1

NEXUS: A NEURAL SIMULATOR FOR INTEGRATING TOP-DOWN AND BOTTOM-UP MODELING

Paul Sajda, Ko Sakai,
Shih-Cheng Yen, and Leif H. Finkel

Department of Bioengineering and Institute of Neurological Sciences
University of Pennsylvania, Philadelphia, PA 19104-6392

ABSTRACT

We have developed the NEXUS simulation environment as a tool for modeling large-scale neural systems. The software is written in C and runs under UNIX. A unique aspect of NEXUS is that it is particularly suited for simulating hybrid neural models (i.e. systems integrating different modeling paradigms and/or architectures.[1]) NEXUS is designed for large-scale simulations, and to facilitate model development, testing and analysis it incorporates several major features: network architectures based on topographic maps, programmable neural units, scalable and modular simulation, support for common learning paradigms including the generalized Hebb rule and backpropagation, and a user-friendly interface. These features make NEXUS a useful environment in which to study the "perceptual" properties of various network architectures.

INTRODUCTION

The effort to understand the structure and function of the nervous system, through computer simulation, has proceeded along several paths. These range from detailed models of single neurons [1] to more abstract models consisting

[1] Examples of hybrid models include 1) different network architectures, e.g. a biologically-based front-end interfacing with a PDP-based recognition module or 2) different modeling levels, e.g. in some networks units represent single cells, in others units represent populations of cells.

of symbolic representation and heuristics [2]. One path which could be considered a middleground between these two extremes is large-scale neural systems (LSNS) modeling. This paradigm treats populations or groups of neurons as fundamental functional units, and typically trades-off detailed cellular modeling in favor of large numbers of units and connections (on the order of 10^5-10^6 units and 10^6-10^7 connections). LSNS modeling differs from connectionist modeling in that attention is paid to the neuroanatomy and neurophysiology at the systems level.

One area which has received considerable attention by LSNS modellers is visual perception [3][4][5][7][8][9]. The goal of these models is to develop an understanding of the neural mechanisms underlying the stimulus/percept transformation process. In general, these models tend to draw on information from neurophysiology, neuroanatomy, visual psychophysics and computational theory. However, a dilemma arises in that neurophysiology and neuroanatomy tend to characterize the neural processes most closely associated with the stimulus while psychophysics and computational theory often best characterize those associated with the perception. Since a complete model should consider the continuum of processes underlying the stimulus/percept transformation, a single level of structural or functional abstraction is not appropriate. Instead, the system is best modeled as a *hybrid neural system* [10][11], where the level of structural and functional detail can vary within the model through a combined top-down/bottom-up approach.

We have developed a neural simulation environment, called NEXUS, which is particularly suited for this hybrid paradigm and offers an ideal platform for combining top-down and bottom-up modeling techniques. The environment incorporates the following important features;

- network architectures based on topographic neural maps.
- programmable generalized neural (PGN) units.
- scalable and modular simulation.
- easy-to-use graphical user interface.

In the following sections we will begin by providing a brief overview of NEXUS. We will then discuss the specifics of each of its important features and the advantages they offer when simulating large-scale neural models. Finally, we will mention some of the models which have been developed under NEXUS, several

of which utilize a hybrid paradigm for modeling aspects of visual perception, and others which incorporate more traditional neural network architectures and paradigms, ranging from connectionist backpropagation models to simulation of local synaptic learning rules.

OVERVIEW OF NEXUS

NEXUS consists of core software and utilities for simulating a variety of neural models. The software is written in C and has an X Windows-based graphical user interface. The system currently runs on SPARC compatible platforms running under UNIX .

The NEXUS environment is divided into three stages, as shown in figure 1. The first stage is called *model design*. In this stage users create an architecture, specifying network parameters and designing model connectivity using the *NX description language*. In addition, users may customize the functional behavior of individual units through the use of *PGN functions*. Following model design, the system passes to the *simulation construction* stage. The NX file, describing the structure of the model, is parsed and data structures, representing individual units and the connections between them, are dynamically created. A "simulation builder" combines these data structures with user defined functions. The system then proceeds to the *user interface* stage, where the user spends the majority of his/her time interacting with the model, performing simulations, testing, and debugging.

IMPORTANT FEATURES OF NEXUS

NEXUS was designed with a number of novel features which decrease the turnaround time between design, simulation, and modification for LSNS models. These features, which are discussed below, range from a simplified scheme for defining a model's architecture to options which allow a model to run in parallel on multiple workstations.

