SPSIM An Integrated Visual Simulation Environment

for Analyzing Spinal Reflex Circuits

by

James Harrison

12/19/97

Supervised by

K. R. Subramanian Department of Computer Science

D. P. Bashor Department of Biology

Variables:

Methods:

Framing Class

Variables:

Methods:

Histogram Class

Variables:

Methods:

MDArray Class

Variables:

Methods:

Parser Class

Methods:

PihData Class

Variables:

PlotType Class

Variables:

Methods:

Population Class

Variables:

Methods:

Random Class

Variables:

Methods:

Simulation Class

Variables:

Methods:

SynapticType Class

Variables:

Methods:

Friend Methods:

TwoDArray Class

Variables:

Methods:

Appendix C

Software Description

Chapter 1 Introduction

1.1 Introduction

The segmental circuitry of the spinal cord has been the subject of much study. Most of that study has focused on a single cell. The focus of our research has been to simulate and visualize the behavior and interactions of spinal neuronal populations during production of reflexes and simple behaviors, such as walking and standing.

SPSIM is a visualization tool that shows the circuit activity of neuronal populations, built up from large numbers of realistic individual elements, with parameters established by animal experimentation. It uses a mathematical model, based on the algorithms of MacGregor (1987), to generate a large scale simulation of cell population interactions. Once run, this large amount of data is visualized in a 3D environment. The user then can view the populations at each time step and how they relate to each other. SPSIM greatly increased the rate at which data could be analyzed, and provides a graphical user interface for carrying out statistical analysis.

Chapter 2 Background

2.1 Background

The programs on which SPSIM is based were originally developed by Dr. K.R. Subramanian (Computer Science), in collaboration with Dr. David P. Bashor (Biology), for use in visualizing simulations of spinal cord circuitry controlling a pair of muscles. Following pilot implementations by Dr. Subramanian, Vann Hasty implemented a VCR-type interface for the simulator, and made a number of improvements to speed simulation (senior thesis for a BS at the UNC Charlotte, 1996).

The numerical part of the simulation was originally written in FORTRAN and created by D. P. Bashor from the SYSTEM series algorithms of MacGregor. Dr. Subramanian converted Dr. Bashor's simulation environment to C and built a command-line interface using TCL as the first step in "modernizing" the system. Later he added some data storage so the entire simulation could be executed once and recorded for future playback.

The 3D visualization, created by Vann Hasty, was written in Open Inventor which is an object oriented graphics tool kit written in C++ and based on the Open GL standard. Open Inventor supports objects such as three dimensional text, polyhedral, triangle strips, quad meshes, Bezier patches, and Non-Uniform Rational B-Splines (NURBS). Inventor also supports control objects such as engines and sensors that control and detect changes in the geometry of the simulation. Open Inventor was chosen because:

- It shields the programmer from the low level of Open GL, letting him/her focus time elsewhere.
- It provided an easy integration with Motif
- It is the standard 3D toolkit on SGI workstations

Since this tool was to be used by non computer scientists, a friendly GUI interface was implemented by Vann Hasty. All of the user interaction with the program is done through the implementation of an OSF Motif style application interface. Motif is an abstraction of the X intrinsics toolkit, which is an abstraction of the Xlib standard developed at the Massachusetts Institute of Technology. Motif was chosen because

- Like Inventor, it is easy to implement in a X windows environment, shielding the programmer from the lowest level system calls. One line of Motif code can implement many lines of X code.
- Inventor has built in functions to handle Motif for easy integration.

The statistical functions were written by Dr. K.R. Subramanian in C. These include:

- **Plot Interval Histogram** This function enables the interval between successive cell firings to be recorded and displayed in the form of a histogram.
- **Population Cell Firings** reports the number of cells that were active within a population at each displays time step.
- **Rate Meter Plot** the firing frequency of a cell in a particular population as a function of the simulation time.
- **Cell Potential Plot** This function describes the potential of a cell in a particular population as a function of the simulation time.

2.2 Problems with Previous System

The SPSIM's main problem at this time was it consisted of two separate programs (see figure 1):

- A TCL command line program where statistical functions could be run.
- A 3D visualization that only showed the populations state at each time step, but there was no analysis tools.

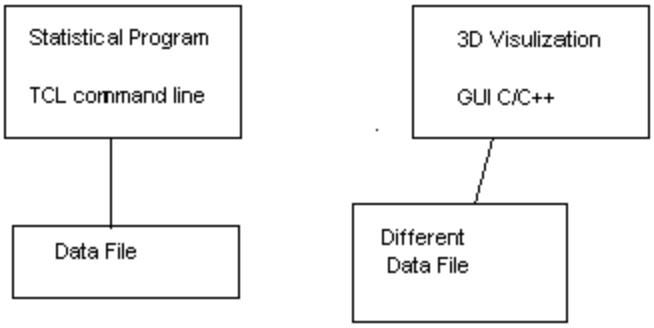


Figure 1 Old SPSIM

This meant that the user would run the 3D visualization to look at the population level, then start the TCL program to analyze at the cell level. This situation was obviously cumbersome. In addition, there were several files used, each with a different format. This lead to more confusion. What was needed was a single environment integrating the statistics and the 3D visualization, giving the user the ability analyze at the population level and the cell level.

Chapter 3 Software Design

3.1 Software Design of SPSIM

The design of SPSIM was driven by three main goals:

- Choose a programming language that would make it easy to maintained and expanded SPSIM.
- Provide a integrated environment for investigating the behavior of cell populations, that includes powerful analytical tools is flexible, and easy to use.
- Increase the power of SPSIM by adding new features based on visualization technology.
- Add more flexibility to SPSIM with functions like picking and printing.

3.1.1 Programming Language

The first step was to decide which programming language should be used. Since the project will be a continuing one and many different people will be working on SPSIM, I chose C++. C++ makes maintenance and extension of code simpler. The encapsulation or data hiding provides protections against unwanted side effects and relieves future programmers from having to have full knowledge of the previous implementation. In addition, Open Inventor is already written in C++.

Since Open Inventor is already written in C++, I had to convert the numerical simulation and the statistical functions. I analyzed the simulation and created the following classes with methods to access and manipulate their data(see figure 2):

- **Population class** This is a base class for Cell and Fiber classes.
- **Cell class** Holds the cell population properties.
- **Fiber class** Holds the fiber population properties.
- **Synaptic class** Holds the synaptic properties for each type.
- **ActiveNeuron class** This class holds which cells have fired and their

  threshold and potential values.

- **Simulation class** This class contains all the above classes, therefore the main program can create a Simulation object and call it's methods to run the simulation without directly accessing the inner classes.

The statistical classes were easier to create. They are simple the original C functions wrapped in the Stat class. Each function has a method associated with it. There are other utility classes such as, Random class and TwoDArray class that are used inside of the other classes as needed. A more detailed explanation of the classes is found in Class Description.

