TECHNICAL ARTICLE

SIMULATION 59:6,358-364
© 1992, Simulation Councils, Inc.
ISSN 0037-5497/92 $3.00 + .10
Printed in the United States of America

NEXUS: A simulation environment for large-scale neural systems

Paul Sajda and Leif H. Finkel
Department of Bioengineering and Institute of Neurological Sciences
University of Pennsylvania
Philadelphia, PA. 19104-6392

NEXUS is a novel simulation environment designed for modeling large neural systems. The simulator allows the user to incorporate complex functional properties and symbolic processing at the level of the individual unit. In addition, NEXUS takes advantage of the principle of topographic map organization, found throughout the mammalian nervous system, to facilitate the modeling and design process. An easy-to-use graphical interface allows the user to interactively build and test models. This paper describes the principles underlying the NEXUS design and its advantages over other current simulation approaches.

Keywords: Neural networks, neural systems, bioengineering, interactive simulation, symbolic processing.

Introduction

Simulation has become an important technique in understanding the functional mechanisms of neural systems. The major problem encountered when designing or using a neural network simulator stems from the multiple levels of organization, from molecular to behavioral, at which neural systems can be studied. Simulations of simple nervous systems incorporating detailed neurobiological data, require a significantly different modeling approach than simulations of connectionist or artificial neural networks. In addition, models concerned with high-level perceptual and cognitive processing often have their foundations in psychology and psychophysics and therefore the details of their architecture may differ from physiologically - based studies. Our research interests deal with integrating data regarding brain structure, particularly that of the cerebral cortex, together with data from psychophysics in order to construct models of perception. These models tend to be very large, on the order of 10^5 units and 10^6 connections, and they require a great deal of computational power and storage capacity

Several software packages are currently available to the neural modeler. Some systems, such as the Rochester Connectionist Simulator (Goddard, et al., 1987), are suited for large-scale simulation of artificial neural networks. Other simulators, such as GENESIS (Wilson et al., 1989) and SLONN (Wang and Hsu, 1990), are capable of modeling both biological and artificial neural networks, however the generality that they introduce comes at the cost of practical limitations on model size-