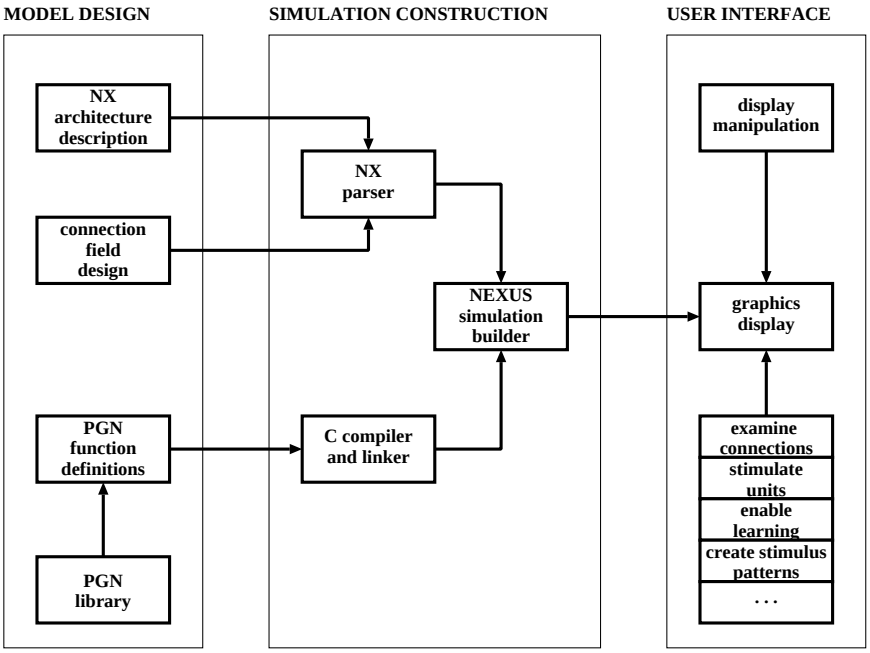

Figure 1 Three stages which make up the NEXUS simulation environment. A model's architecture and function are created in the *model design* stage using the NX description language and various function libraries. The model is dynamically built in the *simulation construction* stage. Finally, control is passed to the *user interface* stage, at which time the model can be simulated, tested, and modified by the user.

Topographic Neural Map Architecture

Within NEXUS a *neural map* is defined as a collection of units which perform the same function and have similar connectivity. The major properties of neural maps important to NEXUS are 1) topographic organization, 2) parallelism and 3) connection field based connectivity. These three properties allow a user to create large interconnected networks with a minimum of specification by exploiting redundancy between units and enforcing topographic constraints. For example, consider specifying connectivity in a large network. Explicitly defining individual connections would be cumbersome, if not intractable, especially if the number of connections is of the order 10^6 or 10^7. In NEXUS, instead of specifying individual connections a user creates a *connection scheme* for each map. Each connection scheme consists of one or more *connection fields* which define the spatial geometry, polarity, and synaptic weighting functions for establishing connectivity to other maps. Figure 2 shows an example of a connection scheme, consisting of two connection fields, for the neural map labeled Net_i. One connection field establishes retrograde connectivity between units in Net_i and Net_j while the other specifies anterograde connections between Net_i and Net_k.

The inset in figure 2 shows the parameters which the user can modify for controlling the location, shape, size, and weights of the connection field. The first step for defining a connection field is to specify a *target map*. The target map represents the map with which to make connections. Connections can be made between different maps or within a map. A *target unit*, representing the center of the connection field projected onto the target map, is implicitly defined by the shift parameters u and v. The default is to have target units "in register" ($u = v = 0$). The user has the option of controlling other parameters affecting the relative location of the target unit (e.g. defining how connections are made between maps of different sizes.) Once the target unit is set, all transformations and calculations are made relative to the local coordinate system having the target unit as the origin. The shape of the connection field can be either rectangular or elliptical, with dimensions given by w and l. The connection field can also be rotated, by an angle ϕ, within the local coordinate system. Finally, a user specifies a function, $F(x, y)$, for assigning the weights, $C_{ij}(x, y)$, within the connection field. There are a number of standard functions available (`difference of gaussians`, `exponential`, `random`, etc.). The user may also import values, contained in individual files, which may have been generated externally or determined from neurophysiological data.

