



NAPA : the Neural Activity Pattern Animator
by Kelli Michele Hodge

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in
Computer Science

Montana State University

© Copyright by Kelli Michele Hodge (1999)

Abstract:

NAPA, the Neural Activity Pattern Animator, is a scientific visualization application capable of animating spatio-temporal patterns of activity within an ensemble of neurons that comprise a neural map. NAPA takes an innovative approach to ensemble responses, by simultaneously animating the individual responses of each constituent neuron to produce the collective ensemble response. For the first time, biologists are able to view a quantitative prediction of a neural map's response to a given stimulus, -

**NAPA: THE NEURAL ACTIVITY PATTERN
ANIMATOR**

by

Kelli Michele Hodge

A thesis submitted in partial fulfillment

of the requirements for the degree

of

Master of Science

in

Computer Science

**MONTANA STATE UNIVERSITY-BOZEMAN
Bozeman, Montana**

April 1999

N378
H6633

APPROVAL

of a thesis submitted by

Kelli Michele Hodge

This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to the College of Graduate Studies.

J. Denbigh Starkey

J. Denbigh Starkey
Chairperson, Graduate Committee

4/8/99
Date

Approved for the Department of Computer Science

J. Denbigh Starkey

J. Denbigh Starkey
Head, Major Department

4/8/99
Date

Approved for the College of Graduate Studies

Bruce McLeod

Bruce S. McLeod
Graduate Dean

4-8-99
Date

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Montana State University-Bozeman, I agree that the Library shall make it available to borrowers under rules of the Library.

If I have indicated my intention to copyright this thesis by including a copyright notice page, copying is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for permission for extended quotation from or reproduction of this thesis in whole or in parts may be granted only by the copyright holder.

Signature

Devin M. Hoge

Date

4-7-99

TABLE OF CONTENTS

1 Introduction	1
1.1 Biological Background	1
1.2 Related Works	4
2 A New Approach to Ensemble Responses	7
2.1 Introduction	7
2.2 Weiner Kernel Analysis	8
2.3 Why Visualize?	9
2.4 Implementation Platform: Open Inventor	10
3 Design and Implementation	13
3.1 Requirements and Specifications	13
3.2 High-level Design	14
3.2.1 Density Clouds	15
3.2.2 Representing Dynamic Activity	16
3.3 Implementation	18
3.3.1 Engines and Sensors	18
3.3.2 Verification and Validation	19
3.3.3 Unsuccessful Engine Network	20
3.3.4 Successful Engine Network	22
4 Visualization	27
4.1 Static Functionality	27
4.2 Dynamic Functionality	30
4.3 Graphical User Interface Components	32
5 Biological Research Using NAPA	36
6 Conclusions	38
6.1 Future Improvements	38
6.2 Strengths of the NAPA Approach	40
References Cited	41

LIST OF FIGURES

1.1 Location of the cerci of the cricket	3
2.1 Scene graph using grouping nodes	12
3.1 Engine network driven by a SoTimeCounter	21
3.2 Engine network driven by a SoTimerSensor	24
4.1 Four afferents color-coded based on length	28
4.2 Four afferents color-coded based on peak tuning	28
4.3 Four afferents color-coded according to activity level in response to a wind stimulus from 290 degrees	29
4.4 Four afferents rendered as density clouds	30
4.5 Four frames generated from a 100HZ sine wave from 220 degrees	31
4.6 The Animation Controls window	33
4.7 The Response Files window	34

Abstract

NAPA, the Neural Activity Pattern Animator, is a scientific visualization application capable of animating spatio-temporal patterns of activity within an ensemble of neurons that comprise a neural map. NAPA takes an innovative approach to ensemble responses, by simultaneously animating the *individual* responses of each constituent neuron to produce the *collective* ensemble response. For the first time, biologists are able to view a quantitative prediction of a neural map's response to a given stimulus.

Chapter 1

Introduction

1.1 Biological Background

The role of computer graphics in the field of biological research is ever increasing. Contemporary scientists need means of viewing their data in more than two dimensions, as well as in the temporal domain. The bar graphs and pie charts of the past lack the power needed to describe the complex relationships within biological systems. Scientific visualization techniques enable this multidimensional data to be presented to the user in a conceptual, analytical, and comprehensible fashion.