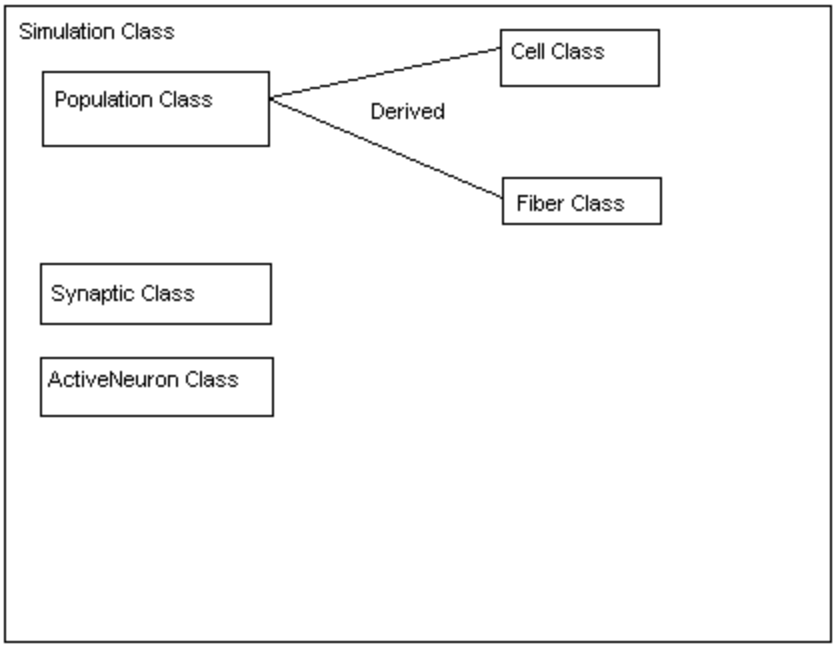


Figure 2 Simulation Class Overview

3.1.2 Integrating SPSIM

In order to integrate SPSIM, I divided it into four major components:

- the numerical simulation
- the 3D visualization
- the interface
- the statistical functions

The final step was to integrate all four components into one application (see figure 3), this involved:

- using Motif's callback mechanism to call the appropriate functions when the user interacts with the interface. It is important to note that the callback functions were written in C. this is because Motif does not include handlers for C++. These C functions then called the appropriate classes method.

- Integrating Inventor SceneGraph viewer into the main Motif frame window.
- . Adding an initialization file to provide the most flexibility possible. This ASCII file contains information that allows SPSIM to change certain parameters, like population labels and population size, easily by non computer scientists. In the future, this should be also accessible from the SPSIM interface

Figure 3 Application Overview

3.1.3 New Visualizations

Interactive Connection Viewer

The old SPSIM showed how individual populations behaved, but it did not directly show how one population effected another population. The Interactive Connection Viewer allows the user to see the relationship between two populations.

Once a simulation has been run, the user can choose two populations he wishes to investigate. A new viewer is generated with a 3D representation of each population and it's output connections to other populations. This new view is directly tied into the simulation, so as the simulation progresses, the strengths of each output connection is updated and color coded. The connections change their color based on number of sending cells which are active. This shows the connections between populations and how they transmit their information.

The user can also specify how often the view should update. Adding the flexibility of comparing values at any interval.

3D Rate Meter Plot

The old SPSIM allowed the user only to compute a rate meter plot for a single cell in a population. The new function produces a 3D rate meter plot of all the cells in a population. This type of analysis will enable one to see the overall view on a population level, spot trends, and narrow the scope. Then other statistical functions can be used to analyze at the cell level.

3.1.4 New Functions

Picking

The user can pick on any population. Once it is picked it can be scaled, rotated and moved around in the environment by clicking and dragging. This gives the user the freedom to arrange populations in any order that is appropriate to their needs.

Printing

Any view can be printed to a file or printer. This allows the user to have a hard copy of a population at a particular time step. This can be used for further analysis at a later date.

Compute Statistics in Batch Mode

A simple text file can contain commands to run any of the statistical menu items. Each type of statistic dialog box has the ability to be provided a batch file. This is useful to perform multiple statistics from a single simulation.

Easy Control of Simulation Environment Parameters

A initialization file, called sim.ini, is provided to let the user determine certain environmental parameters. These include the population labels. This file is in ASCII, therefore is easy to edit with a simple text editor.

Time Step Slider Bar

This allows the user to jump to any point in the simulation easily by clicking and dragging a slider bar. Being able to jump from time step 50 to time step 1200 is a great time saver. In the old SPSIM, the user would have to run the simulation from 50 to 1200.

Chapter 4 System Description

4.1 SPSIM features

Increased Speed of Analysis of Data

The rate at which data can be analyzed is 100 fold increase compared to GeomView. SPSIM also allows the user to look at a broad view with many parameters and then focus in on a small subset of the data quickly. Therefore, it is easy to spot trends that need to be investigated.

Interactive Connection Viewer

This allows the user to specify two populations. A 3D representation of each population and it's output connections to other populations is generated. In addition, as the simulation progresses, the strengths of each output connection is updated and color coded. This allows the user to quickly evaluate how these two populations affect other populations.

Built in Statistical functions

The data can be easily analyzed by using the built in statistical functions. After the simulation is run, the user can click on the statistics menu item to choose between two kinds of rate meter plots, a histogram of population firing intervals, a plot of cell potentials as a function

of time, and a plot of the firings of cells in a population. Immediately, a 2D or 3D graph is generated to display the results. The data is saved to a file that can be viewed later.

Compute Statistics in Batch Mode

A simple text file can contain commands to run any of the statistical menu items. Each type of statistic dialog box has the ability to be provided a batch file. This is useful to perform multiple statistics on a single simulation.

Ability to Easily Alter Simulation and Environment Parameters

SPSIM reads the simulation parameters from an easily editable text file. This allows the user to change one value, run the simulation, and see the results immediately. Certain environment variable, like population labels, are also in the text file. *Future work will allow the simulation to be altered through a GUI interface and saved to a file*.

Interactive 3D Environment

3D environment allows the user to manipulate the scene 360 degrees, as well as zooming in or out. Populations can be easily hidden by clicking on the label. In addition, populations can be scaled, rotated and moved around in the environment by clicking and dragging.

Printing

Any view can be printed to a file or printer.

VCR interface

The VCR like interface allows the complete control over the visualization. The user can increment step by step to view each time step, or play continuously forward, backwards, fast forward, fast rewind. The speed of playback is also adjustable.

Time Step Slider Bar

Allows the user to jump to any point in the simulation easily by clicking and dragging.

GUI interface

Easy to use intuitive GUI interface built in Motif. Insures that users will be able to use SPSIM almost immediately.

4.2 Quick Start to Using SPSIM

4.2.1 Initializing a Simulation

The following steps must be done to initialize a simulation:

1. *Make sure that the sim.ini file exists and is consistent with the simulation data file that you plan to use. See File Formats for an explanation of the sim.ini file.*

2. Click on File, choose Open, and type in or click the name of the simulation data file you wish to run.

3. Click OK to start the simulation. A message box will display how time steps have been completed.

4. A message box will display that the simulation has completed. You are now ready to view the simulation results.

4.2.2 Viewing the Simulation

1. Using the control bar, you can Play, FF, Rev, Step, Pause, or Restart the simulation visualization.

2. The Time Step Slider Bar allows you to jump from time step to time step within the simulation.

A more complete description of the Control Bar can be found under Application Window description.