The connection schemes, along with other parameters describing a model's ar-

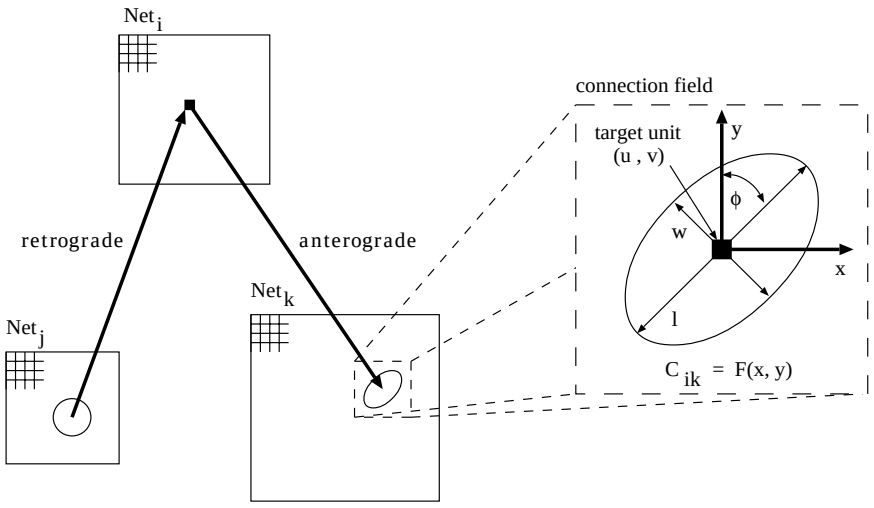

Figure 2 Illustration of the connection scheme paradigm for defining model connectivity. The figure shows the connection scheme, consisting of two connection fields, for units in map Net_i. One connection field establishes retrograde connectivity to Net_j while the other makes anterograde connections to Net_k. The inset shows the parameters which the user can modify to control the shape, size, location, and values of the connection field. Once the connection scheme is defined, NEXUS dynamically generates the individual connections for each of the units.

chitecture (such as number of units, dimensions of each map, transfer functions, etc.) are created using the *NX* description language. Figure 3 shows a segment of NX code. NX is a hierarchical language, employing an intuitive syntax and regular structure, making it straightforward to learn and understand. NEXUS takes the connection schemes and architecture described in the NX file and generates the individual connections for each of the units in the map. As we have mentioned, this method of establishing network connectivity offers the advantage that each individual connection need not be assigned by the user. In addition, this paradigm offers a degree of scalability into model development. When creating a model, a user often does not require a full scale implementation and instead would be best served by a smaller scale version for development and testing. However, once the system's desired architecture and functionality are reached, the user may wish to extend simulations to full size models. Since the NEXUS software is responsible for establishing connectivity, a user simply changes the single parameter specifying the size of the maps–NEXUS will automatically remap the connections.

Programmable Generalized Neural Units

In NEXUS, both the architecture of a model and the functional behavior of the individual units are controlled by the user. Unit function is specified as a *transfer function*, representing how a unit maps input to output. NEXUS offers a number of standard transfer functions, including `linear`, `sigmoid` and `binary`. However, in many cases a user requires more flexibility when specifying the functional characteristics of a unit. The function a given unit carries out will depend on its level of abstraction within the model–for instance whether it represents an individual neuron or an assembly of neurons. Customizing unit function within NEXUS is done through the use of *Programmable Generalized Neural* (PGN) units. PGN units are treated in the same way as conventional units except their transfer functions are user defined and are created using PGN library functions.

PGN units offer important advantages over conventional models employing strictly linear (and nonlinear) thresholding elements. The first is that a function representing the processing of a population of neurons can be embedded into a single unit. For example, consider the problem of modeling the dynamics of a network of neural oscillators. One possible model is to design each unit as a linear threshold element, determine the local excitatory and inhibitory connectivity necessary to generate local oscillations, and then couple these local populations to form a network. However, if the user is only interested in

```
Network Net_i {
        # of units            = 4096;
        x dimension           = 64;
        y dimension           = 64;
        transfer function     = sigmoid(0.0, 100.0, 1.0);
        threshold             = const(0.0);
        decay                 = 1.0;
        clamp                 = off;
        initial firing rate   = const(0.000000);
        evaluations per cycle = 1;
        connections{
         from Net_j {           /* connection field #1 */
               projection             = full;
               mapping type           = normalize;
               connection field shape = ellipse;
               length                 = 5;
               width                  = 5;
               rotation angle         = 0;
               shift x                = 0;
               shift y                = 0;
               weight function        = rand(1.0, -1.0);
               }
         to Net_k {             /* connection field #2 */
               projection             = full;
               mapping type           = normalize;
               connection field shape = ellipse;
               length                 = 7;
               width                  = 3;
               rotation angle         = 0;
               shift x                = 0;
               shift y                = 0;
               weight function        = exp(1.0, 0.0, 0.5);
               }
         }
}
```