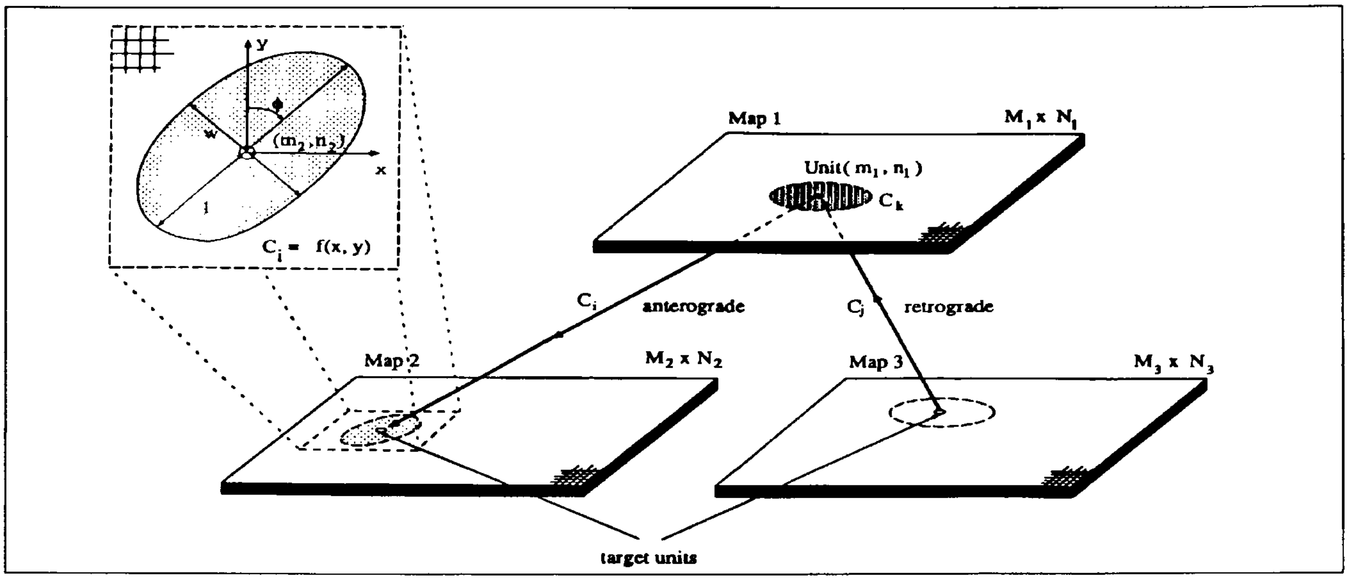


Figure 1: Schematic of a neural map architecture. Connection schemes are shown for a unit in map 1 at location (m_1, n_1). Shaded regions show the extent of the connection fields. (inset) The connection field around the target unit (m_2, n_2) and the associated parameters defining its spatial characteristics. (Note that the grid lines indicate that these maps contain multiple units.)

(i.e. an upper size limit of roughly 10^4 units for biologically detailed models. Since none of the available systems are adequate for the types of models we wish to simulate, we have developed the NEXUS simulation environment. The capabilities we require of our system include:

- a simple, easy-to-use methodology for building and testing very large neural models
- an efficient means of storing and simulating large models
- a generalized method of simulating the functional properties of neural circuits and assemblies, which can then be used as building-blocks of larger networks
- a provision for integrating different network paradigms into a single simulation

To realize these capabilities we have built three distinguishing features into NEXUS:

- a language for describing network architecture which is based on the topographic organization of neural maps
- a programmable neural unit with a language for defining its functional operation
- support for the simulation of hybrid networks

Though not a general-purpose neural network simulator, NEXUS offers enough flexibility to be used for a variety of network models, particularly those which require large numbers of units. In addition, the support for hybrid systems and programmable units allows a modeler to integrate symbolic processing and heuristics within a highly distributed system.

Design Principles

Neural map architecture

For our purposes, a neural map consists of a collection of units with similar connectivity and functional properties. Map organization has been found in all areas of the cerebral cortex as well as in many subcortical areas. The three major properties of neural maps which are critical to the NEXUS design are:

- Topographic organization
- Parallelism
- Receptive field (connection field) based architecture

One of the major difficulties in simulating large neural networks is defining the network connectivity. Explicitly defining connections can be cumbersome, and in fact intractable, when the number of connections is on the order of 10^5 or 10^6. In NEXUS, individual neural connections are not specified, rather a user creates a general connection scheme for each map. The system uses topology and parallelism to establish the connections for each of the units in the map. Figure 1 illustrates how a typical set of three maps could be connected. The number of units in a map is reflected by its dimensions. A graphical representation of the connection scheme is shown for a unit in map 1, located at (m_1, n_1). In this example, the unit in question has anterograde connections to map 2 (connections projecting from map 1 to map 2) and retrograde connections from map 3 (connections projecting from map 3 to map 1). In addition it has internal connections within its own map. Connections are defined by a target unit and a connection field . The target unit represents the geometric center of the connection pattern and is determined by a topological mapping which takes map size and spatial transformations into account.

The connection field (CF) is defined as the function which specifies the spatial extent and connection strengths (weights) for the connections about a given target unit. Note that the connection field implicitly defines a unit's receptive field, the area which when stimulated affects the output of the unit. Physiologists have often used receptive fields as a means of classifying the functional properties of neurons. In NEXUS a CF is the kernel in the spatial convolution;

$$I(i,j) = \sum_{k=-d_1}^{d_1} \sum_{l=-d_2}^{d_2} C_{i,j}(k,l)\, O(i+k, j+l) \qquad (1)$$

The input to a particular unit, $I(i,j)$, is given by the convolution of its connection field, $C_{ij}(k,l)$, with the outputs, $O(i+k,j+l)$, of those units from which it receives connections. The inset in figure 1 is an expanded view of a CF about the target cell (m_2,n_2). The dimensions and orientation of the CF are specified by three parameters: length l, width w, and the angular rotation, ϕ, about the x-y axis. The weight assigned to a particular connection within the CF is a function of the position of the given unit relative to the target unit.