4.2.3 Statistics on the Simulation

SPSIM computes Interval Histograms, Rate Meter Plots, Population Cell Firings and Cell Potentials, These are graphed by Xgraph or SPSIM. There is also an option to run batch files to do many versions of the above statistics. See File Formats for an explanation of the batch file.

A more complete description of the Statistic Menu can be found under Application Window description.

4.3 Application Window Description

4.3.1 Menu Bar

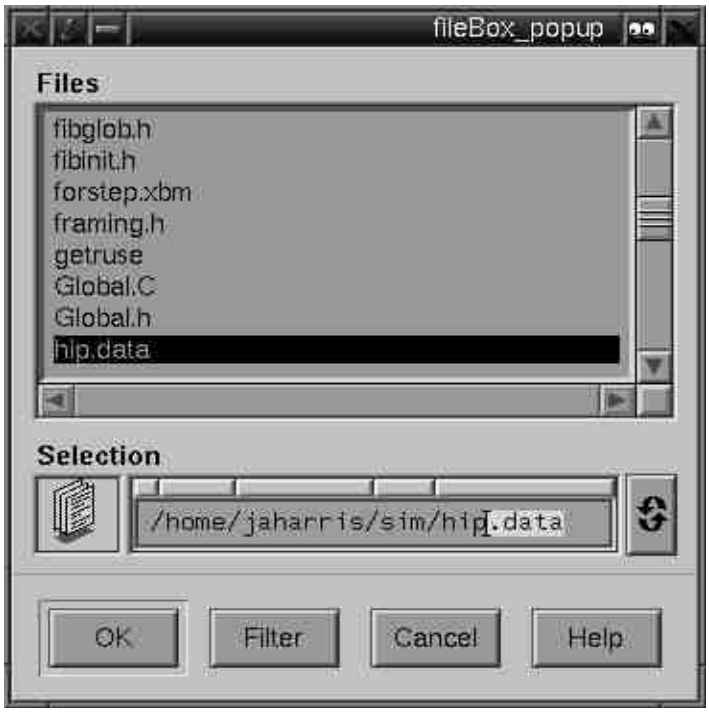
File

Open

-

Prompts the user to choose which simulation data file to open. Note: only one simulation can be open at a time. Opening one will delete any existing simulation.

-



-

-

-

-

-

Once a file is chosen another dialog box appears, the user is prompted to enter in the starting and stopping time steps for the simulation.

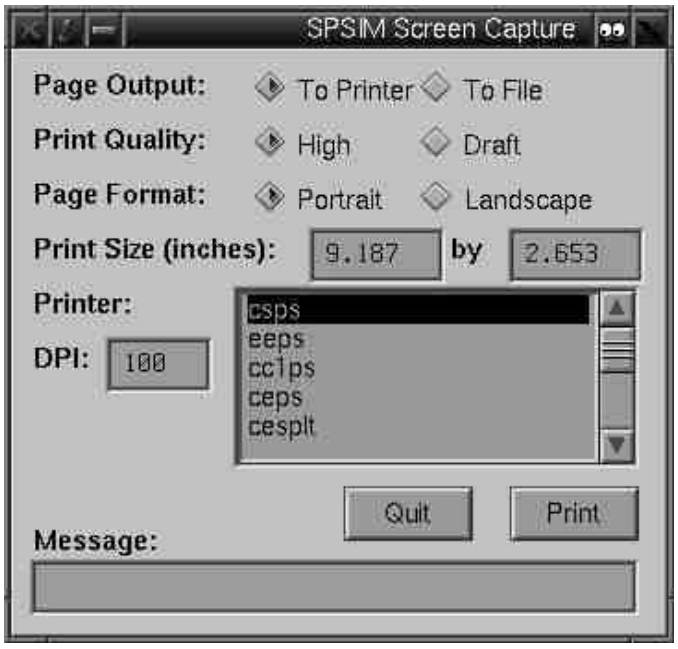
After clicking OK, SPSIM will check the sim.ini file to verify that the starting and stopping times are valid and if so run the simulation. A message box will appear showing how many time steps have been completed.

Save

For future use: Save a simulation to file, once it has been run.

Print

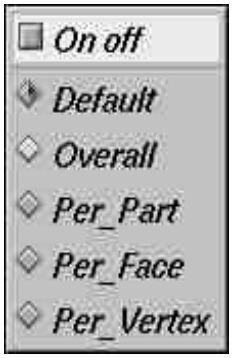
A standard print dialog box allows the user to print out the current view to a file or a printer. The file formats allowed are PostScript and RGB. The size and resolution can also be set by the user.



Quit

Exit SPSIM.

Shading



For future use: Change the type of shading used in rendering.

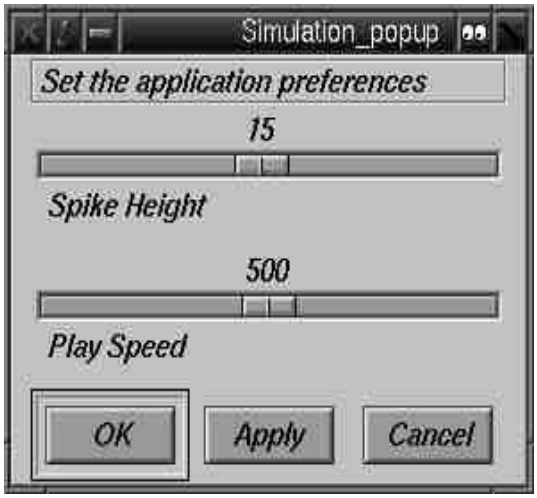
Options

Preferences

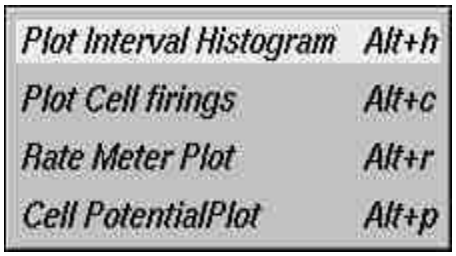
-

-

Allows the user to set the spike height during visualization and at what speed the simulation renders.



Statistics



Plot Interval Histogram

-

This function enables the interval between successive cell firings to be recorded and displayed in the form of a histogram.