A grand challenge in the field of neuroscience is to understand the biological basis of neural computation. This challenge must be approached from several fronts, some of which are purely technological in nature. Neuroscience investigators require tools that perform three primary activities: exploration of the relationships between nerve cells that lead to the emergent properties of neural *ensembles*, study of the correlation of nerve ensemble *activity* with *behavior*, and synthesis and testing of complex hypotheses on neural computation. The technologies to *observe and record* aspects of nervous system structure and function have increased dramatically in power over the last decade, with the advent of confocal microscopes, activity-dependent imaging, large-scale multiunit recording, and functional MRI. However, tools are still needed that enable

researchers to *synthesize* the knowledge stored within a data set into better understandings of the dynamic processes underlying neural computation. A major challenge to both the neuroscience and computer science communities lies in the development of visualization tools for displaying the results of data acquired through the aforementioned techniques. Powerful data analysis and simulation software packages must interface with these data acquisition techniques. This software must be capable of operating on diverse types of data, in order to enable the researcher to propose and test hypotheses related to neural computation.

NAPA addresses some of these critical needs. This software application provides both visualization and simulation of activity patterns in neural ensembles. Most importantly it serves as an interactive platform for testing hypotheses about the functional organization of neural ensembles.

Our studies focus on the cellular mechanisms of information processing in sensory systems. The model sensory system we have selected for study is the cricket cercal sensory system. This system mediates the detection and analysis of air currents in the cricket's immediate environment. The cercal system is sensitive to the *direction*, *velocity*, and *acceleration* of air currents. On the posterior region of the cricket are two appendages, called cerci, which are covered with filiform hairs of varying lengths. (See Figure 1.1) Each hair is innervated by *one* identifiable sensory neuron.[1] These sensory neurons, or *afferents*, become excited or inhibited by different types of wind stimuli. Also, each afferent has a preferred direction, and fires maximally when it detects a stimulus from that direction.

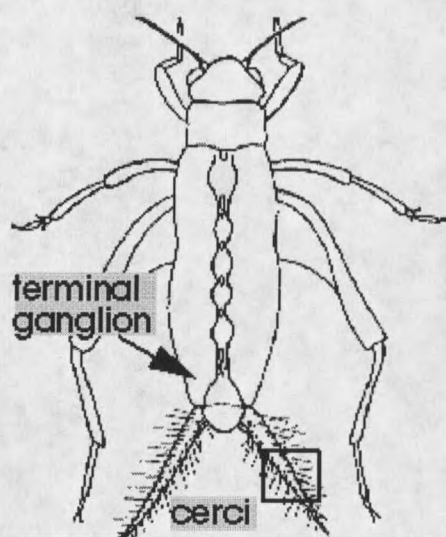


Figure 1.1: Location of the cerci of the cricket.

The cercal system is implemented around a neural map. A *neural map* is simply an ensemble of neurons. Across this map there is a systematic variation in the value of some parameter. Neural maps offer efficient means by which information is transmitted between different levels of the nervous system. The *anatomical arrangement* of the neurons within the ensemble determines to a large extent the *computational properties* of the ensemble.

Neurobiologists at the Center for Computational Biology (CCB) at Montana State University have a database of anatomical reconstructions of afferents from the cricket's cercal system.[2] To create a neural map of this system, the individual digitized afferents taken from different animals were scaled and aligned to a common coordinate system to create one collective map of the ensemble.[3] Scientists in CCB study the relationship between structural arrangement and computational function within this system by studying these neural maps.

Responses of the individual afferents to various wind stimuli have been recorded. The response of each afferent to a *wide variety* of stimuli can be extrapolated from these measured responses.[4] This thesis describes a method for visualizing the collective response of this neural map by combining the individual responses of the afferents. NAPA, the Neural Activity Pattern Animator, is a visualization application and researcher's tool for rendering a neural map, mapping static functional properties across the ensemble, and animating ensemble response patterns to a given stimulus.[5]

1.2 Related Work

Tools for neurobiologists to perform dynamic research typically fall into one of three categories: molecular modeling, optical recordings, or neural simulators. SCULPT [14] is an interactive tool for molecular modeling that is used to study the structure-function issue at the *molecular level*: users can experiment with how the 3D structure of a molecule affects or determines with which other molecules or chemicals it will bond.[14] The fact that SCULPT's functionality is coupled with anatomically correct data and allows the user to interact with the models is very similar to NAPA. SCULPT is used by molecular biologists in a similar manner that NAPA is used by neurobiologists; yet, NAPA also provides the user with animation capabilities.