Figure 3 A segment of NX code describing the architecture and connectivity of the map Net_i, shown in figure 2. Parameters specifying the size, dimensions, and transfer function are first assigned. Next the connection scheme is defined, shown here having two connection fields. The `projection` variable is used to specify an "area of interest", representing the units in Net_i to which the connection field should be applied. In this case the `projection` is set to "full"–the field applies to all units. The `mapping type` indicates how connectivity is established if source and target maps have different dimensions. The parameters `connection field shape`, `length`, `width`, `rotation angle`, `shift x`, and `shift y` control the shape, size, location, and local coordinate system of the connection field. Finally, `weight function` specifies the function or set of values for assigning connection weights.

modeling the network dynamics, independent of how oscillations are generated locally, then he/she can abstract a unit as a local oscillator representing the interaction of a small population of neurons.

A second benefit of using PGN units is they allow a modeller some control over CPU/memory tradeoffs. When simulating large-scale neural models on workstations one must consider the problem of finite computing resources, most notably limited amounts of fast memory (DRAM). Though UNIX workstations support virtual memory, the speed at which such memory can be accessed is several orders of magnitude slower than access to fast memory. Since much of the size and memory resources required by a model are taken up by *explicit* connections (connections which are stored in memory), reducing the number of explicit connections will increase the size of the models which can be simulated. PGN units, together with the regular architecture maintained by topographic maps, allow the user to create *implicit* connections. Implicit connections are not stored but instead are calculated "on the fly". For example, a unit can have implicit nearest-neighbor connectivity by creating a PGN function which computes the location (x-y position in the map) of its neighbors to find their output values. These values are then combined to form a weighted sum, which can then be used as the effective input for determining the unit's new output.

An example of a PGN function is shown in figure 4. This function is used in a simulation of coupled neural oscillators and implements an algorithm adapted from Baldi and Meir [12]. The model contains three maps, "Input", "Input_average" and "Phases". "Input" represents stimulus-driven activity, "Input_average" is a local average of the activity in "Input", and "Phases" represents the phase at which the "Input" units fire. The model shows how populations of neurons, modeled as coupled oscillators, can synchronize their firing rates when driven by a common input. The function of the code segment in figure 4 is to compute the phase of each oscillator unit in the network "Phases". A unit first loops through its connections (lines 8-25), determines if various inputs are above threshold (lines 17-20), and then augments its total input based on a coupling equation (lines 21-23). A example of implicit connectivity is shown for connections made between "Phases" and "Input" (lines 11-13) and "Phases" and "Input_average" (lines 14-16). The implicit connection is established by the PGN library function `get_cell_at_position`, which returns the firing rate of a unit given the name of a map and the desired location. The state of the maps after simulation is shown in figure 5A. One can see that the "Input" units responding to the rectangle tend to have "Phase" units which are synchronized. Figure 5B also illustrates this synchronization, showing histograms of the phases computed by the PGN function, before and after simulation.

```
 1  calculate_phase(cell, network, . . .)
 2          CELL            cell;
 3          NETWORK         network;
 4                            .
 5                            .
 6                            .
 7  phase_connections = cell->connect_list;
 8  for (count = 0; count < cell->number_connections; count++) {
 9    phase_network2 =
           get_network_id(phase_connections->input_cell->net_id);

10    /* implicit connections */
11    Aj = get_cell_at_position("Input",
12                              phase_cell_x,
13                              phase_cell_y)->firing_rate;
14    Aj_avg = get_cell_at_position("Input_average",
15                                  phase_cell_x,
16                                  phase_cell_y)->firing_rate;
17    if((Aj_avg - T) > 0)
18      Fj = 1.0;
19    else
20      Fj = 0.0;
21    sum = sum + Ai * Aj * K * Fi * Fj *
22      (float)sin(phase_connections->input_cell->firing_rate_old -
23                 cell->firing_rate_old);
24    phase_connections =  phase_connections++;
25  }
26                            .
27                            .
28                            .
```

Figure 4 A segment of a PGN function used for computing the phases of coupled oscillators. The function abstracts the interaction of linear thresholding units as a single nonlinear differential equation. The function also incorporates implicit connectivity.