The dialog box to allow the user to specify:

Population ID

Stepsize

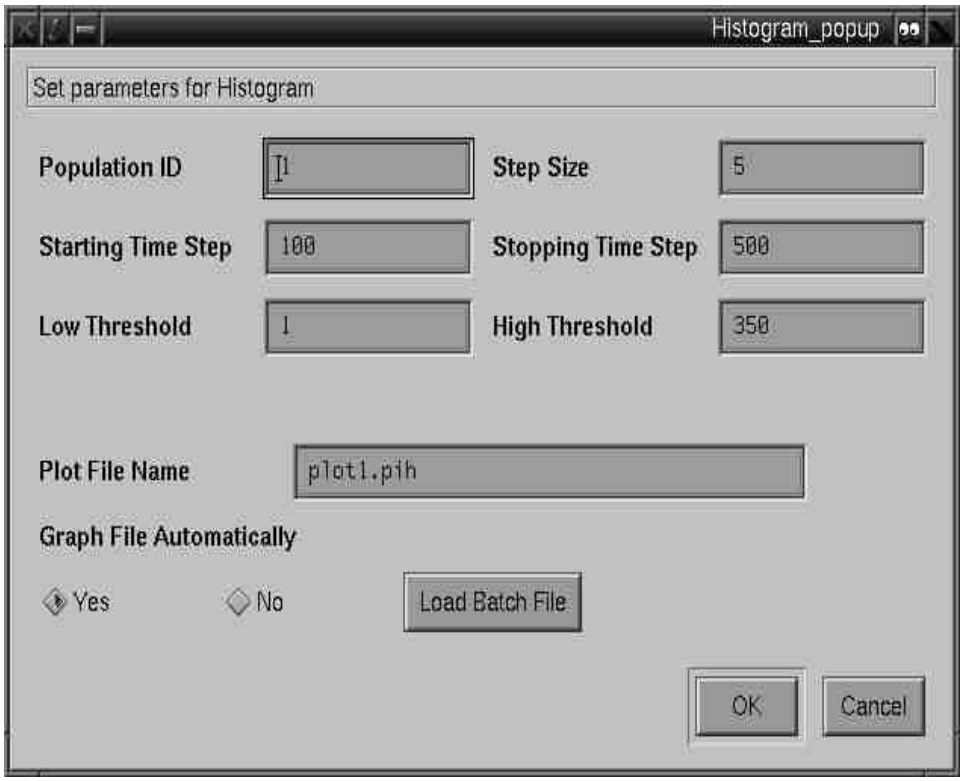
Starting time step

Stopping time step

Low Threshold

High Threshold

Filename to store the graphing data produced



Population Cell Firings

-

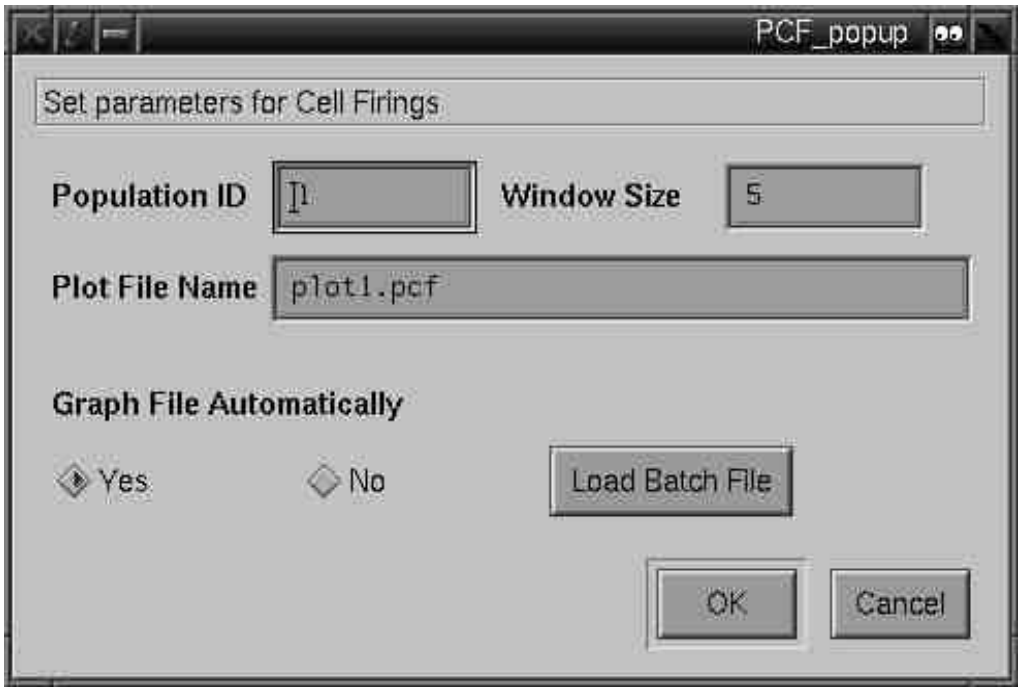
This function reports the number of cells that were active within a population as a function of the time step.

The dialog box to allow the user to specify:

Population ID

WindowSize

Filename to store the graphing data produced



Rate Meter Plot

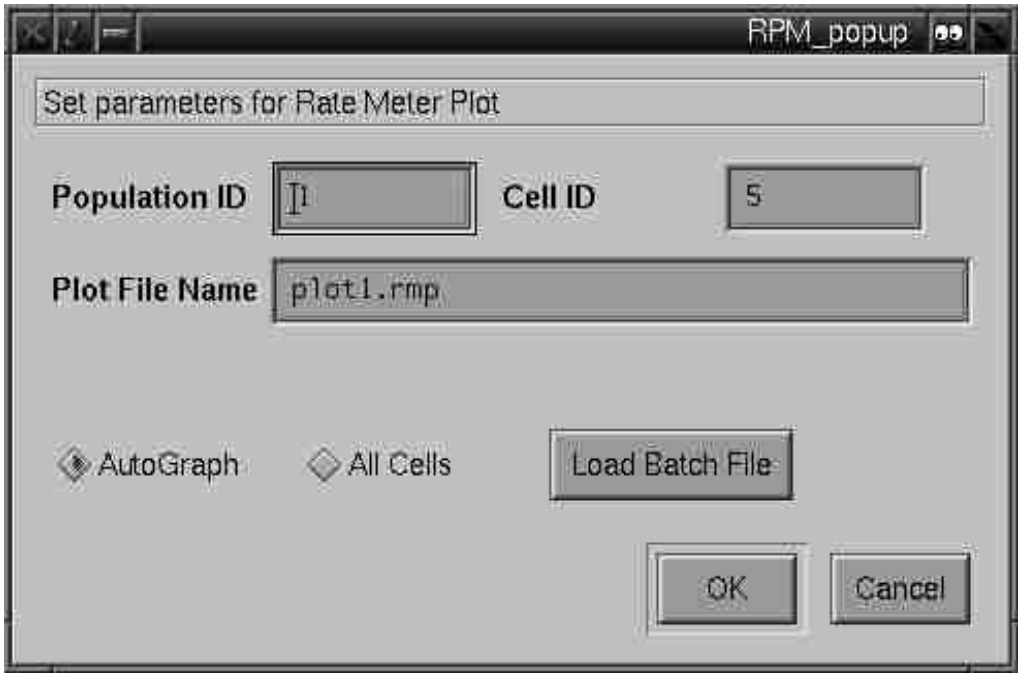
The rate meter plot describes the firing frequency of a cell in a particular population as a function of the simulation time.

