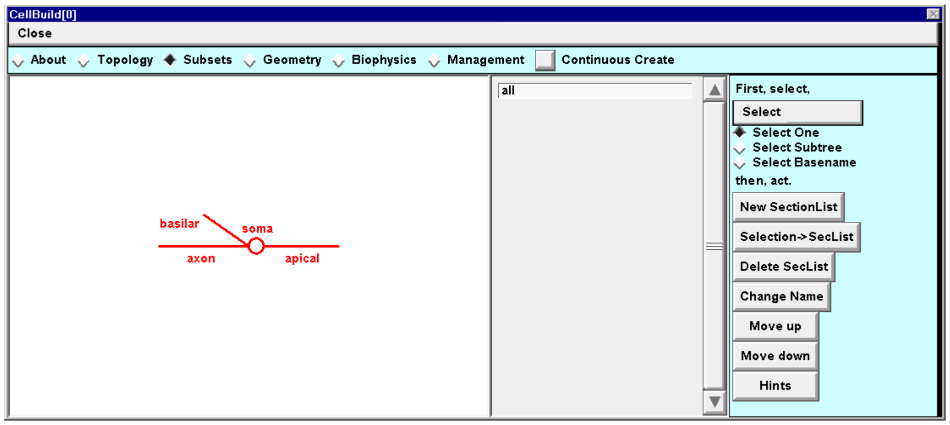
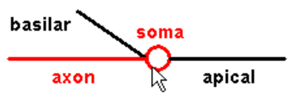


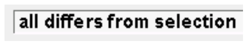
Figure 1.12. The Subsets page. The middle panel lists the names of all existing subsets. In the shape plot, the sections that belong to the currently selected subset are shown in red. When the Subsets page initially appears, it already has an all subset that contains every section in the model.

The properties of the sections in this particular example suggest that we create two subsets: one that contains the basilar and apical branches, which are passive, and another that contains the soma and axon, which have Hodgkin−Huxley spike currents. To make a subset called has_HH that contains the sections with HH currents, follow the steps in Fig. 1.13. Then make another subset called no_HH that contains the basilar and apical dendrites.

Figure 1.13. Making a new subset.

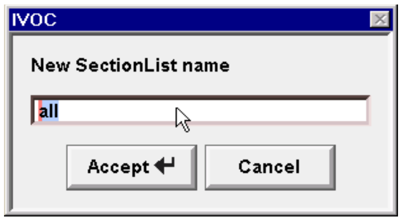


With the Select One radio button on (Fig. 1.12), click on the axon and soma sections while holding down the shift key. The selected sections will be indicated in red . . .

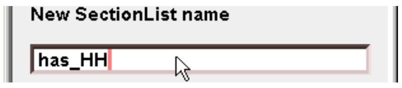


. . . and the list of subsets will change to show that all is not the same as the set {axon, soma}.

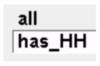
Next, click on the New SectionList button (a subset is a list of sections).



This pops up a window that asks you to enter a name for the new SectionList.



Click inside the text entry field of this new window and type the name of the new subset, then click on the Accept button.



The new subset name will appear in the middle panel of the CellBuilder.

- Geometry

In order to use the Geometry page (Fig. 1.14) to specify the anatomical dimensions of the sections and the spatial resolution of our model, we must first set up a strategy for assigning these properties. After we have built our (hopefully efficient) strategy, we will give them specific values.

The geometry strategy for our model is simple. Each section has different dimensions, so the length and diameter of each section must be entered individually. However, for each section we will let NEURON decide how fine to make the spatial grid, based on a fraction of the length constant at 100 Hz (spatial accuracy and NEURON's tools for adjusting the spatial grid are discussed in **Chapter 5**). Figure 1.15 shows how to set up this strategy.

Having set up the strategy, we are ready to assign the geometric parameters (see Figs. 1.16 and 17).

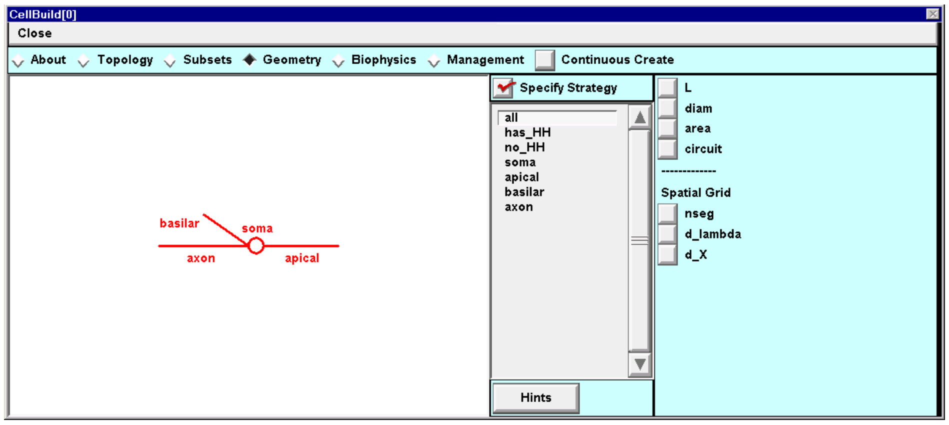
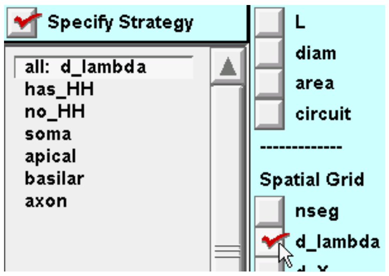
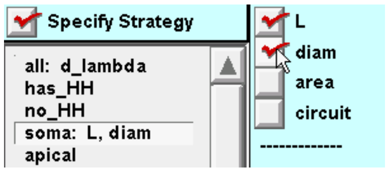


Figure 1.14. When the Geometry page in a new CellBuilder is first viewed, a red check mark should appear in the Specify Strategy checkbox. If not, clicking on the checkbox will toggle Specify Strategy *on*.

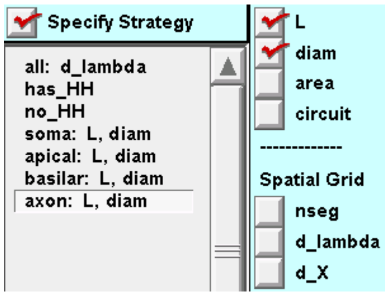
Figure 1.15. Specifying strategy for assignment of geometric parameters. First make sure that Specify Strategy contains a red check (see Fig. 1.14). Then proceed with the following steps.