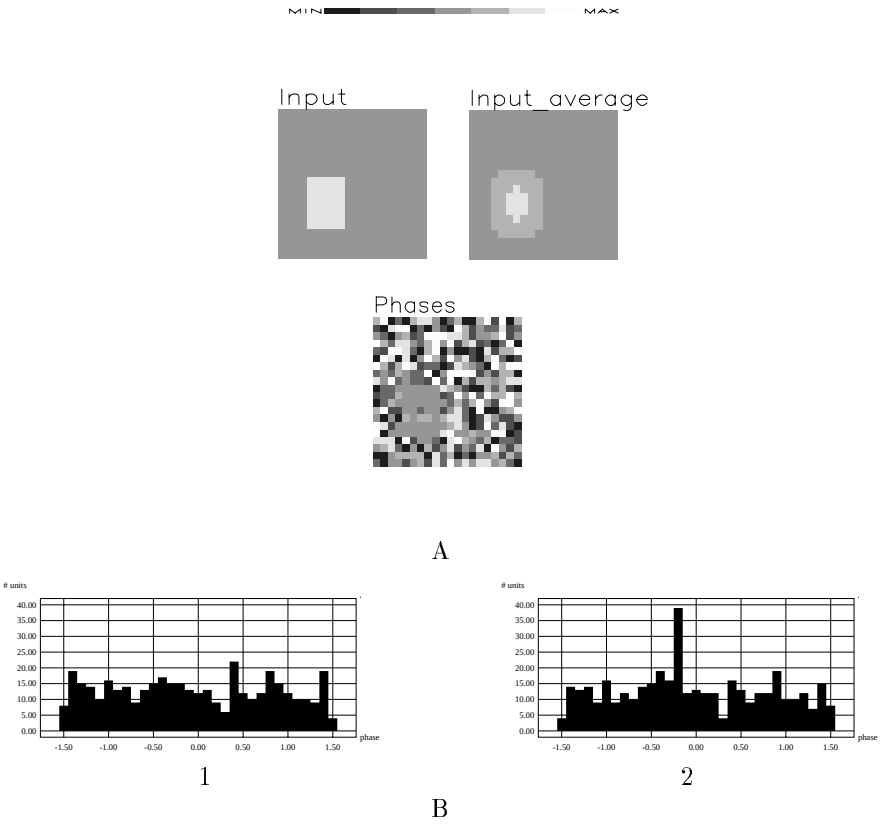

Figure 5 **A** Display of the model which implements the PGN function shown in figure 4. The three maps are shown after simulation (note the legend represents phase between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$ and applies only to "Phases"– the firing rates in "Input" and "Input_average" are on a different scale). The input to the system is a small rectangle, shown as activity in the lower left of "Input". "Input_average" computes a local average of activity in "Input", while "Phases" represents the phase at which units in "Input" fire. Note the phase-locking (common grey-level) of those units in the lower left of "Phases". **B** Further indication of phase-locked firing is seen in the phase histogram before (1) and after (2) simulation. The peak around -0.2π in (2) represents synchronization of units responding to the rectangle.

Scalable and Modular Simulation

To make LSNS modeling practical and manageable, we have incorporated features into the NEXUS environment which allow progressive and modular model development. As we have mentioned previously, topographic maps and the connection scheme paradigm offer the user a degree of scalability. Since connectivity is established automatically by the software, the size of individual neural maps can be modified independent of connection definitions. This allows for a quick transition from small scale design and development to large scale testing.

Perhaps just as important as scalability is the option for modular simulation. LSNS models often consist of different modules or subsystems, each of which represents the aggregate function of many individual neural maps. In many instances a user may wish to design and test these subsystems independent of one another, linking them into a complete model at some later time. In addition, practical reasons such as processing limitations of conventional computers make it advantageous to support modular simulation. For example, simulation times can be decreased if different modules can be simulated simultaneously on several workstations which communicate via a local area network.