The dialog box to allow the user to specify:

Population ID

Cell ID

Filename to store the graphing data produced



- *The user can also choose to have SPSIM calculate all the cells for a population by clicking on the AllCells radio button. This will disable AutoGraph and ignore the value in the Cell Number field. Once OK is clicked, another viewer will appear showing the Rate Meter Plot for all the cells in the population over the simulation.*

-

-

-

-

-

Cell Potential Plot

-

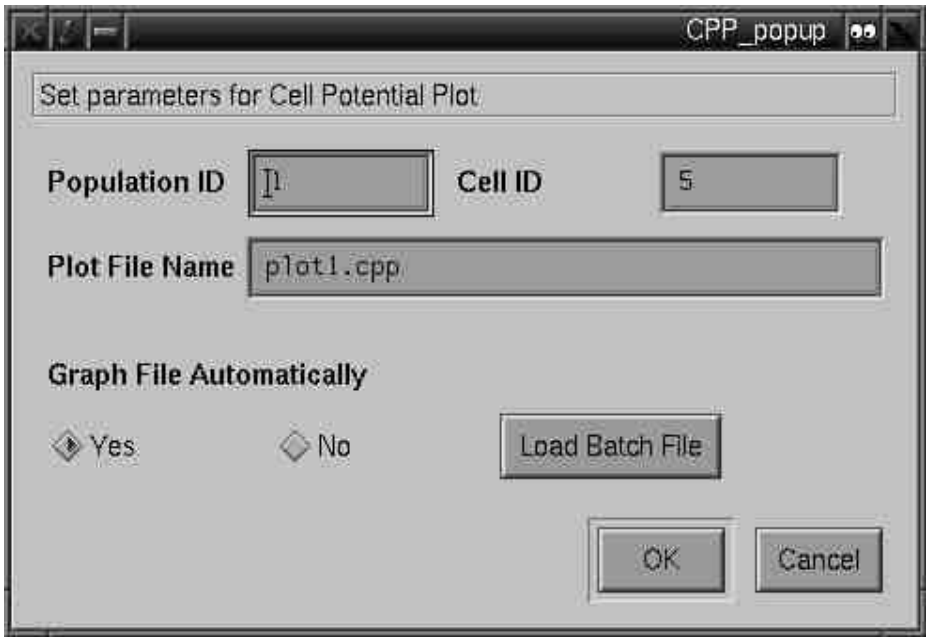
The cell potential plot describes the potential of a cell in a particular population as a function of the simulation time.

The dialog box to allow the user to specify:

Population ID

Cell ID

Filename to store the graphing data produced



-

-

-

-

-

-

All the statistical dialog boxes contain:

A button for auto graphing. If left checked it will automatically call Xgraph with the filename that the user supplied. Uncheck the autograph button to disable this feature.

A batch file button. If checked it will prompt the user for a batch file of

commands to run. Autographing is turned off automatically. The default for this button

is unchecked. See File Formats for a description of the batch file format.

Views

Interactive Connections

Creates a new view that shows the output connections of two populations that is updated in parallel with the simulation (see figure 1). They are assigned a strength value based upon:

- *the strength of the output synapse of the population*
- *the number of terminals the population projects*
- *the total number of cells in the population that have fired during a given interval*

. A dialog box to allow the user to specify:

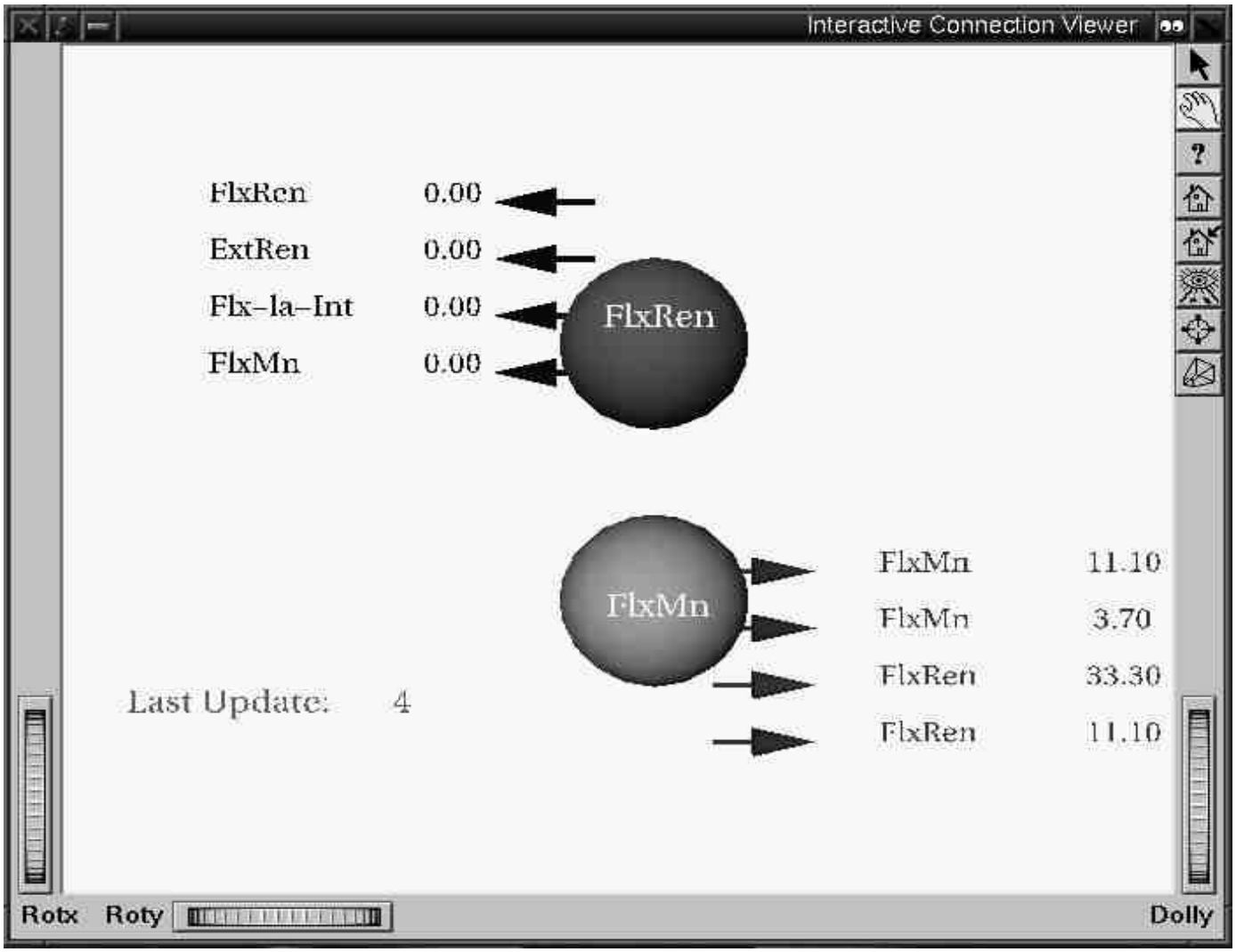
The ID of the first Population

The ID of the second Population

The time step duration

The user has the option of choosing a time duration, causing the strengths to be added. For example, if a duration of 5 ms were chosen, then the strength would be calculated for each time step and added together to get the total strength for the 5 ms. This value then would be displayed next to the connection

strength value is displayed by it's connection is also assigned to a color. This is to help visually show the changes. The color table ranges from blue to red, with black = 0.