Optical recording is a technique employed by neurobiologists to record the response of an ensemble of neurons to a stimulus. By using voltage-sensitive dyes, a researcher can measure the cumulative response of a neural system. This approach is limited in several respects. First of all, the only manipulative technique available to the

user is the ability to change the stimulus; he/she cannot choose which cells from which to record. Secondly, this method does not allow one to extrapolate the responses of the *individual* neurons; only the cumulative response is measured. NAPA takes a different approach to ensemble responses that will be described in Chapter 2.

Neural simulators, like GENESIS [15], provide neurobiologists with the capability to decompose a neuron's anatomical structure into *compartments*, then simulate the response of that neuron as a stimulus propagates from compartment to compartment. Like NAPA, the anatomical structure of a neuron is preserved when animating an *individual* neuron's response.[15] For an ensemble of neurons, however, the 3D structure of the constituent neurons must be significantly simplified. For example, to simulate the response of an ensemble of 26 cercal afferents in GENESIS, approximately 130,000 equations would have to be solved for every time-step: each of the 26 afferents has approximately 1000 compartments and 5 equations for each compartment. *Massive* computational power is required to perform such a simulation. Thus, each afferent's 3D structure must be significantly simplified, which is unacceptable in the study of how structural arrangement affects function.

NAPA's uniqueness lies in the fact that it enables neurobiologists to apply statistically correct predictions of neural responses to anatomically correct 3D data. Plus, NAPA's interactive design allows the user to manipulate the data and/or stimulus, then view the results of such changes on the resulting neural map's ensemble response. For example, the user can interactively change which neurons compose a certain neural map and change the stimulus applied to the map, then immediately view the resulting

animation output that those changes produce. No other system provides this type of functionality.

Chapter 2

A New Approach to Ensemble Responses

2.1 Introduction

NAPA takes an innovative approach to animating the ensemble response properties of a neural map to a given stimulus. Before this approach was implemented, researchers were restricted to making a single optical recording of the entire ensemble. Consequently, they were unable to discern the behavior of each *individual* afferent, since they were recording the *collective* response of the entire ensemble. This technique did not meet the needs of biologists who study the structure-function issue: how does the anatomical arrangement of neurons within a neural map contribute to, or determine, the computations performed by that map?[3] These scientists require the capability to decompose the *collective ensemble response* into its *constituent individual responses* in order to study how spatial arrangement influences function. However, optical recording techniques do not produce quantitative data on the response patterns of the *individual* neurons within the map.

NAPA approaches the animation of ensemble responses from a *constructive* standpoint. This means that instead of working with the neural map in its entirety, NAPA begins with the individual afferents. The responses of the individual neurons are then combined to yield the ensemble response.[5] This approach produces a *quantitative system response* that is not possible from optical recordings.[5] NAPA synthesizes the

individual recorded responses from the afferents into a collective response for the system. This enables biologists to perform quantitative experiments for *the first time* on the neural map as a whole.[5]

2.2 Wiener Kernel Analysis

Biologists in CCB have recorded the responses of individual afferents to a variety of stimuli. However, it is impossible to conduct experiments on each neuron with every possible wind stimulus. Thus, a generic mathematical modeling approach was developed so that a wide variety of simulations could be performed.[4] Techniques other than physiological experiments were needed in order to characterize and predict the types of responses elicited from the various afferents to different stimuli.