NEXUS offers the added feature of modular simulation, incorporating a protocol for partitioning complete models into distinct modules which can then be simulated simultaneously on multiple workstations. The first step in constructing a modular model involves the user partitioning the complete model into individual subsystems. This partitioning is specified in the NX file as a list of neural maps which should be grouped together. This list is the only direct action that is required by the user. The NX file describing the complete system is parsed by a NEXUS utility, which creates new NX files for each module and establishes the communication protocol so that they can be simulated separately. The user then starts an *instance* of NEXUS for each module. An instance consists of a module and its associated NEXUS process. Instances communicate via a shared file system, such as a local disk. Since disk access can be slow, a user should partition a complete model so that the majority of communication (connectivity) is within a module and not between modules.

Figure 6 illustrates how two complete models can be divided into separate modules. In each case four instances are created, denoted as N-1, N-2, N-3, and N-4. In figure 6A, the complete model consists largely of parallel modules, and therefore benefits from the coarse-grained parallelism offered by NEXUS. Figure 6B is a model constructed of sequential modules. Simulation times for these types of models can be reduced by a pipeline between instances. In summary, the

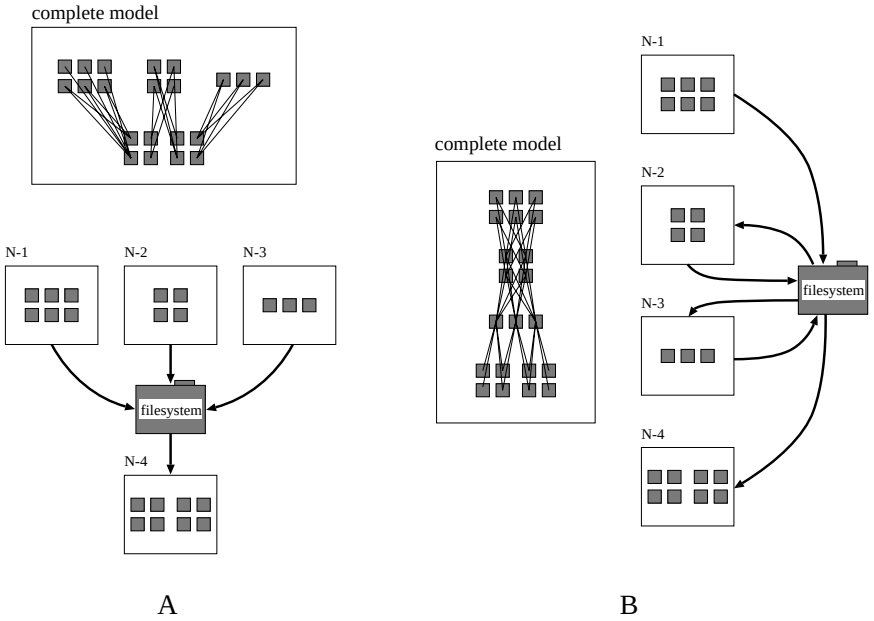

Figure 6 Two examples of the types of models which benefit from modular simulation. **A** A model with groups of maps operating in parallel can be divided into parallel instances N-1, N-2, N-3, the output of which are integrated by N-4. Since N-1, N-2, and N-3 can be computed in parallel on separate workstations, simulation time is decreased. **B** A model which can be divided into sequential instances. Simulation time is decreased by a pipeline between instances.

modular simulation capabilities of NEXUS facilitate the modeling process by providing a protocol for integrating independent subsystems to form complete models and offering a coarse-grained parallelism for distributing computation among workstations in a local network.

Graphical user interface

The amount of data and number of parameters associated with large scale neural simulations makes user interfaces an important part of a simulation environment. We have designed NEXUS around a menu-driven graphical user interface which is designed to simplify data analysis and testing. The interface runs under X Windows and requires the XView libraries. A large graphics

window displays the arrays of units which form the neural maps. If a given model is divided into several NEXUS instances then each instance has its own interface and displays only those neural maps associated with that instance. However, since X Windows is a networked window system all GUIs can be displayed on a single local workstation.

The graphics window can be set to either display the activity of individual units or the relative connection strengths between units. In addition, the user can set the display to update in real-time or at the end of the simulation, with the latter being considerably faster. The graphics display is built on top of a 3-D graphics environment, allowing the user to create a customized layout by changing the three dimensional positioning of the neural maps. A typical NEXUS display, consisting of the main graphics window and several menus, is shown in figure 7.