Connection Image

Allows the user to load an image of the population connections. This image must be in a format supported by XV, like GIF, JPG, or RGB. A system call is made to XV, which then will display the image chosen.

Help

Help

-

For future use

About

Displays the version number and date of SPSIM

-

4.3.2 Simulation Control Bar

The buttons only function after a valid simulation has been completed.

Play Button

Starts the simulation visualization.

Stop Button

Stops the simulation visualization.

FF Button

Fast Forward scan of the simulation visualization.

Rev Button

Reverse scan of the simulation visualization at the same speed as play.

Forward Step Button

Increases the time step of the simulation by 1 step.

Reverse Step Button

Decreases the time step of the simulation visualization by 1 step.

Pause Button

Pauses the simulation visualization.

Restart Button

Resets the time step to the first time step in the simulation.

Time step Slider Bar

Allows the user to jump to any time step in the simulation. Either by clicking and dragging

the slider or using the right and left arrow keys to move to the appropriate time step.

4.3.3 Population List Box

This list box contains the names of the cell and fiber populations. Clicking on any of the names will hide that population from the render area. Conversely, if the population is already hidden, it will unhide that population.

4.3.4 Population Picking

The user may move populations anywhere on the screen by following these steps:

- *Click on the arrow icon on the upper right hand of the tool bar*
- *Click on any population*

A bounding box will enclose the population. The user can scale, rotate, or move the population. Clicking on another population or anywhere in the environment will remove the bounding box.

Chapter 5

5.1 Future Work

Below are some of the improvements and enhancements that will be added to SPSIM:

Interactive Circuit Diagram Creation

Let the user create a Circuit Diagram by clicking and dragging objects onto a drawing canvas. This would be similar to drawing program like PaintBrush. You would have certain atomic objects, like population objects and connections objects that one could put together to build a Circuit Diagram. Like the Interactive Connection diagram, this would also show the connection between populations and how they

transmit their information. The connections change their "quality" based on number of sending cells which are active, this could be represented by changing the color or width of the connection. This would be update every 25 or 50 ms. This diagram then would build a data structure that would be used in a simulation.

Voltage-Time plot

Allow the user to click on a population and then show how the voltage is changing with

each time step for a chosen cell in a population.

Change the Visibility of the Threshold and Potential Height Fields.

Allow the user to click on a populations Threshold or Potential quad mesh in the 3D visualization and hide it.

Save the simulation

Allow the user to save the numerical simulation to a file. This could later be reread, this would save the time it would take to rerun the numerical simulation.

-

Appendix A

File Formats

Sim.ini

A text file called sim.ini needs to be in the same directory as the SPSIM executable. It

contains values that do not change often, but may change in the future. These add

greater flexibility to SPSIM. SPSIM uses these values to initialize the classes. **This file must be correct or SPSIM will run incorrectly**. Below is an explanation of the sim.ini

file format.

The format is VARIABLE NAME VARIABLE VALUE

The following variables must be defined:

NUM_CELL_POP Number of cell populations

NUM_FIBER_POP Number of fiber populations

MAXIMUM_CONDUCTANCE_PLUS1

NUM_SYNAPTIC_TYPES Number of synaptic types

CELL_ARRAY_WIDTH

CELL_ARRAY_HEIGHT

TIME_STEP_SIZE

MAX_NUM_CELLS Maximum number of cells in a population

NUM_LABELS Number of population labels to read in

This is where the population label

text goes. The number of line of

text must match the NUM_LABELS

value

All other text will be ignored.

Batch Files

SSPIM has the ability to read in commands, used with the statistics, stored in a batch

file. All the statistic dialog boxes have a batch file button. If checked it will prompt the

user for a batch file of commands to run. Autographing is turned off automatically. The default for this button is unchecked. Below is a description of the batch file formats.

The basic pattern for all the formats is *-flag value*.

Interval Histogram Flags:

-p population number

-hist low threshold high threshold stepsize

-int starting time step stopping time step

-o output file name

i.e.

-p 12 -hist 5 35 5 -int 100 1500 -o pih.plot1

Rate Meter Flags:

-p population number

-c cell number

-o output file name

i.e.

-p 12 -c 5 -o rmp.plot1

Population Cell Firings:

-p population number

-m window size

-o output file name

i.e.

-p 12 -m 5 -o pcf.plot1

Appendix B

Class Description

ActiveNeuron Class

Contains the properties for a all the cells that have fired for each time step. Contains a linked list of FiringCells, which holds information on which cells have fired. The data in this class is used as data points for the quad meshes in the Inventor code.

Variables:

TwoDArray threshold *Holds the threshold value of the firing cell*.

TwoDArray potential *Holds the potential value of the firing cell*.

Public:

FiringCell *firingcell *Linked list of the cells that fired for the entire simulation*..

Methods:

void setPotential(int popid, int cellid, double value)

Sets the potential for a cell in a population to value.

double getPotential(int popid, int cellid)

Returns the potential for a cell in a population.

void setThreshold(int popid, int cellid, double value)

Sets the threshold for a cell in a population to value.

double getPotential(int popid, int cellid)

Returns the potential for a cell in a population.

Void updateThreshPot(CellPopulation* cellpop, int NUM_CELL_POP, int popid)

For each timestep update the threshold and potential for each cell in each CellPopulation.

void ini(int NUM_CELL_POP, MAX_NUM_CELLS)

Allocated memory for arrays and sets all values to 0. Allocates memory for a FiringCell.

void updateCellFirings(FiringPop* firingpop)

Transfers the data from firingpop to a FiringCell, then puts onto the end of a linked list.

CellPopulation Class

Population->CellPopulation. Contains an array of cells and the properties of the population.

Variables:

Static int numCellPops *Total number of CellPopulations*

double snsAccomadationofCell *sensitivity to accommodation of cells in a population*.

Range of 0-1.

double timeConstAccomodationofCell *Time constant of accommodation of the cells in the*

population.

double snsofPotassiumConductance *Sensitivity of the potassium to spike*.

double timeConstGKRecovery *Time constant of the GK recovery following a spike*.

double timeConstMembrane *Time constant of the membrane*.

double threshold *Baseline threshold potential of cells in a population*.

double dcth

double dcg

double *potential *Array, size of numCellInPop, that holds the membrane*

potential for each cell in the population.

double *firingThreshold *Array, size of numCellInPop, that holds the firing*

threshold for each cell in the population.

double *potassiumConductance *Array, size of numCellInPop, that holds the potassium*

conductance (GK) for each cell in the population.

MdArray synapticConductance MdArray type, size of

*NUM_SYNAPTIC_TYPES*numCellInPop**

MAX_CONDUCTANACE_TIME_PLUS1, that contains the conductance of a particular synaptic type on a particular cell in the population.