A Wiener kernel analysis was implemented to predict the response of an afferent to a wide range of air current stimuli. First, physiological experiments were performed where white noise air currents were presented to afferent preparations and the firing patterns elicited by these currents were recorded.[4] Then, a set of kernels was derived which allows the *prediction* of each afferent's response to an arbitrary stimulus.[4]

What is a kernel? Given two related functions, in this case, the stimulus presented to the cricket and the response elicited from that stimulus, a *mapping* can be derived of the stimulus function to the response function. This is accomplished by convolving the response function with some "other function." For a *polynomial expansion approximation* of the mapping, the "other function" is called the *kernel*. If

the mapping from the stimulus function to the response function is believed to be linear, one derives a first-order kernel. This first-order kernel is used to produce a linear estimate of the response. If the mapping is believed to be quadratic, one must compute the second-order kernel, which provides a quadratic approximation of the response. The *sum* of the first and second-order kernels is used to generate predictions of the response elicited by an arbitrary stimulus. The Wiener kernel analysis produces statistically sound approximations of neural responses for this system.[4] These predictions are in the form of arrays, where the value in each cell is proportional to the probability that the neuron fired at that time step.

2.3 Why Visualize?

Scientific visualization techniques are vital to the efforts at CCB because of the nature of the *data* under study and the nature of the *process* under study: both are inherently three-dimensional. The anatomical data is in the form of binary branch tree (BBT) representations of neurons reconstructed in 3D; there is no better way to view the anatomical structure of a neuron than a three dimensional rendering. Plus, many of the biologists in CCB focus their studies on the structure-function question. As described in Section 2.1, they are interested in how the spatial arrangement of neurons comprising a neural map influences or controls the computational functions of the neural map. This problem is actually four dimensional by nature, since the 3D organization of the ensemble reacts temporally to a stimulus. For this reason, an

animation in three dimensions was selected as the method of choice for analyzing the data produced from the Weiner kernel-driven simulations.

2.4 Implementation Platform: Open Inventor

NAPA is implemented on Silicon Graphics workstations in the object-oriented application programming interface, Open Inventor (OI), which uses OpenGL for rendering. Open Inventor consists of a library of objects and methods, which are used to create a "scene database." [6] This database consists of nodes that describe the geometry, lighting, transformation, material, and behavioral properties of a 3D scene. [6] The database also contains nodes for grouping objects together that have similar properties, animating parts of the scene, and selecting paths to and from nodes within the database. Since knowledge of the mechanisms by which a scene graph is organized and rendered is a prerequisite to designing an application in OI, a brief description of scene graph principles follows:

The database is stored internally as a directed acyclic graph. [6] The first node in the graph is the "root" of the scene; all nodes to be displayed must be added as "descendents" of the root. To render the scene, the graph is traversed from top to bottom, left to right. As each node is encountered during a traversal, it has a certain pre-defined behavior. [6] For example, a material node with a diffuse color value in the RGB color scheme of (0.0, 0.0, 1.0), will change the current color to blue. Thus, any geometry nodes that are children of the material node will be colored blue. Also,

anytime there is a change in any node within the scene graph, a re-rendering is automatically triggered by a traversal of the graph.[6]

A category of nodes, grouping nodes, enables one to apply or isolate certain material changes and transformations within chosen areas of the scene graph.[6] Grouping nodes contain an ordered list of child nodes. There are two primary grouping nodes. First, the SoGroup node simply provides a container for a collection of nodes. As the scene graph is traversed, the traversal state is accumulated within the group and then passed on to the parents of the SoGroup and the subsequent nodes within the graph.[7] In contrast, the SoSeparator node performs a "push" of the current traversal state *before* traversing the nodes within the SoSeparator group.[7] The transformations that occur within that group are applied *only within the group*. Following the traversal of the group, the traversal state is "popped," thereby restoring the previous traversal state. Thus, the transformations made within the SoSeparator group do not affect the parent of the group or the subsequent nodes within the scene graph. Grouping nodes are indispensable when constructing a scene graph because they provide efficient means by which to add structure and isolate transformations.[6] They *automatically* take care of the "pushing" and "popping" that OpenGL programmers must perform *explicitly*. [8]

As an example of a scene graph using SoGroup and SoSeparator nodes, consider Figure 2.1. This is a schematic diagram of a scene graph for a simple flower with two petals. As this scene graph is traversed from top to bottom and left to right, the "Stem Separator" contains a translation and a color change that only affect the

geometry for the stem. The original traversal state is then restored after processing the nodes within the "Stem Separator." Thus, the nodes within the "Petal Group" are unaffected by the transformation and color change.