A neural map architecture provides several powerful advantages in building and simulating neural models. The first is the savings in time and effort required to build and implement the model. Since a modeler need only define connection schemes, not individual connections, a major amount of the tedious design process is removed. A second advantage is the inherent scalability introduced with neural maps. In designing a model, often one does not require a full size implementation and in fact a scaled version of the model can increase speed and save on development time. Since connectivity in neural maps is functionally specified, rescaling the size of the networks is trivial, with the simulator recalculating the individual connections. Finally, the structural organization of neural maps makes them particularly well suited for models which involve processing spatial information, such as vision.

The NX network architecture specification language

The neural map architecture is specified within the NEXUS environment using the language NX. NX is a hierarchical language which allows a user to quickly and easily create a model. It incorporates an intuitive syntax and regular structure, making it straightforward to learn and understand. The main features of the NX syntax are illustrated in the following example.

Figure 2 is a segment of NX code which defines a model consisting of two networks (a neural map and a

```
/* segment of NX network specification code */

Retina {
  number of units      = 4096;
  x dimension          = 64;
  y dimension          = 64;
  initial firing rate  = file(image.in);
  transfer function    = sigmoid(0.0, 100.0, 1.0);
}

LGN {
  number of units      = 4096;
  x dimension          = 64;
  y dimension          = 64;
  transfer function    = sigmoid(0.0, 100.0, 1.0);
  connections {
        from Retina {
          mapping type               = direct;
          connection field shape     = ellipse;
          length                     = 7;
          width                      = 7;
          rotation angle             = 0;
          shift x                    = 0;
          shift y                    = 0;
          weight function            = dog(1.0, 1.0, 1.5, 1.0);
    }
        to LGN {
          mapping type               = direct;
          connection field shape     = ellipse;
          length                     = 20;
          width                      = 20;
          rotation angle             = 0;
          shift x                    = 0;
          shift y                    = 0;
          weight function            = exp(1.0, 1.0, -1.0);
    }
  }
}
```

Figure 2: Example of NX code describing a model with two neural maps.

network are considered the same object within NX). The architecture of the model is defined in a single NX file and constitutes the simulation structure . Within the file are the descriptions of network structures . The two networks defined in figure 2 are named "Retina" and "LGN". The number of units in the network "Retina" is 4096 and its spatial dimensions are 64 x 64 . An initial firing rate of the neural units is specified, usually as a function of a random variable or loaded from a matrix of values. The transfer function transforms a unit's input (as defined by the CF convolution in equation 1) into its output. The specifics of transfer functions will be discussed in more detail in the section on PGN units. All units in a given network have the same transfer function.

Within a network structure exist several connection structures . These represent the definitions of the connection fields and spatial transformations for determining the target unit, and thus define the connection schemes discussed previously. For "Retina" in figure 2, no connection structures are defined. In contrast, "LGN" has two connection declarations. The first set of connections are retrograde, indicated by the syntax **from**, originating from the "Retina"; the second set are anterograde, indicated by the syntax **to**, and remain within the "LGN". The shape of the connection field can be either elliptical or rectangular, with the dimensions specified by the user. The spatial transformations of the target connections are set with the **rotation angle** , **shift x** and **shift y** declarations. Control of these parameters allows the user considerable flexibility in defining the location of the CF. Finally the **weight function** determines the spatial distribution of the weights for the CF. CFs can be specified by explicit functional descriptions or can be defined in a file containing a matrix of weights. In Figure 2 the first set of connections has an explicitly defined weight function

dog(), which denotes a "difference of gaussians". In this case the weight of the connection for the unit at location (x,y) is;