Methods:

int getNumCellPops()

Returns the total number of cell population classes.

void setNumCellPops(int i)

Sets numCellPops to the value i.

void iniCells(int TIME_STEP_SIZE, int NUM_SYNAPTIC_TYPES, int MAX_CONDUCTANCE_TIME_PLUS1)

Allocates memory for the arrays, initializes the dcth, dcg values, the threshold and firing threshold for each cell (these values were read in from the data file), and sets all other variables in the class to 0.

void svUpdate(int NUM_SYNAPTIC_TYPES, int TIME_STEP_SIZE, int POTASSIUM_POTENTIAL, int popid, SynapticType synaptictype, int cellid)*

Updates the state of each cell in the population. If the potential is greater than the threshold, then values are transmitted to the receiving population(s) with transmit.

Void updateSynCond(int NUM_SYNAPTIC_TYPES, int MAX_CONDUCTANCE_TIME_PLUS1,

SynapticType synaptictype)*

Updates the synaptic conductance value for each cell in the population. ????

Friend Methods:

inline void transmit(int popid, int cellid, int CELL_ARRAY_WIDTH, int CELL_ARRAY_HEIGHT,

FiberPopulation fiberpop, CellPopulation* cellpop)*

Projects synaptic activation from a sending Fiber population to it's targeted receiving cell if the cell's spike[i] = 1.

Inline void transmit(int popid, int cellid, int CELL_ARRAY_WIDTH, int CELL_ARRAY_HEIGHT,

CellPopulation cellpop)*

Same as above except, project synaptic activation from a sending Cell population to it's targeted receiving cell if the cell's spike[i] = 1.

Ifstream& operator>>(ifstream&, CellPopulation&)

Overloaded >> operator used when reading the data file. Method reads in the data

formatted for the CellPopulation.

FiberPopulation Class

Population->FiberPopulation. Contains an array of cells and the properties of the population.

Variables:

static int numFiberPops Total number of Fiber Populations.

double fiberFireProbability The probability that a particular fiber will fire at a particular time.

int fiberStartTime starting time of activity in a fiber system.

int fiberStopTime stopping time of activity in a fiber system.

int randomFiberSeed Random number seed governing action potential timing in a fiber population.

Methods:

int getFibstart()
Returns fiberStartTime.

int getFibStop()

Return fiberStopTime.

int getNumFiberPops()
Returns the total number of fiber Populations.

void setNumfiberPops(i)

Sets numfiberPops equal to i.

int chkFiring(int fcellid)

Checks to see if the probability of the fiber to fire exists. If it does then the spike array is marked with a value of 1.

void iniCells()

Allocates memory for spike array and sets each cell = 0.

Friend Methods:

inline void transmit(int popid, int cellid, int CELL_ARRAY_WIDTH, int CELL_ARRAY_HEIGHT,

FiberPopulation fiberpop, CellPopulation* cellpop)*

Projects synaptic activation from a sending Fiber population to it's targeted receiving cell if the cell's spike[i] = 1.

Ifstream& operator>>(ifstream&, FiberPopulation&)

Overloaded >> operator used when reading the data file. Method reads in the data

formatted for the FiberPopulation.

FiringCell Class

Receives it data from the FiringPop class each timestep. Contains the number of firings per population and which cells fired. Used only with the ActiveNeuron Class.

Variables:

public:

int numcells total number of cells that have fired in a population.

int popid Population id number.

*int *cellids Array, size of MAX_NUM_CELLS_POP, holds which cells have*

fired.

*FiringCell *next Pointer to the next FiringCell in the linked list.*

Methods:

FiringCell()

Constructor, sets all variable to 0 or NULL.

void ini(int MAX_NUM_CELLS_POP)

Allocates memory for the cellids array.

FiringPop Class

Used to store the number of firings for population and which cells fired per time step. The data then is transferred to the FiringCell class after each time step and reset to NULL. This is used with the ActiveNeuron class.

Variables:

public:

int numFirings total number of cells that have fired in a population.

int popid Population id number.

int timestep Time step in the simulation.

*int *cellids Array, size of MAX_NUM_CELLS_POP, holds which cells have*

fired.

Methods:

FiringPop()

Constructor, sets all variable to 0 or NULL.

void ini(int MAX_NUM_CELLS_POP)

Allocates memory for the cellids array.

Framing Class

Stores the starting and stopping points of the simulation and which timestep is currently being used.

Variables:

int timeStep Current timestep in the simulation.

int startFrame 1st timestep in the simulation.

int stopFrame The last timestep in the simulation.

Methods:

Framing()

Initializes all variables to 0.

int getTimeStep()

Returns the current timestep.

void setTimeStep(int i)

Sets the current timestep to i.

int getStartFrame()

Returns the starting timestep of the simulation.

void setStartFrame(int i)

Sets the starting timestep of the simulation to i.

int getStopFrame()

Returns the last timestep of the simulation.

void setStopFrame(int i)

Sets the last timestep of the simulation to i.

Histogram Class

Holds data to be displayed in a histogram. Used with the createPIH function.

Variables:

#define MAX_BINS 200 Maximum number of bins in histogram.

int LowThresh Low Threshold.

int HighThresh High Threshold.

int start_time Starting time for histogram loop.

int stop_time Stopping time for histogram loop.

int step_size Size of the histograms bins.

int LowThreshFreq Low Threshold Frequency.

int HighThreshFreq High Threshold Frequency.

int num_bins Number of the histograms bins.

int freq[MAX_BINS] Array, size of #define MAX_BINS, holds the frequency values for each bin.

Methods:

Histogram()

Initialize num_bins to 1 and all other variable to 0.

*void setValues(PihData *data)*

Sets values for LowThresh, HighThresh, start_time, stop_time, and step_size.

void setNumBins()

Sets the value of num_bins = HighThresh/step_size.

int getNumBins()

Returns the value of num_bins.

void setFrequency(int i , int value)

Sets the freq array[I] = value.

int getStartTime()

Returns the value of start_time.

int getFinishTime()

Returns the value of finish_time.

void setLowThreshFreq(int i)

Sets the value of LowThreshFreq = i.

void setHighThreshFreq(int i)

Sets the value of HighThreshFreq = i.

int findHistogramBin(int cell_interval)

Returns the histogram bin the cell_interval belongs to.

Void plotHistogram(ofstream &fout)

Writes out freq[] values to file pointed to by fout.

MDArray Class

Three dimensional array class that can be allocated dynamically.

Variables:

*double *array Pointer to array.*

int tmaxx Maximum value of x component of array.

int tmaxy Maximum value of y component of array.

int tmaxz Maximum value of z component of array.

Methods:

MDArray()

Constructor, initializes all values to 0 or NULL.

void iniMDArray(int x, int y, int z)

*allocates memory for the array, size of x*y.*

double getValue(int x, int y, int z)

returns the value at array index (x,y, z).

void setValue(int x, int y, int z, double value)

Sets the value of (x,y,z) to value.

Parser Class

Used to parse out the command line arguments for pih, rmp, pcf statistical function batch files.

Methods:

*PihData *pihParse(char* buff, PihData* PihData)*

Takes a null terminated string and parses out the command for a pih plot as follows:

-p, -hist, -int, -o and stores the values into the proper place in the PihData class.

*PihData *pcfParse(char* buff, PihData* PihData)*

Takes a null terminated string and parses out the command for a pcf plot as follows:

-p, -m, -o and stores the values into the proper place in the PihData class.