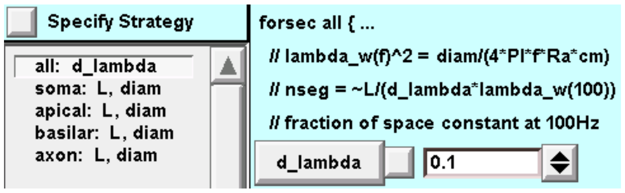
For the all subset, toggle d_lambda *on*.



Select soma in the middle panel, and then toggle L and diam *on*.



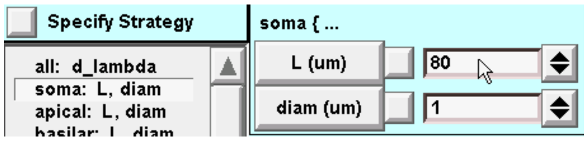
Repeat for apical, basilar, and axon, and the result should resemble this figure.



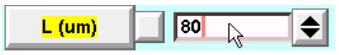
```
forsec all { ...
 // lambda_w(f)^2 = diam/(4*PI*f*Ra*cm)
 // nseg = ~L/(d_lambda*lambda_w(100))
 // fraction of space constant at 100Hz
```

Figure 1.16. Assigning values to the geometric parameters. Toggling Specify Strategy *off* makes the middle panel show only the subsets and sections that we selected when setting up our strategy. Adjacent to each of these are the names of the parameters that are to be reviewed and perhaps changed. Here the subset all is selected; the right panel displays the current value of the parameter associated with it (d_lambda) and offers us the means to change this parameter if necessary. According to the d_lambda criterion for spatial resolution, NEURON will automatically discretize the model, breaking each section into compartments small enough that none will be longer than d_lambda at 100 Hz. The default value of d_lambda is 0.1, i.e. 10% of a length constant. This is short enough for most purposes, so we do not need to change it. Discretization is discussed in **Chapter 5**.

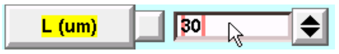
Figure 1.17. Assigning values to the geometric parameters *continued*.



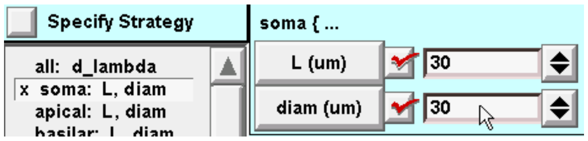
The default length and diameter of a new section are 80 and 1 μm, respectively.



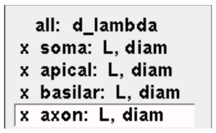
To set the length of the soma to 30 μm, first click inside the numeric field for L so that a red editing cursor appears.



Then use the backspace key to delete the old value, and finally type in the new value.



After doing the same for diam, the dimensions of the soma should look like this. The checkboxes adjacent to the L and diam buttons indicate that these parameters have been changed from their default values. The x in the middle panel is another reminder that at least one of the parameters associated with soma has been changed.



After adjusting L and diam for the dendrites and the axon, the middle panel shows an x next to the name of each section.

## ● Biophysics

The Biophysics page (Fig. 1.18) is used to insert biophysical properties of membrane and cytoplasm (e.g. Ra, Cm, ion channels, buffers, pumps) into subsets and individual sections. As with the Geometry page, first we set up our strategy (Fig. 1.19), and then we review and adjust parameter values (Fig. 1.20). The CellBuilder will then contain a complete specification of our model.

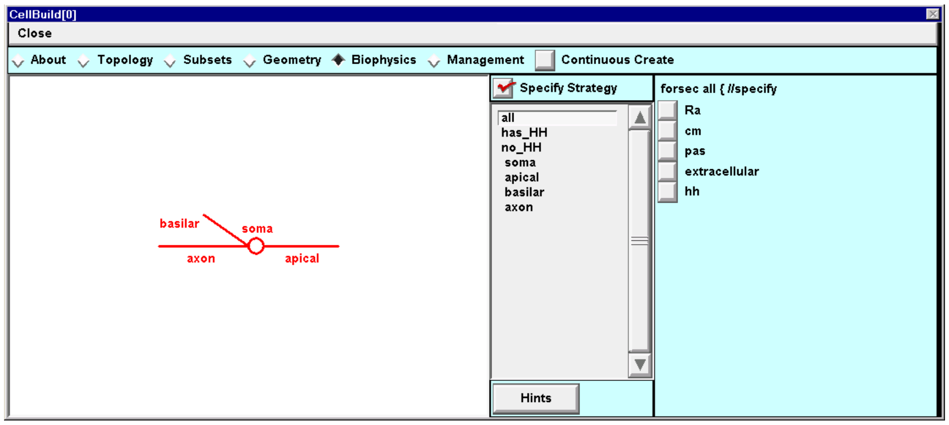
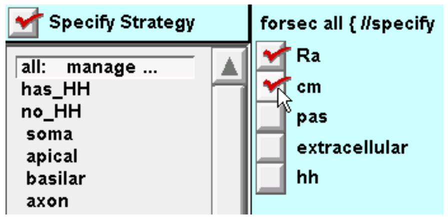


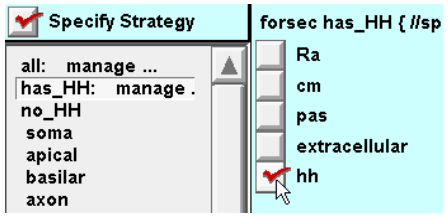
Figure 1.18. The Biophysics page, ready for specification of strategy. The right panel shows the mechanisms that are available to be inserted into our model.

For this simple example, the number of mechanisms is deliberately small; later chapters show how to expand NEURON's library of biophysical mechanisms.

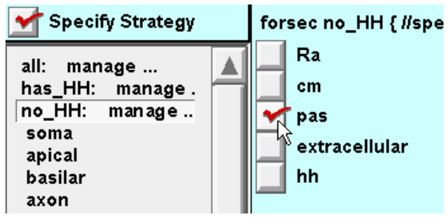
Figure 1.19. Specifying strategy for assignment of biophysical parameters. First make sure that Specify Strategy contains a red check, then proceed with the following steps.



For the all subset, toggle Ra (cytoplasmic resistivity) and cm (specific membrane capacitance) *on*.

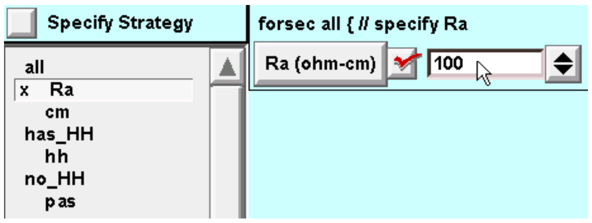


Select the has_HH subset in the middle panel, and then toggle HH *on*.