Various menus allow the user to interactively examine and change parameters in the model. For example, menus are available for changing connection field weights and transfer functions of networks which are currently loaded into the simulator. Allowing the user to make such modifications online is advantageous since the alternative would be to make a change to the NX file and rebuild, a option which is time consuming for large models. A menu is also available for exporting data for display and analysis by other utilities and software.

We have incorporated menus for several common learning paradigms, such as backpropagation and the hebb rule. Creating an easy-to-use menu system which directly supports these paradigms makes the simulator well suited for educational use. Another component of the interface which makes the system easy-to-use for a novice is the mouse/locator. The mouse is not only used to access and control the menu system but it also is used to directly manipulate the neural maps in the graphics display. For example, point-and-click operations can move and position maps to create a custom layout. Alternatively the mouse can be used as an "electrode" and positioned over individual units in order to examine their parameter values. Finally, the user can examine the connectivity between individual units. Clicking on a given unit results in the highlighting of all of its retrograde connections, the color reflecting the strength of the connection.

SUMMARY

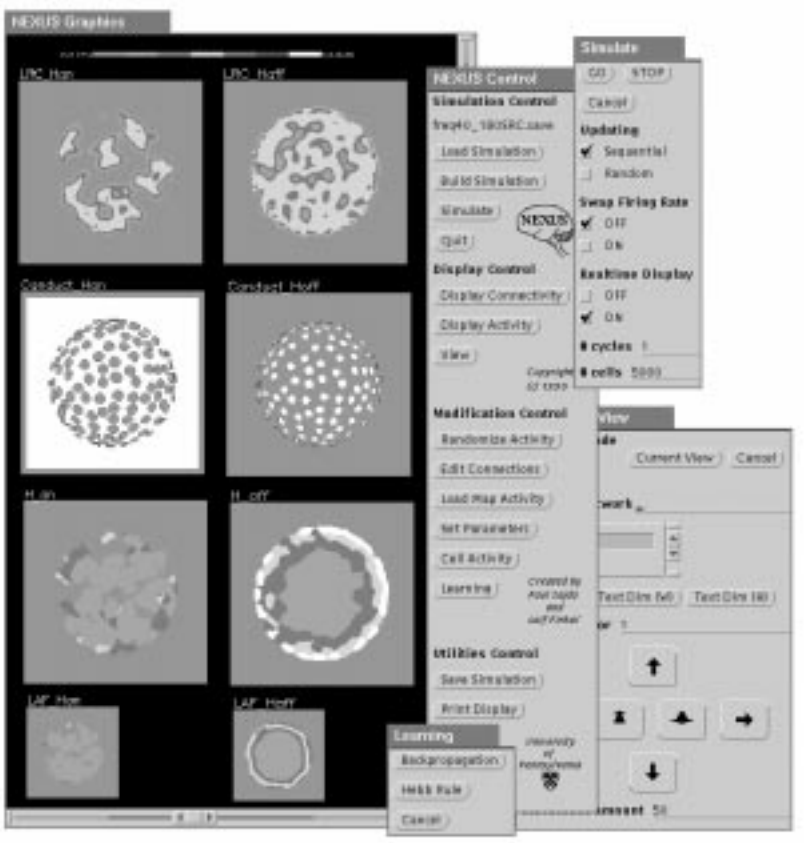

Figure 7 NEXUS user interface. Shown are the main graphics display and several menus.

MODEL	# UNITS	# CONNECTIONS	
		explicit	implicit
depth-from-occlusion (Sajda and Finkel, 1992)	1.8×10^5	2.0×10^6	1.0×10^7
shape-from-texture (Sakai and Finkel, in prep)	1.1×10^6	–	4.7×10^8
color perception (Courtney et al., 1992)	1.1×10^4	1.6×10^6	–
spatial transformations by the superior colliculus (Groh and Sparks, 1992)	100	400	–
backpropagation (leaf recognition)	2000	2.7×10^4	–
cortical map organization (somatosensory cortex)	1000	3.1×10^4	–
texture discrimination (energy model)	1.5×10^4	2.0×10^5	–

Table 1 Models which have been simulated using NEXUS.