$$C(x, y) = \frac{1}{2\pi}\left[S_{ex}\left(\frac{1}{\sigma_{ex}} e^{\frac{-x^2}{2\sigma_{ex}}}\right)\left(\frac{1}{\sigma_{ex}} e^{\frac{-y^2}{2\sigma_{ex}}}\right) - S_{in}\left(\frac{1}{\sigma_{in}} e^{\frac{-x^2}{2\sigma_{in}}}\right)\left(\frac{1}{\sigma_{in}} e^{\frac{-y^2}{2\sigma_{in}}}\right)\right] \tag{2}$$

where σ_{ex} and σ_{in} are standard deviations and S_{ex} and S_{in} are scaling factors of the excitatory and inhibitory gaussian components. This type of CF can be used to extract edges from an image, and its center-surround structure (central region of excitatory connections and a surrounding region of inhibitory connections) has been widely used in modeling early vision (Marr, 1982). The second set of connections has a circularly symmetric exponential CF, as indicated by the syntax **exp()** .

Larger simulations, having more than two maps, are created by extending the code in figure 2. It is important to note how easy it is to define a model with NX. The limited syntax and hierarchical organization make defining a model more a process of establishing specifications than writing programming code. In fact one can simply use a template, similar in structure to the code in figure 2, and fill-in the desired values of the parameters.

It should be clear from examining the NX code that the functional properties of a given unit depend on its CF and transfer function. The CF transforms outputs of units into the input of a given unit, while the transfer function maps input to output. A typical transfer function is given by the sigmoidal function;

$$O(i,j) = \frac{1}{1 + e^{\frac{-I(i,j) - \theta}{S}}} \tag{3}$$

where O(i,j) is the output of the cell located at (i,j) , I(i,j) is the input to unit (i,j) (determined in equation 1), θ is a thresholding constant and S is a constant which controls the gain of the sigmoid. Many neural network models use this type of transfer function, most notably backpropagation models which require continuous derivatives (Rummelhart and McClelland, 1986). However this model of a neuron's response is not applicable to all types of neural simulations. We therefore have included in NEXUS the capability of defining a unit's transfer function. These Programmable Generalized Neural (PGN) units not only can have arbitrary transfer functions, but can execute a logical sequence of code, making for an integration of symbolic and distributed computation.

Programmable generalized neural units

A second fundamental design principle of NEXUS involves the use of a novel network construct we call PGN (Programmable Generalized Neural) units. These units are capable of performing more complex operations than conventional models of single neurons. Each PGN unit can execute a set of commands analogous to a sequential instruction set. A single PGN unit can in fact be substituted for a small network or local assembly of neurons. By constructing PGN units, the modeler can generate functional properties without simulating the low-level details of individual units. PGN units thus represent a compromise between a top-down and bottom-up modeling approach.

PGN units are particularly attractive in situations where the exact nature of the underlying neural operation is unknown (so modeling at a detailed level is useless), yet the computational nature of the problem is intense, requiring masses of cells each performing simple operations. In these cases, a single PGN unit can replace a whole hierarchy of conventional units. In addition to such generalized functions, PGN units can also carry out more typical cellular or network type operations (such as scaling and adding inputs, thresholding, and computing output functions). Thus, one has the option of designing a system where, in different maps, the individual units are modeled at any level between neurons and assemblies.

Another advantage of modeling with PGN units concerns the implementation problems associated with current computing technology. Limited memory and speed, as well as the obvious disadvantage of modeling a highly parallel system on a sequential machine, constrain the size of most networks. Until fast parallel machines become available on a wider scale, large neural models consisting of simple neurons will only be able to simulate rudimentary aspects of neural processing, and higher processes such as perception and cognition will remain unapproachable. By using maps of PGN units in place of large networks of simple neurons, one can save on the memory required to store individual cells and their connections. Depending on the particular implementation, a user can save both system memory and execution time.