Finally select the no_HH subset and toggle pas *on*.

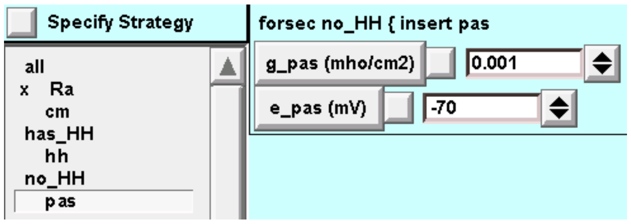
Figure 1.20. Assigning values to the biophysical parameters. Toggling Specify Strategy *off* shows a list of the names of the subsets that are part of the strategy. Beneath each subset are the names of the mechanisms that are associated with it. Clicking on a mechanism brings up a set of controls in the right panel for displaying and adjusting the parameters of the mechanism.



forsec all { // specify Ra

Ra (ohm-cm) 100

For the subset all, change the value of Ra from its default (80 Ω cm) to the desired value of 100 Ω cm.

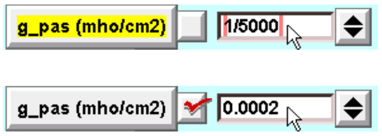


forsec no_HH { insert pas

g_pas (mho/cm2) 0.001

e_pas (mV) -70

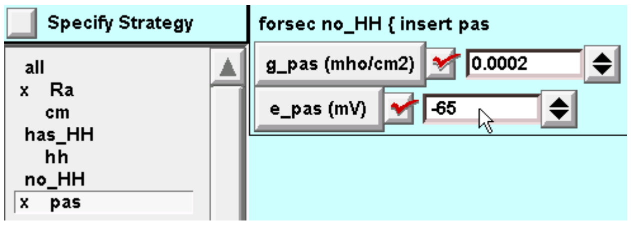
The sections in the no_HH subset have a passive current whose parameters must be changed from their defaults (shown here).



g_pas (mho/cm2) 1/5000

g_pas (mho/cm2) 0.0002

The value of g_pas can be set by deleting the default and then typing 1/5000 ( = 1/Rm).



forsec no_HH { insert pas

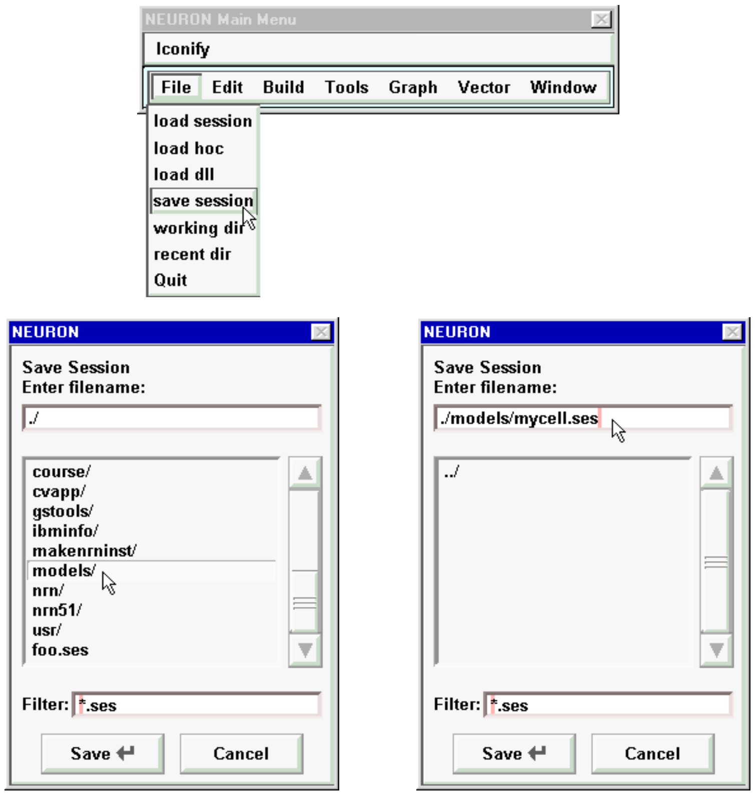
g_pas (mho/cm2) 0.0002

e_pas (mV) -65

The final values of g_pas and e_pas. Not shown: cm (all subset) and the parameters of the hh mechanism (has_HH subset), which have the desired values by default and do not need to be changed, although it is good practice to review them.

### *Save the model cell*

After investing time and effort to set up our model, we would be wise to take just a moment to save it. The CellBuilder, like NEURON's other graphical windows, can be saved to disk as a "session file" (Fig. 1.21) for future re−use (Fig. 1.22). Further information about saving and retrieving session files is presented in **Chapter X**.



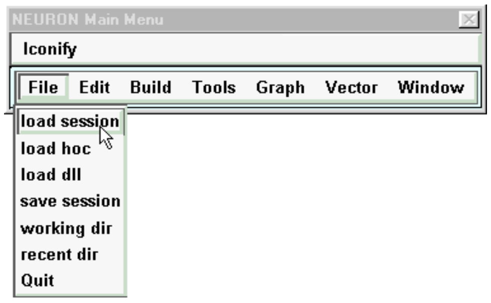
./

*.ses

./models/mycell.ses

*.ses

Figure 1.21. Top: To save all of NEURON's graphical windows into a session file, first click on File in the NEURON Main Menu and scroll down to save session. Bottom left: This brings up a directory browser that can be used to navigate to the directory where the session file will be saved. Bottom right: Click in the edit field at the top of the directory browser and type the name to use for the session file, then click on the Save button.



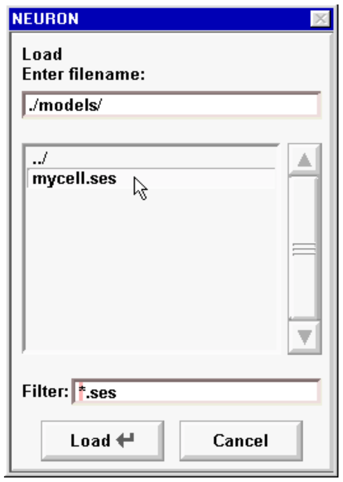


Figure 1.22. Left: To recreate the graphical windows that were saved to a session file, first click on File in the NEURON Main Menu and scroll down to load session. Right: Use the directory browser that appears to navigate to the directory where the session file was saved. Then double click on the session file that you want to retrieve.