Table 1 lists some of the models which have been simulated using the NEXUS software. Several LSNS models of human visual perception have been designed and tested using the NEXUS environment. Smaller systems, such as a model of head-eye coordinate transformations carried out by the superior colliculus, have also been developed using the NEXUS software. NEXUS is used as part of a graduate course in Computational Neuroscience at the University of Pennsylvania and some of the simulation laboratories have included 1) leaf recognition using backpropagation, 2) reorganization of somatosensory cortical maps via the Hebb rule, 3) development of ocular dominance columns and 4) preattentive texture discrimination. Plans for future development include extensions to the PGN function library and incorporation of approximation/interpolation network paradigms, for example based on radial basis functions, directly into the base environment. We are also currently investigating porting the software for use on massively parallel architectures.

Since the support for the PGN function abstraction implies that a unit can represent arbitrary neural processes, a user can conceivably construct models which incorporate details at the cellular level, having PGN units emulate ion channels, individual compartments, synapses, etc. However, NEXUS was not designed to be a general neural simulation environment, and models posed at

the cellular level would be best suited for systems such as GENESIS [16]. Our aim in developing NEXUS has been to create a simulation environment for developing models of large-scale neural systems which may lie at the interface of bottom-up and top-down methodologies.

NEXUS is available free of charge and users interested in obtaining the NEXUS software should send e-mail to nexus@ganymede.seas.upenn.edu.

Acknowledgements

This work is supported by grants from The Office of Naval Research (N00014-90-J-1864), The Whitaker Foundation, and The McDonnell-Pew Program in Cognitive Neuroscience. P.S. is supported by an ONR-NDSEG Fellowship.

REFERENCES

[1] C. Koch and I. Segev, editors. *Methods in Neuronal Modeling: From Synapses to Networks*. MIT Press, Cambridge, MA, 1989.

[2] S. M. Kosslyn, R. A. Flynn, J. B. Amsterdam, and G. Wang. Components of high-level vision: A cognitive neuroscience analysis and accounts of neurological syndromes. *Cognition*, 34:203–277, 1990.

[3] S. Grossberg and E. Mingolla. Neural dynamics of form perception: Boundary completion, illusory figures, and neon color spreading. *Psychology Review*, 92:173–211, 1985.

[4] T. Poggio, E. B. Gamble, and J. J. Little. Parallel integration of vision modules. *Science*, 242:436–440, 1988.

[5] L. Finkel and G. Edelman. Integration of distributed cortical systems by reentry: A computer simulation of interactive functionally segregated visual areas. *Journal of Neuroscience*, 9:3188–3208, 1989.

[6] J. Malik and P. Perona. Preattentive texture discrimination with early vision mechanisms. *Journal of the Optical Society of America A*, 7(5):922–932, 1990.

[7] J. Malik and P. Perona. Preattentive texture discrimination with early vision mechanisms. *Journal of the Optical Society of America A*, 7(5):922–932, 1990.

[8] L.H. Finkel and P. Sajda. Object discrimination based on depth-from-occlusion. *Neural Computation*, 4(6):901–921, 1992.

[9] J. Skrzypek and B. Ringer. Neural network models for illusory contour perception. In *Proceedings of the IEEE Computer Vision and Pattern Recognition*, pages 681–683, 1992.

[10] J. M. Bower. Reverse engineering the nervous system: An anatomical, physiological, and computer-based approach. In S. F. Zornetzer, J. L. Davis, and C. Lau, editors, *An Introduction to Neural and Electronic Networks*, pages 3–24. Academic Press, 1990.

[11] P. Sajda and L.H. Finkel. Simulating biological vision with hybrid neural networks. *Simulation*, 59(1):47–55, 1992.

[12] P. Baldi and R. Meir. Computing with arrays of coupled oscillators: An application to preattentive texture discrimination. *Neural Computation*, 2(4):458–471, 1990.

[13] P. Sajda and L. H. Finkel. Object segmentation and binding within a biologically-based neural network model of depth-from-occlusion. In *Computer Vision and Pattern Recognition*, pages 688–691, 1992.

[14] S. M Courtney, G. Buchsbaum, and L. H. Finkel. Biologically-based neural network model of color constancy and color contrast. In *International Joint Conference on Neural Networks*, pages 55–60, 1992.

[15] J. M. Groh and D. L. Sparks. Two models for transforming auditory signals from head-centered to eye-centered coordinates. *Biological Cybernetics*, 67:291–302, 1992.

[16] M.A. Wilson, U. S. Bhalla, J.D. Uhley, and J.M. Bower. GENESIS: A system for simulating neural networks. In *Advances in Neural Network Information Processing Systems I.* Morgan Kaufmann Publishers, San Mateo, CA, 1989.