Figure 3 illustrates how a single PGN unit can functionally replace a small network of simple units. This example models a mechanism for determining the local orientation of line segments. Figure 3a shows a small set of cells, connected so that they are selective to 45° lines. For this implementation there are nine input units, one response unit and nine connections. The firing rate of the response unit, as a function of line orientation, is shown in figure 3b. Figure 3c is a segment of code for the transfer function of a PGN cell which is also selective to 45° lines. This particular transfer function is called **diagonal_line( )** and would be referenced in the NX file for this model. PGN units are programmed using the C Programming Language and a set of library functions unique to NEXUS. The first function call in the PGN code gets the target unit using the PGN library function **get_cell_from_connection()** .

We will not go into the details of the syntax except to say that we have incorporated an object-oriented style for defining PGN functions. With the target cell location in the input map, we find the firing rates of neighboring cells using the functions **get_nearest_neighbor()** and **get_firing_rate().** By multiplying the firing rates by a set of constants equal to the weights in figure 3a, we form the input I(i,j). Finally, using a transfer function, such as in equation 3, we determine the firing rate of the response cell. In this example, we still require nine input cells and a response cell (the response cell would in fact be the PGN unit executing the code in figure 3c). However, in the PGN model, only one explicit connection (the connection to the target unit) is stored. The other connections are implicit and are computed using the **get_nearest_neighbor()** function. The savings in the number of connections (one connection for the PGN implementation of figure 3c versus nine for the architecture in figure 3a) may seem minimal at this scale, but for large simulations and more complex architectures the savings can be enormous.

The previous example is just one way in which PGN units can be used in NEXUS simulations. Some models may not warrant this structural abstraction. However, the example illustrates how a function, presumably performed by many simple units, can be incorporated into the operation of a single complex unit. Models of complex biological processing, in which a corresponding simple unit architecture is difficult to construct, can also be simulated using the PGN paradigm (Sajda and Finkel, 1992).

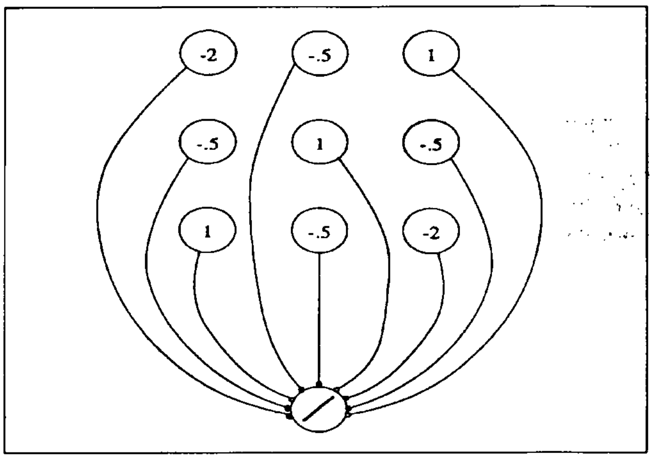


Figure 3A: Example of a simple architecture for orientation selectivity. Configuration which is selective to 45° lines. The 3 x 3 input array is connected to the response cell (unit with dark 45° line segment), with the value of the weight indicated by the value in the input unit.