## *Execute the model specification*

Now that the CellBuilder contains a complete specification of the model cell, we could use the Export button on the Management page (see **Chapter 6**) to write out a `hoc` file that, when executed by NEURON, would create the model. However, for this example we will just turn Continuous Create *on* (Fig. 1.23). This makes the CellBuilder send its output directly to NEURON's interpreter without bothering to write a `hoc` file. The model cell whose specifications are contained in the CellBuilder is now available to be used in simulations.

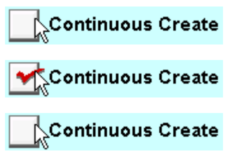


Figure 1.23. Continuous Create is initially *off*, but clicking on the adjacent button toggles it *on* and *off*.

If we make any changes to the model while Continuous Create is *on*, the CellBuilder will automatically send new code to the interpreter. This can be very convenient during model development, since it allows us to quickly examine the effects of any change. Automatic updates might bog things down if we were dealing with a large model on a slow machine. In such a case, we could just turn Continuous Create *off*, make whatever changes were necessary, and then cycle it *on* and *off* again.

## 4. Instrument the model

### Signal sources

In the NEURON simulation environment, a synapse or electrode for passing current (current clamp or voltage clamp) is represented by a point source of current which is associated with a localized conductance. These signal sources are called "point processes" to distinguish them from properties that are distributed over the cell surface (e.g. membrane capacitance, active and passive ionic conductances) or throughout the cytoplasm (e.g. buffers), which are called "distributed mechanisms" or "density mechanisms."

We have already seen how to use one of NEURON's graphical tools for dealing with distributed mechanisms (the CellBuilder). To attach a synapse to our model cell, we turn to one of NEURON's tools for dealing with point processes: the PointProcessManager (Fig. 1.24). Using a PointProcessManager we can specify the type and parameters of the point process (Fig. 1.25) and where it is attached to the cell.

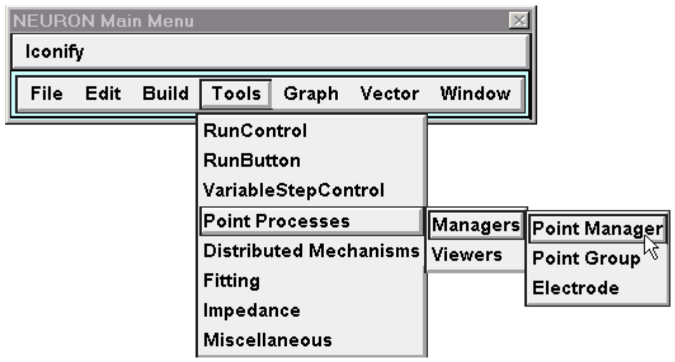
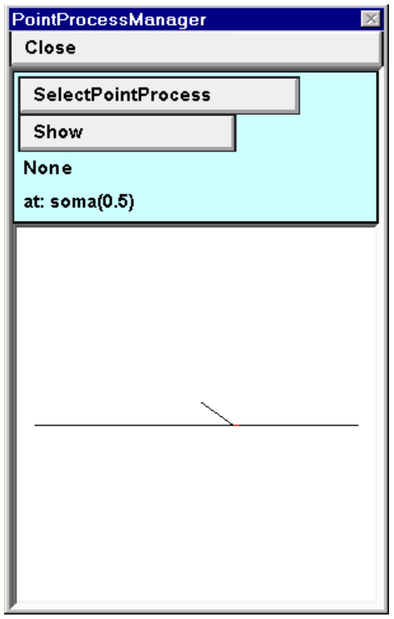
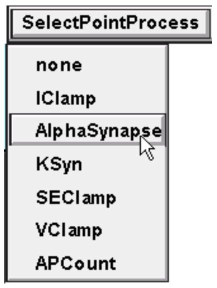


Figure 1.24. Bringing up PointProcessManager in order to attach a synapse to our model cell. In the NEURON Main Menu, click on Point Processes / Managers / Point Manager, then proceed as shown in Fig. 1.25.

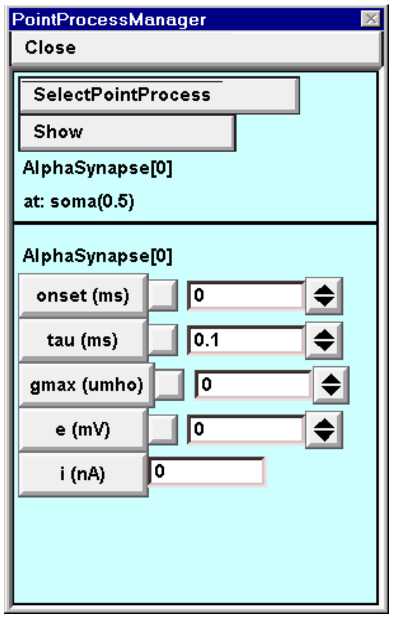
Figure 1.25. Configuring a new PointProcessManager to emulate a synapse.



A. Note the labels in the top panel. None means that a signal source has not yet been created. The bottom panel shows a stick figure of our model cell.



B. SelectPointProcess / AlphaSynapse creates a point process that emulates a synapse with a conductance change governed by Eq. 1.1, and shows us a panel for adjusting its parameters.

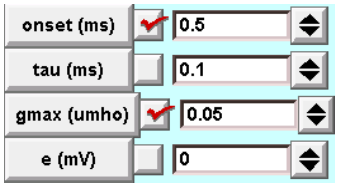


AlphaSynapse[0]
at: soma(0.5)

AlphaSynapse[0]

onset (ms) 0
tau (ms) 0.1
gmax (umho) 0
e (mV) 0
i (nA) 0

C. The top panel of the PointProcessManager indicates what kind of point process has been specified, and where it is located (in this case, at the midpoint of the soma). The bottom panel shows the parameters of an AlphaSynapse: its start time onset and time constant tau ($t_{act}$ and $\tau_s$ in Eq. 1.1), peak conductance gmax ($g_{max}$ in Eq. 1.1), and reversal potential e ($E_s$ in Table 1.2). The button marked i (nA) is just a label for the adjacent numeric field, which displays the instantaneous synaptic current.



|  |  |  |
|---|---|---|
| onset (ms) | ✓ | 0.5 |
| tau (ms) |  | 0.1 |
| gmax (umho) | ✓ | 0.05 |
| e (mV) |  | 0 |

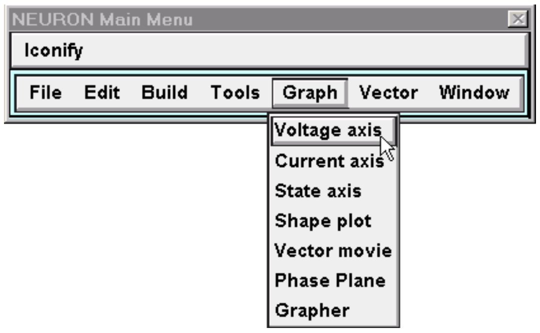
D. For this example change onset to 0.5 ms and gmax to 0.05 µS; leave tau and e unchanged.