The "color" child of the "Petal Group" sets the color for *both* "Petal 1" and "Petal 2." However, the translation for "Petal 1" does not affect the geometry for "Petal 2", since the state was pushed and popped upon traversing the "Petal 1 Separator."

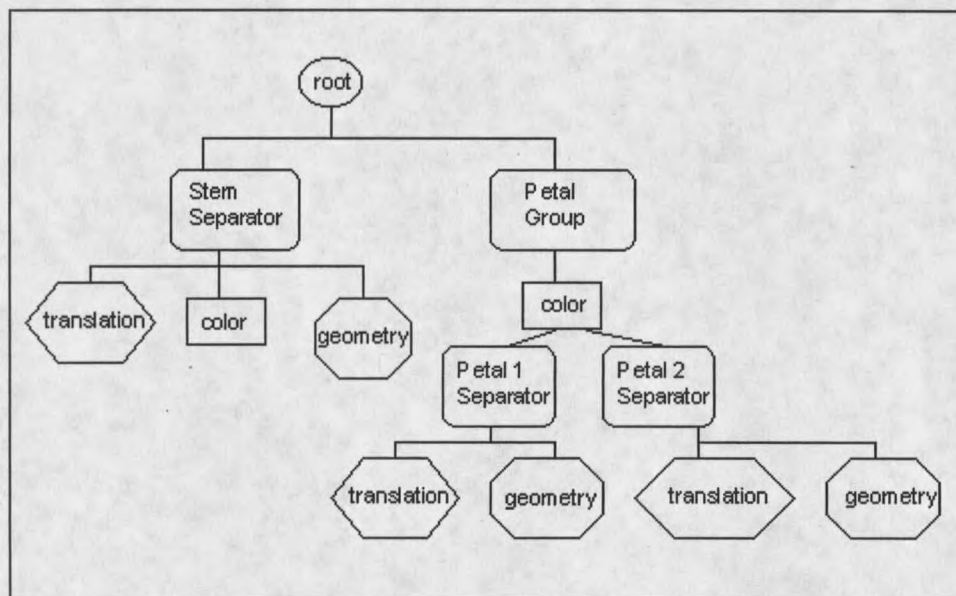


Figure 2.1: Scene graph using grouping nodes.

Chapter 3

Design and Implementation

Software engineering principles mandate that a programmer with a new software assignment does not begin writing code before he or she has a thorough understanding of the functionality the software should provide.[9] One must become familiar with the domain in which the software will be utilized, the constraints under which it should function, and the precise specifications that the various levels of users will require.[10]

3.1 Requirements and Specifications

The initial goal of this project was to augment visualization software written using Open Inventor with animation capabilities. This software parsed binary branch tree (BBT) representations of the 3D structure of digitized neurons and rendered them in an OI Examiner Viewer window. The requirement definition was clear: the augmentation to the software should animate the predicted response of an ensemble of neurons to a chosen stimulus. The exact functional specifications were not so clear.

A simple set of specifications was used during the initial design and implementation phase:

- The user will select neurons of interest.
- The user will choose the stimulus direction and stimulus waveform.
- The predicted responses for long, medium, and short hair-types will be modified at run-time based on each neuron's preferred direction.
- Temporal changes in some parameter of the rendered neuron will encode the probability of firing at that time-step.
- The user will view a "movie" of the predicted response.

This set of specifications turned out to be insufficient. After working with the prototype, the biologists in CCB determined that a more quantitative animation approach was necessary. We concluded that the animation must guarantee that each and every frame be shown, plus, researchers must be able to correlate each frame rendered with the exact location within the response file. As a result, the animation module of the system needed minor redesigning. This will be discussed in Sections 3.3.3 and 3.3.4.

3.2 High-level Design

The three *primary* issues that had to be resolved during the design phase were as follows: how to add a "density cloud" rendering representation, how to visually represent temporal changes in activity, and how to drive the animation. These will be discussed in Sections 3.2.1, 3.2.2 and 3.3.1, respectively. Another design consideration that had to be addressed was how to integrate the animation capabilities

into the framework of the existing code. Before I could modify and augment the existing software, I familiarized myself with the structure and organization of the code. I added numerous attributes and methods to the existing classes, and developed an animation module to provide the animation functionality. Overall, the integration with the existing code was straightforward and unambiguous.