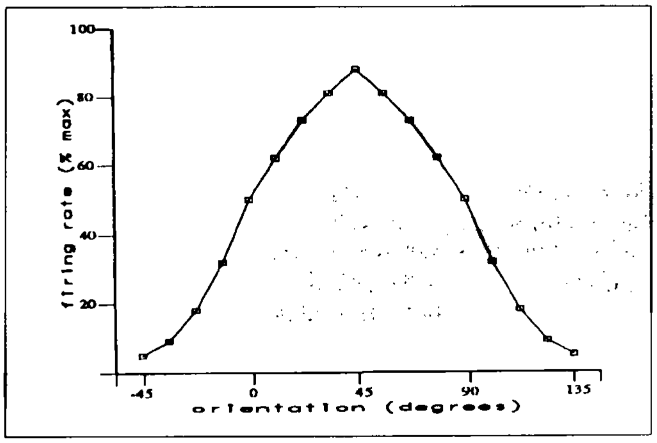


Figure 3B. Firing rate of the response cell as a function of orientation of the stimulus.

```
/* Code for PGN unit which is selective for 45 degree lines. */

diagonal_line(cell, network)
CELL cell;
NETWORK network;
{
int input_sum;
CONNECTION connection;
CELL target_cell;


/* Get the target (center cell) of the implicit receptive field. */

target_cell =
  get_cell_from_connection(get_network_connection("Input",
                                                  get_connection_list(cell),
                                                  get_number_conections(cell)));

/* Sum the inputs for the target cell and its nearest neighbors.
   Note that nearest neighbors are referenced by their relative
   location (in degrees) to the target cell. */

input_sum +=  1.0 * get_firing_rate(target_cell);
input_sum +=  -.5 * get_firing_rate(get_nearest_neighbor(target_cell,0));
input_sum +=   .5 * get_firing_rate(get_nearest_neighbor(target_cell,45));
input_sum +=  -.5 * get_firing_rate(get_nearest_neighbor(target_cell,90));
input_sum += -2.0 * get_firing_rate(get_nearest_neighbor(target_cell,135));
input_sum +=  -.5 * get_firing_rate(get_nearest_neighbor(target_cell,180));
input_sum +=   .5 * get_firing_rate(get_nearest_neighbor(target_cell,225));
input_sum +=  -.5 * get_firing_rate(get_nearest_neighbor(target_cell,270));
input_sum += -2.0 * get_firing_rate(get_nearest_neighbor(target_cell,315));

/* set the cells activity using a sigmoid with maximum value 100,
   minimum value 0 and centered at 0 (units has 0 threshold). */

set_firing_rate(cell,sigmoid(input_sum, 100.0, 0.0, 0.0));
}
```

Figure 3C. PGN code for 45° orientation—selective cell.

Hybrid networks

Since our models incorporate several different levels of organization, from single cells to more complex perceptual processing, we have built into NEXUS the capability of simulating hybrid neural networks . Our definition of a hybrid network is a set of interconnected neural networks which are based at different levels of abstraction. For example, an object recognition system might be designed to incorporate both connectionist and biologically-based networks. In this case the image would be preprocessed by units whose connection fields loosely conform with biological data. However, instead of modeling a biological memory mechanism, the system might learn to recognize particular objects by training with backpropagation (see Rummelhart and McClelland, 1986 for details of the backpropagation algorithm). The learning rule used by the recognition units should not affect the preprocessing units, but both sets of units must be capable of exchanging information. The model therefore consists of two sections, a biologically-based preprocessing module and a connectionist recognition module.

Simulating hybrid networks is not difficult to incorporate into a simulator, and in fact the simulators which have been previously mentioned should be capable of simulating such hybrid systems. The point we wish to make is that since our models require many different levels of abstraction, the principle of hybrid networks is an intrinsic property of the NEXUS design.

Simulator Operation

Running a simulation with NEXUS consists of three phases: model specification, compilation, and interactive display. Figure 4 illustrates the system's organization

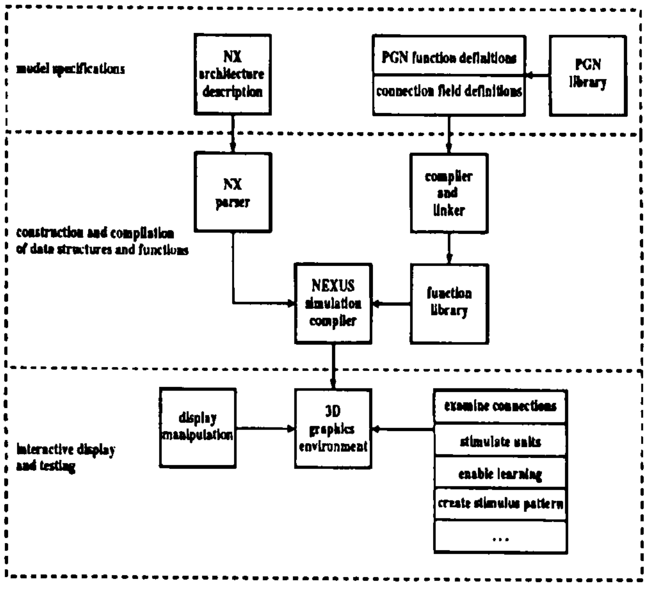


Figure 4: Organization of the NEXUS system.