## Signal monitors

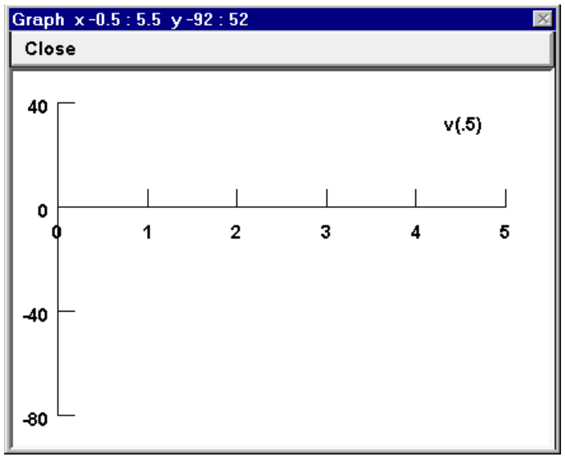
Since one motivation for the model is to examine how synaptic responses observed at the soma vary with synaptic location, we want a graph that shows the time course of somatic membrane potential. In the laboratory this would ordinarily require attaching an electrode to the soma, so in a NEURON simulation it might seem to require a point process. However, the computer automatically evaluates somatic $V_m$ in the course of a simulation. In other words, graphing $V_m$ doesn't really change the system, unlike attaching a signal source, which adds new equations to the system. This means that a point process is not needed; instead, we just bring up a graph that includes somatic $V_m$ in the list of variables that it plots (see Fig. 1.26).

We could monitor $V_m$ at other locations by adding more variables to this graph, and bring up additional graphs if this one became too crowded. However, it can be more informative and convenient to create a Space Plot (Fig. 1.27), which shows $V_m$ as a function of position along one or more branches of a cell. This graph will change throughout the simulation run, displaying the evolution of $V_m$ as a function of space and time.

Figure 1.26. Creating a graph to display somatic membrane potential as a function of time.

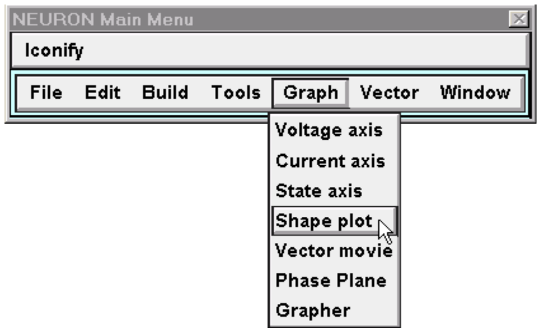


A. Click on New Graph / Voltage axis in the the NEURON Main Menu.

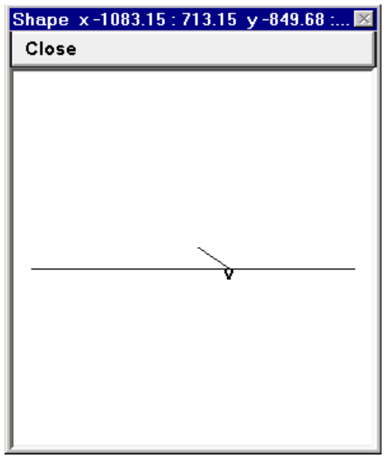


B. In the graph that appears, the horizontal axis is in milliseconds and the vertical axis is in millivolts. The label v(.5) signifies that this graph will show $V_m$ at the middle of the default section. With the CellBuilder, this is always the root section, which in this example is the soma (the concepts of "root section" and "default section" are discussed in **Chapter 5**; features of graph windows are presented in **Chapter Z**).

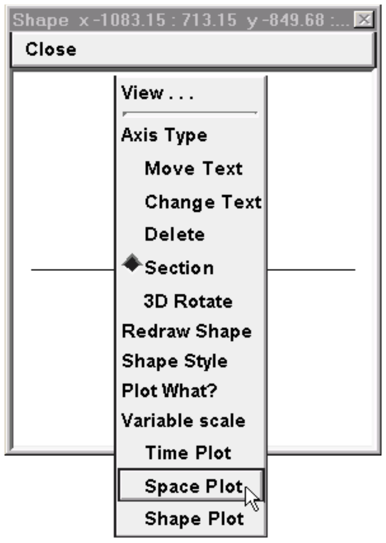
Figure 1.27. Setting up a **Space Plot**.



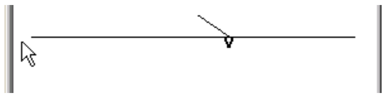
A. The first step is to click on New Graph / Shape plot in the NEURON Main Menu.



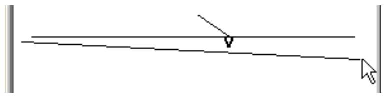
B. This brings up a Shape plot window, which is used to create the Space Plot.



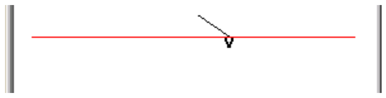
C. Right click in the Shape plot window to bring up the primary graph menu. While still pressing the mouse button, scroll down the menu to the Space Plot item, then release the button.



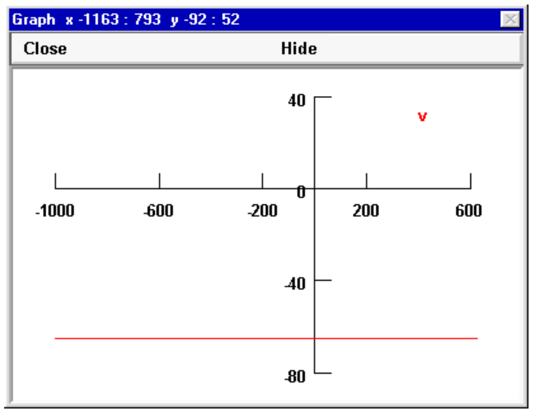
D. Place the cursor just to the left of the distal end of the axon and press the left mouse button.



E. While still holding the button down, drag the cursor across the window to the right, finally releasing the button when the cursor has passed the distal end of the apical dendrite.