3.2.1 Density Clouds

A new representation of the neurons needed to be added that was not currently supported in the software. The components of an afferent responsible for transmitting information to other neurons are called *synapses*; synapses are located in sites called *varicosities*. [11] In the existing software, the varicosities were rendered as spherical objects. This representation did not serve the needs of the biologists in CCB.

A statistical representation of the varicosities, called a "density cloud" was needed. This representation "smears" each varicosity into a cloud of points. The smearing is accomplished by convolving each sphere with a 3D Gaussian distribution. [3] The algorithm for computing density clouds was developed by mathematicians and neurophysiologists who previously worked in CCB. Unfortunately, the algorithm was implemented in a non-modular, non-object-oriented, and undocumented manner in C and GL.

One of the specifications for the system was the capability to animate the afferents using a density cloud representation. Therefore, I incorporated the density cloud functionality into NAPA. Rather than rewriting the entire implementation of

this algorithm, which would have been a larger undertaking that time permitted, I modified the existing program to generate "density files". For each neuron, a density file was created, which contained the local density for each non-empty voxel within the map. These density values were easily translated into point locations for rendering the afferents as density clouds.

3.2.2 Representing Dynamic Activity

When choosing a method for visually representing the dynamic activity patterns of the afferents, two implementation decisions had to be made. The first decision concerned the question of whether the human visual system would be able to perform the summation necessary to integrate the individual responses into an ensemble response. In other words, should a voxelization method be used? This method would divide 3D space into smaller sub-units, or voxels, and calculate a *cumulative* dynamic response for *each voxel* based on the contribution of each nearby afferent. A computationally inexpensive alternative would be to animate the response of each afferent *individually*, and allow the human eye to perform the visual summation to produce the ensemble response. Since utilizing the voxelization approach would be an additional transformation of the data, I chose an initial approach of allowing the user's eye to perform the summation.

The second design decision involved choosing a method to represent the dynamic activity patterns. In the realm of computer graphics, there are a finite number of ways to represent temporal information, including color value, motion,

transparency, and color intensity.[12] The choice of the appropriate method for this application was determined by several factors: which of these choices were not encoding *other* information, preservation of the anatomical positions of the ensemble's constituent afferents, and the size of the object to be animated. These factors will be discussed in the text that follows.

NAPA utilizes color to distinguish amongst the different categories of neurons. There are several different color-coding masks that can be applied to the neurons. For example, using a 360-degree color wheel, each afferent can be color-coded with its peak tuning angle. Since color was needed to encode the different categories of neurons, a yellow-to-blue color gradient representing an excitatory-to-inhibitory range, would not be a viable option for encoding dynamic activity.

Also, the use of motion to encode temporal activity was not possible. When an afferent is rendered, the BBT file of the digitized neuron is parsed point-by-point. Thus, the relationships between the locations of each point are maintained. It is imperative that the rendered proportions be consistent with the anatomy of the actual neuron. For this reason, slightly moving the density clouds at varying rates to represent each afferent's activity was not an option.

Modifying the transparency of an object is an effective technique for larger objects. For example, a one-inch diameter sphere appears to fade in and out of view by changing its transparency component. Unfortunately, transparency is not useful for encoding changes in a cloud of points.

Variations in the color intensity of the density clouds proved to be the method of choice. The biologists wanted to maintain the color-coding of the afferents based on hair type: the long, medium, and short hairs were each rendered in a different color. By dynamically changing the intensity parameter of the color, the brightness of the clouds increased as the probability of firing increased. This method effectively encoded the activity pattern, plus allowed the afferents to maintain their color-coding while being animated.

3.3 Implementation

After outlining the requirements and specifications, and creating the high-level design, I made more detailed implementation decisions. The primary design issue that remained was a method to drive the animation. Double buffering techniques[8] used in conjunction with timer system calls was a method I considered. However, Open Inventor encapsulates animation capabilities into its sensor and engine nodes, thereby making the implementation of animations much easier for the programmer. The following section describes these two types of nodes, and the remainder of this chapter discusses two networks implemented to drive the animation.