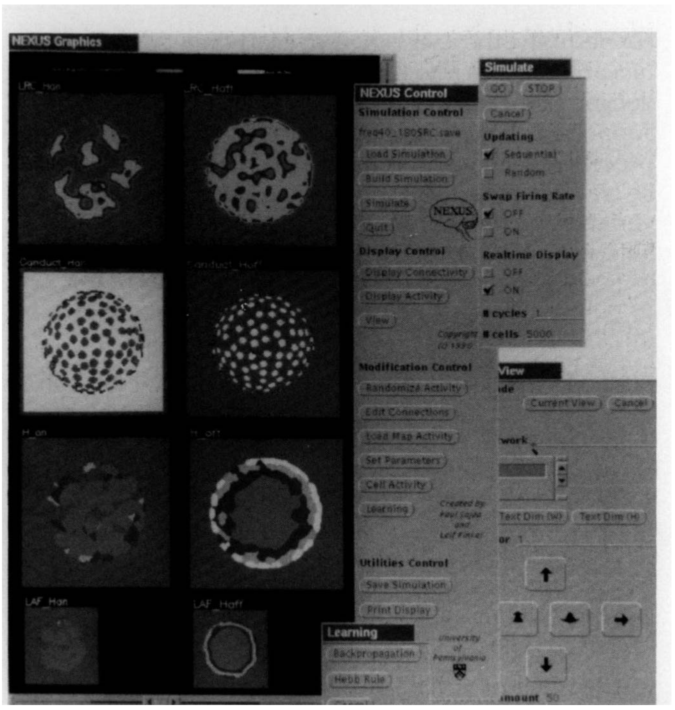


Figure 5: NEXUS graphics display.

and control flow. The first task for the NEXUS user is creating the specifications for the model. As discussed previously, the architecture of the model is defined within an NX file. In addition, the user can define transfer functions for PGN units. If the model requires connection field architectures which are not part of the standard NEXUS library then these too can be created.

Once the proper specifications are written, NEXUS compiles and constructs the functions and data structures. The NX architecture file is parsed and the neural maps are dynamically built as a linked list. This is in contrast to several other simulators which require the declaration of static arrays. The main advantage of dynamic construction is that one does not need to recompile the simulator for each model. In addition, dynamic construction supports models which require unit or connection replication, such as meiosis networks (Hanson, 1990). An additional step of data reformatting and compression is used to align the data structure in working memory and reduce its overall size.

All phases of NEXUS are menu driven, using an X Windows based graphics interface. A 3D graphics environment is used to display the model, allowing the user to manipulate the neural maps, units and connections as if real objects. An attractive feature of NEXUS is the ease with which a user can interact with and test a given model. A locator or mouse is used as an electrode, allowing the user to probe connectivity and stimulate particular units in the network. Figure 5 shows a typical graphics display. In addition to stimulating units and interactively examining and modifying connectivity, the user can also incorporate several types of learning rules into the model. Backpropagation and a version of the Hebb rule (Hebb, 1949) are currently supported. For models which are involved with image processing, large and complicated stimulus patterns can be created and loaded into individual maps. Finally, a variety of updating schemes for unit activity are available, from random sampling to sequential evaluation.