F. The branches along the selected path (axon, soma, and apical dendrite) are now shown in red, and a new graph window appears (see G). If you like, you may now click on the Close button at the upper left corner of the Shape plot window to conserve screen space



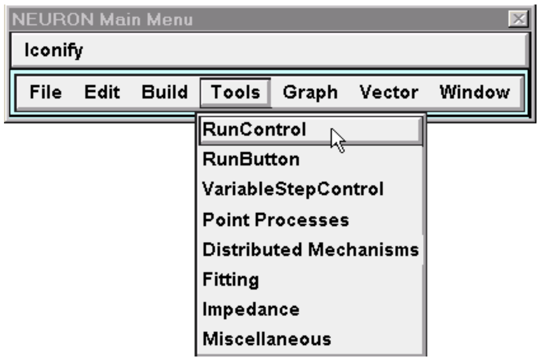
G. The x axis of the Space Plot window shows the distance from the 0 end of the default section, which in this example is the left end of the soma.

## 5. Set up controls for running the simulation

At this point we have a model cell with a synapse attached to the soma, and a graphical display of somatic $V_m$. All that is missing is a means to start and control the subsequent course of a simulation run. This is provided by the RunControl window (Fig. 1.28), which allows us to specify many more options than we will use in this example.

## 6. Save model with instrumentation and run control

After rearranging the RunControl, PointProcessManager, and graph window, our user interface for running simulations and observing simulation results should look something like Fig. 1.29. For the sake of safety and possible future convenience, it is a good idea to use NEURON Main Menu / File / Save Session to save this interface to a session file.



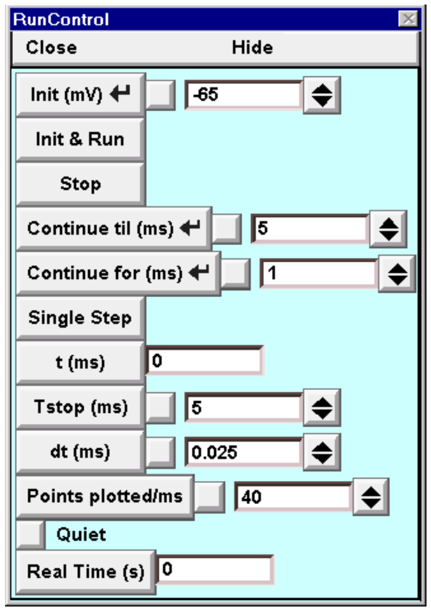


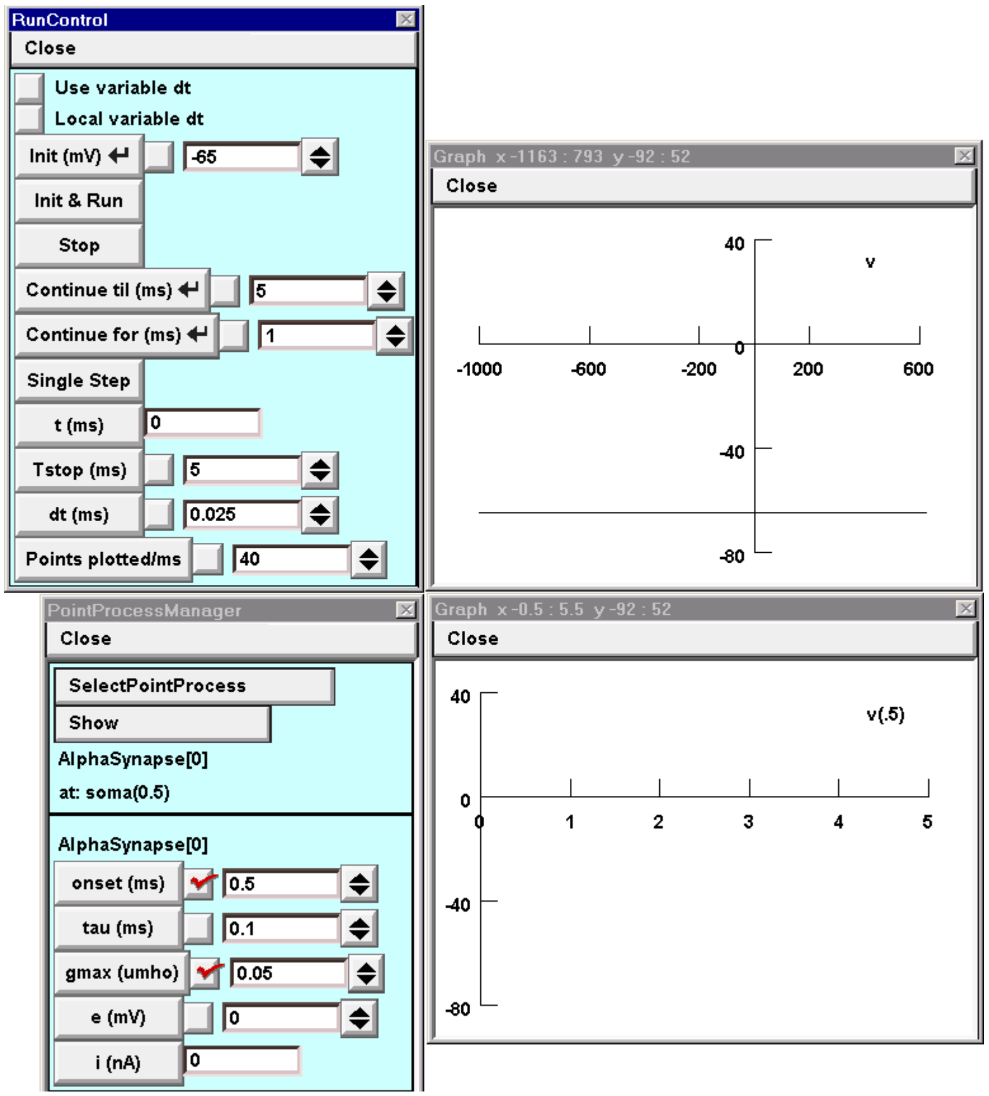
Figure 1.28. Left: To bring up a window with controls for running simulations, click on the RunControl button in the NEURON Main Menu. Right: The RunControl window provides many options for controlling the overall time course of a simulation run. For this example, only three of these controls are relevant.

1. Init (mV) sets time t to 0, assigns the displayed starting value (−65 mV) to $V_m$ throughout the model cell, and sets the ionic conductances to their their steady−state values at this potential.

2. Init & Run performs the same initialization as Init (mV), and then starts a simulation run.

3. Points plotted/ms determines how often the graphical displays are updated during a simulation.

Three other items in this panel are of obvious interest, although we will not do anything with them in this example. The first is dt, which sets the size of the time intervals at which the equations that describe the model are solved. The second is Tstop, which specifies the duration of a simulation run. Finally, the button marked t doesn't actually do anything but is just a label for the adjacent numeric field which displays the elapsed simulation time. Additional features of the RunControl window are discussed in **Chapter W**.