*PihData *rmpParse(char* buff, PihData* PihData)*

Takes a null terminated string and parses out the command for a rmp plot as follows:

-p, -c, -o and stores the values into the proper place in the PihData class.

PihData Class

Contains data needed for the pih, rmp, and pcf statistical functions.

Variables:

public:

int lowthresh Low Threshold.

int highthresh High Threshold.

int start Starting timestep.

int stop Stopping timestep.

int stepsize Size of the step for the histogram use in createPih.

int popid Population id.

int cellid Cell id.

int widowsize Size of the window used in createPcf.

int flag flag variable.

PlotType Class

Contains the values used in XGraph to graph the statistical functions createPih, createRmp, and createPcf.

Variables:

char file_name[50] File name to save the graphing data onto disk.

char title[50] Title for the XGraph.

char graph_name[50] Name of the XGraph.

char label_x[50] Name of X axis.

char label_y[50] Name of Y axis.

Methods:

void setLabels(char x, char* y, char* title, char* graphname, char* filename)*

Sets the values in the PlotType class.

void writeHeaderInfo(ofstream &fout)

Writes the header info out to file.

Population Class

The base class for the Cell and Fiber classes.

Variables:

*int *idReceivingCellPop Array of integers, size of NUM_TARGETS, that has the*

index of the cell population that receives input through a fiber?

*Int *TypeofsynapticJunction Array of integers, size of NUM_TARGETS, that*

identifies the type of synaptic junction.

1 = excitatory, 2=inhibitory, 3=long esc, 4=inh

*int *conductionTime*

*int *numProjectTerminals Array of integers, size of NUM_TARGETS, Number of*

terminals that are projected by a fiber of a cell

*Int *randomCellSeed of integers, size of NUM_TARGETS, Random number*

seed governing specific anatomical made by terminals of cells in populations

*Int *cellsFired Array of integers, size of numCellInPop, to hold whether*

a cell fires. 1 = fires, 0= cell did not fire.

int numCellInPop Number of cells on a population. As of

7/8/97 this is set too 100 because of dependencies with the Inventor code. Future work will make this more flexible.

int numTargetPop Number of populations projected to or targeted by that

fiber or cell population or number of types of target connection, if a sending population makes more than one type connection.

*int *spike Array of integers, size of numCellInPop, that contains*

whether a cell has spiked. 1= spike, 0 = no spike.

int numColumns Number of columns that a population has.

This is set to 10 because of dependencies with Inventor

code. Future work will make this more flexible.

int numRows Number of rows that a population has.

This is set to 10 because of dependencies with Inventor

code. Future work will make this more flexible.

*double *strenghtofOutputSynapses*

*double *widthofTerminal*

*double *heightofTerminal*

Methods:

void intializeArray(int NUM_TARGETS)

Dynamically allocate the arrays once the value of NUM_TARGETS is read from the

sim.ini file.

Int getNumCellInPop()

Returns an integer that is the number of cells in the populations.

Int getSpike(const int I)

Returns the integer value of spike[i].

Random Class

Generates a random value between 1.0 and the seed

Variables:

int seed The random seed.

Methods:

int getSeed()

Returns the value of seed.

Void setSeed(int i)

Sets seed = i.

double generateRandomNum()

Generates a random number.

Simulation Class

Used to run the simulation, takes care of allocation and deallocation of memory.

Variables:

int NUM_CELL_POP Number of Cell Populations read in from the sim.ini file.

int NUM_FIBER_POP Number of Fiber Populations read in from the sim.ini file.

int MAX_CONDUCTANCE_TIME_PLUS1 Read in from the sim.ini file.

int NUM_SYNAPTIC_TYPES Number of Synaptic types read in from the sim.ini file.

int CELL_ARRAY_WIDTH Cell Population's width read in from the sim.ini file.

int CELL_ARRAY_HEIGHT Cell Population's height read in from the sim.ini file.

int TIME_STEP_SIZE Size of each timestep in the simulation read in from the sim.ini file.

int MAX_NUM_CELLS Maximum number of cells in any one Populations read in from the sim.ini file.

int TOT_NUM_POP NUM_CELL_POP + NUM_FIBER_POP

int LTSTOP Last possible time step in the simulation read in from the data file.

Public

CellPopulation cellpop Pointer to a CellPopulation.*

FiberPopulation fiberpop Pointer to a FiberPopulation.*

SynapticType synaptictype Pointer to a SynapticType.*

ActiveNeuron activeneuron Pointer to a ActiveNeuron.*

FiringPop firingpop Pointer to a FiringPop.*

int spikeHeight Value of the spike height used in Inventor's quad mesh.

Framing frame Framing class.

Methods:

~Simulation()

Deallocates memory for the activeneuron, synaptictype, fibpop, and cellpop.

int getMaxCells()

Returns the value of MAX_NUM_CELLS.

int getTotNumPops()

Returns the value of TOT_NUM_POPS.

*void readIniFile(char *filename)*

Reads in the values of the sim.ini file into their appropriate values.

*void iniPops(char *datafilename, int start, int stop)*

Calls readIniFile to get values, then allocates memory and calls the ini methods for the main classes (Cell, Fiber, Synaptic, ActiveNeuron). Sets spike height to 15 and set the value of MAX_NUM_CELL_POP.

void start(char filename, int start, int stop)*

Calls iniPops then sets up mask for Motif to check event queue every 5 timesteps. Calls simulateOneStep for each timestep in the simulation, looping between start and stop.

void simulateOneStep(int timestep, int recordflag)

Completes one timestep of the simulation. If recordflag=1 then the cells fired are recorded into the activeneuron class.

void forwardStep()

SynapticType Class

Contains the properties for a type of synaptic junction. There are four types:

1 = excitatory, 2=inhibitory, 3=long esc, 4=inh.

Variables:

static int numSynapticTypes Total number of Synaptic types.

double timeConstPsp Time constant of psp.

double equilibriumPotential For synaptic type relative to 0 MV.

double dcs Decay constant for synaptic conductance.

Methods:

int getNumSynapticTypes()

Returns the total number of Synaptic types.

double gettimeConstPsp()

Returns the time constant of Psp.

double getequilibriumPotential()

Returns the equilibriumPotential.

double getdcs()

Returns the dcs.

void iniSynapticTypes(int NUM_SYNAPTIC_TYPES , int TIME_STEP_SIZE)

Initializes the dcs once the timeConstPsp is read from the data file.

Friend Methods:

Ifstream& operator>>(ifstream&, SynapticType&)

Overloaded >> operator used when reading the data file. Method reads in the data formatted for the SynapticType.

TwoDArray Class

Two dimensional array class that can be allocated dynamically.

Variables:

*double *tarray Pointer to array.*

int tmaxx Maximum value of x component of array.

int tmaxy Maximum value of y component of array.

Methods:

TwoDArray()

Constructor, initializes all values to 0 or NULL.

void iniTwoDArray(int x, int y)

*allocates memory for the array, size of x*y.*

double getValue(int x, int y)

returns the value at array index (x,y).

void setValue(int x, int y, double value)

Sets the value of (x,y) to value.

Appendix C

Software Description