Conclusion

NEXUS, and the simulation methodology it incorporates, is designed for simulating large neural networks or systems which require several levels of structural organization. Several large-scale models are currently being simulated with NEXUS, including models of depth-from-occlusion (Finkel and Sajda, 1992), color perception (Courtney, Buchsbaum, and Finkel, 1992), and texture discrimination (Sakai, Sajda, and Finkel, 1992). Currently, NEXUS is designed to run exclusively on a conventional sequential computer. However, future versions will be designed to take advantage of parallel architectures, and work is being done to modify the NEXUS software for the Connection Machine™ (Hillis, 1985). Finally, we feel that the system's utility is not limited to neural modeling and may in fact be useful for simulating any number of large distributed systems having units of varying degrees of complexity and abstraction and requiring both a top-down and bottom-up approach.

References

[1] 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.

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

[3] N. Goddard, K.J. Lynne, and T. Mintz. Rochester connectionist simulator. Technical Report TR-233, Department of Computer Science, University of Rochester, Rochester, NY, 1987.

[4] S.J. Hanson. Meiosis networks. In D.S. Touretzky, editor, *Neural Information Processing Systems*, volume 1. Morgan Kaufmann Publishers, San Mateo, CA, 1990.

[5] D.O. Hebb. *The Organization of Behavior*. Wiley, New York, 1949.

[6] D.W. Hillis. *The Connection Machine*. MIT Press, Cambridge, MA, 1985.

[7] D. Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. W. H. Freeman, San Francisco, 1982.

[8] D.E. Rumelhart and J.L. McClelland. *Parallel Distributed Processing*. MIT Press, Cambridge, MA, 1986.

[9] P. Sajda and L.H. Finkel. "The NEXUS neural simulation environment." Internal Report, University of Pennsylvania, 1990.

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

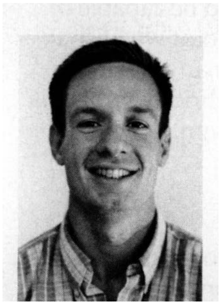
[11] K. Sakai, P. Sajda, and L.H. Finkel. Texture discrimination and binding by a modified energy model. In *International Joint Conference on Neural Networks*, pages 780-785, 1992.

[12] D.Wang and C.Hsu. SLONN: A simulation language for modeling of neural networks. *SIMULATION*, 55:69-83, 1990.

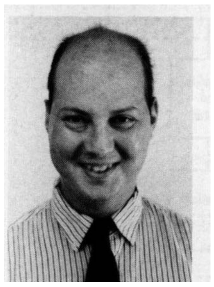
[13] M..A. Wilson, U.S. Bhalla, J.D. Uhley, and J.M. Bower. GENESIS: A system for simulating neural networks. In *Advances in Neural Network Information Processing Systems I*. Morgan Kaufmann Publishers, San Mateo, CA, 1989.

Acknowledgements

Development of the NEXUS simulator was supported by grants from The Whitaker Foundation and The Office of Naval Research (N00014-90-J-1864). P.S. is supported by an ONR-NDSEG Fellowship.



PAUL SAJDA received a B.S. degree in Electrical Engineering at the Massachusetts Institute of Technology. He has done research at the MIT Artificial Intelligence Laboratory, Whitaker College for Health Sciences, and the Whitehead Institute for Biomedical Research. He is currently completing a Ph.D. degree in Bioengineering at the University of Pennsylvania His present research interests include neural simulation, computational and cognitive neuroscience,visual perception and hardware implemenations of biologically-based vision systems.



LEIF H. FINKEL received a B.S. degree in Physics from the University of Maryland, and an M.D. and Ph.D. in Biophysics from the University of Pennsylvania. He is currently an assistant professor in the Department of Bioengineering and Associate of the Institute of Neurological Sciences at the University of Pennsylvania. Before joining the faculty at the University of Pennsylvania, Professor Finkel conducted research in computational neuroscience and neural modeling at the Rockefeller University. He has performed some of the largest neural network simulations to date. He is the author of scientific papers on such topics as synaptic plasticity, neural map formation and visual processing. His current interests are largely concerned with models of visual perception.