Use variable dt
Local variable dt
Init (mV) -65
Continue til (ms) 5
Continue for (ms) 1
t (ms) 0
Tstop (ms) 5
dt (ms) 0.025
Points plotted/ms 40

AlphaSynapse[0]
at: soma(0.5)

AlphaSynapse[0]
onset (ms) ✓ 0.5
tau (ms) 0.1
gmax (umho) ✓ 0.05
e (mV) 0
i (nA) 0

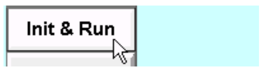
Figure 1.29. The windows we will use to run simulations and observe simulation results. Other windows that are present on the screen but not shown in this figure are the NEURON Main Menu and the CellBuilder.

## 7. Run the simulation experiment

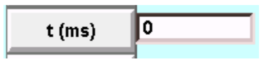
We are now ready to use our "virtual experimental rig" to exercise the model. When we run a simulation with the synapse located at the soma (Fig. 1.30 and 31), a spike is triggered. However, if we move the synapse even a small distance away from the soma along the apical dendrite (Fig. 1.32) and run a new simulation, the epsp is too small to evoke a spike (Fig. 1.33).

The utility of the space plot as a tool for understanding the temporal evolution of $V_m$ throughout the cell can be enhanced by using it like a storage oscilloscope, as shown in Fig. 1.34. This allows us to compare the distribution of $V_m$ at successive intervals during a run. It might be helpful to do something similar with the plot of somatic $V_m$ vs. t if we wanted to compare responses to synaptic inputs with different parameters or locations.

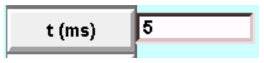
Figure 1.30. Running a simulation.



A. Press Init & Run in the RunControl window to launch a simulation.



B. This makes time t advance from 0 . . .



. . . to 5 ms in 0.025 ms increments. The response of the model is shown in Fig. 1.31.

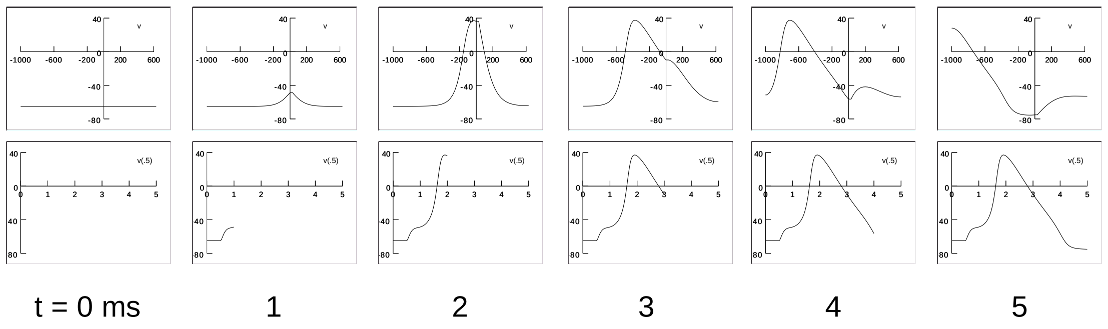
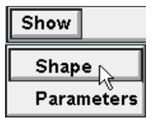
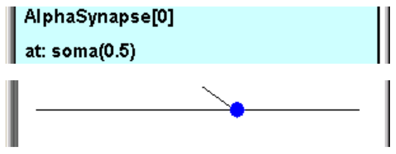


Figure 1.31. Snapshots of the space plot (top) and the graph of $V_m$ vs. t at the soma (bottom) taken at 1 ms intervals. Synaptic input at the soma triggers a spike that propagates actively along the axon and spreads with passive decrement into the apical dendrite.

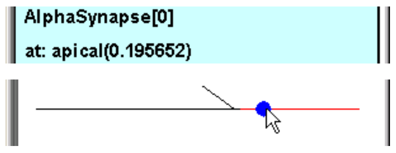
Figure 1.32. Changing synaptic location.



A. In the top panel of the PointProcessManager, click on Show and scroll down to Shape.



B. The top panel remains unchanged, but the bottom panel of the PointProcessManager now displays a shape plot of the cell, with a blue dot that indicates the location of the synapse.



C. Clicking on a different point in the shape plot moves the synapse to a new location. This change is reflected in the top and bottom panels of the PointProcessManager.

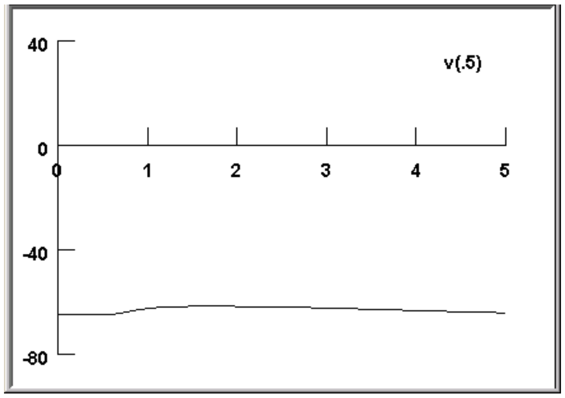


Figure 1.33. Pressing Init & Run starts a new simulation. Even though the synapse is still quite close to the soma, the somatic depolarization is now too small to trigger a spike (space plot not shown).

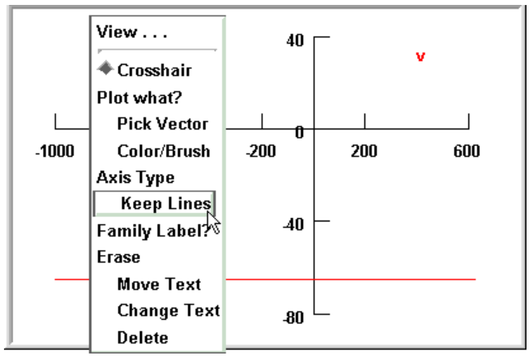
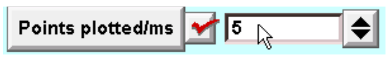
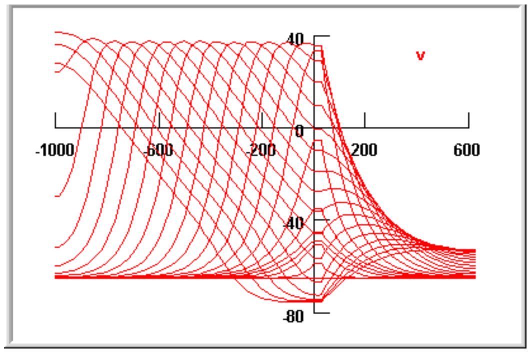


Figure 1.34. A. Activating the keep lines option can help visualize the evolution of $V_m$ more clearly.

Right click in the space plot window to bring up the primary graph menu, then scroll down to Keep Lines and release the mouse button. The next time the primary graph menu is examined, a red check mark will appear next to this item as an indication that keep lines has been toggled *on* (Fig. 1.35 A).



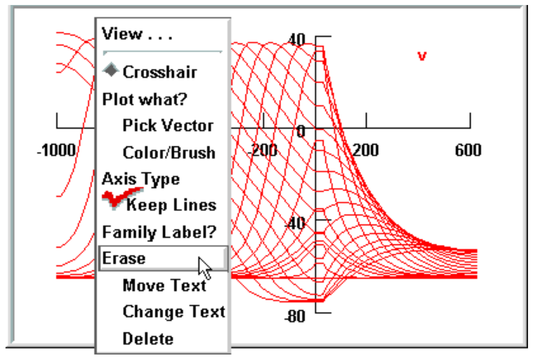
B. To keep the graph from filling up with an opaque tangle of lines, we should make sure the stored traces will be sufficiently different from each other. Plotting only 5 traces per millisecond will do the trick for this example (leave dt = 0.025 ms).



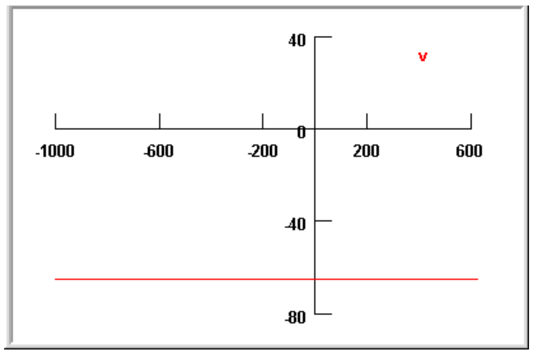
C. Now pressing Init & Run generates a set of traces that facilitate a close examination of the process of excitation and impulse conduction over the model.

For this example the synapse was at the middle of the soma (soma(0.5)). Before running another simulation with a different synaptic location, it would be a good idea to erase these traces (see Fig. 1.35).

Figure 1.35. How to erase traces.



A. Bring up the primary graph menu and scroll down to Erase.



B. The traces will disappear when the mouse button is released. Since keep lines is active, running another simulation will generate a new set of traces.

## 8. Analyze results

In this section we turn from our specific example to a consideration of the analysis of results. Models are generally constructed either for didactic purposes or as a means for testing a hypothesis. The design and analysis of any model are both strongly dependent on this original motivation, which determines what features are included in the model, what variables are regarded as important enough to measure, and how these measurements are to be interpreted.

While computational models are arguably simpler than any (interesting) experimental preparation, analysis of simulation results presents its own special problems. In the first place, attempting to use a digital computer to mimic the behavior of a biological system introduces many potential complexities and artifacts. Some arise from the fact that neurons are continuous in space and time, but a digital computer can only generate approximate solutions for a finite number of discrete locations at particular instants. Even so, under the right conditions the approximation can be very good indeed. Furthermore, a well−designed simulation environment can reduce the difficulty of achieving good results.

Other difficulties can arise if there is a mismatch between the expectations of the user and the level of detail that has been included in a model. For example, the most widely used computational model of a conductance−change synapse is designed to do the same thing each and every time it is "activated," yet most real synapses display many kinds of use−dependent plasticity, and many also have a high degree of stochastic variability. And even the venerable Hodgkin−Huxley model [Hodgkin, 1952 #129], which is probably *the* classical success story of computational neuroscience, does not replicate all features of the action potential in the squid giant axon, because it does not completely capture the dynamics of the currents that generate the spike [Clay, 1982 #370][Fohlmeister, 1980 #369][Moore, 1976 #368]. Such discrepancies are potentially a problem only if a user who is unaware of their existence attempts to apply a model outside of its original context.

The first analysis that is required of all computational modeling is actually the verification that what has been implemented in the computer is a faithful representation of the conceptual model. At the least, this involves checking to be sure that the intended anatomical and biophysical features have been included, that parameters have been assigned the desired values, and that appropriate initialization and integration methods have been chosen. It may also be necessary to test the model’s biophysical mechanisms to ensure that they show the correct dependence on time, membrane potential, ionic concentrations, and modulators. This means understanding the internals of the computational model, which in turn demands a nontrivial grasp of the programming language in which it is expressed. A graphical interface that includes well−designed menus and "variable browsers" can make it easier to answer the frequently occurring question "what are the names of things?" Even so, every simulation environment is predicated on a set of underlying concepts and assumptions, and questions inevitably arise that can only be answered on the basis of knowledge of these core concepts and assumptions.

Verification should also involve the qualitative, if not quantitative, comparison of simulation results with basic predictions obtained from experimental observations on biological preparations or generated with prior models. Discrepancies between prediction and simulation are usually caused by trivial errors in model implementation, but sometimes the fault lies in the prediction. Detecting these more interesting outcomes requires practical facility with the simulation environment, so that the level of effort does not obscure one’s thinking about the problem.

Agreement between prediction and simulation is reassuring and suggests that the model itself may be useful for generating experimentally−testable predictions. Thus the effort shifts from verifying the model to characterizing its behavior in ways that extend beyond the initial test runs. Both verification and characterization of neural models may entail determining not only membrane potential but also rate functions, levels of modulators, and ionic conductances, currents, and concentrations at one or more locations in one or more cells. Thus it is necessary to be able to gather and manage measurements, both within a single simulation run and across a family of runs in which one or more independent variables are assigned different values.

Similar concerns arise in connection with optimization, in which one or more parameters are adjusted until the behavior of the model satisfies certain criteria. Optimization also opens a host of new questions whose answers depend in part on the user’s judgment, and in part on the resources provided by the simulation environment. Which parameters should remain fixed and which should be adjustable? What constitutes a "run" of the model? What are the criteria for goodness of fit? What constraints, if any, should be imposed on adjustable parameters, and what rules should govern how they are adjusted?

In summary, analysis of results can be the most difficult aspect of any experiment, whether it was performed on living neurons or on a computer model, yet it can also be the most rewarding. The issues raised here are critical to the informed use of any simulation environment, and in the following chapters we will reexamine them in the course of learning how to develop and exercise models with NEURON.