3.3.1 Sensors and Engines

Open Inventor provides two types of objects that enable software implementors to animate parts of a scene graph: sensors and engines. A network of these objects drives the animation capabilities in NAPA. Sensors are objects that call a user-

defined callback function after a certain type of event occurs.[6] This event can be a change in the data stored within a specific node in the scene graph, or when a certain amount of time expires.[7] An engine is an object that is used to animate or constrain parts of the scene graph, but can also perform other general functions. There are engines that perform arithmetic operations, create vectors, and select elements from an array, as well as engines that encapsulate animation capabilities.[6]

Engines and sensors can be connected to create a network.[6] This is accomplished by connecting fields in nodes together, or to the input or output of an engine. Also, one can connect the inputs and outputs of engines together, thereby creating a network that propagates information. As a simple example of an engine network, consider a SoTimeCounter connected to the position of a sphere. The SoTimeCounter can be initialized to repeatedly loop from 0 to 100 outputting values in steps of one. The "output" field from this engine is connected to the y-component of the sphere's translation. The result is that the sphere bounces up and down.

3.3.2 Verification and Validation

The verification and validation process involves ensuring that a piece of software meets its defined specifications and performs the tasks required by its users.[10] Following testing and use by different user-groups, I determined that the animation module met its initial specifications. *However*, that initial set of specifications did not fully meet the needs of the biologists. During the design phase, we determined that the output desired from the animation was a "movie". However, upon testing of the

animation functionality, it became clear that a more quantitative output was necessary. With this in mind, the following section discusses the initial animation module and how it failed to meet the needs of the users. Then, Section 3.3.4 describes the more quantitative approach taken with the redesigned animation module.

3.3.3 Unsuccessful Engine Network

The validation phase brought to our attention that one functionality requirement we failed to think of during the design phase was a guarantee that *each and every frame* of the animation would be displayed. The biologists realized they needed the capability of correlating each frame that was rendered with the corresponding location in the response file. The first engine network I implemented utilized an engine to drive the animation that was often dependent upon CPU load. This characteristic was not mentioned in the documentation and only became apparent in the testing phase. This meant that an "every frame guarantee" was not possible with the original engine network design.

Before describing the specific engines used to drive the animation module, I will first give a general description of the design of the engine network. In the most high-level terms, the animation must encode the probability values from a "response array" into color intensities for the density clouds. To implement this, one must cycle through the array, select each element, and connect that value to the intensity of the

density clouds.[5] Thus, the higher the value in a given cell of the array, the brighter the clouds will appear in the corresponding frame of the animation.

The animation module was initially driven by a SoTimeCounter engine. (See Figure 3.1) The SoTimeCounter cycled from 0 to ARRAYSIZE in steps of 1. The value output from the counter was used as the index into the response array via the SoSelectOne engine. The SoSelectOne engine simply selects one value from an array.[7] The value output from this engine was the probability of firing at that time step; it was connected as the input into a SoComposeVector engine. Since each array element is a scalar, but color is stored as a vector, this engine was needed to construct a vector from the current array element. Finally, this engine's output was connected to the diffuse, ambient, specular, and emissive color of the density clouds of the afferent.

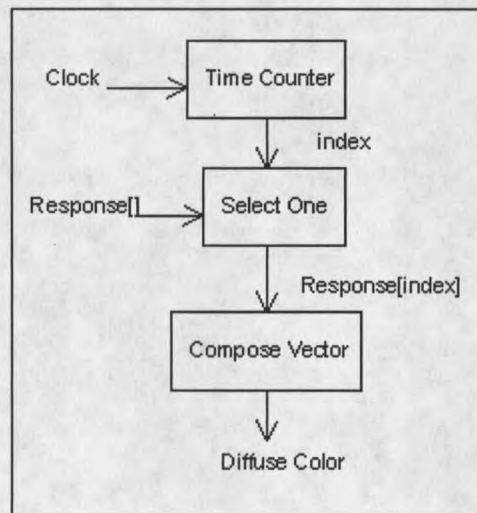


Figure 3.1: Engine network driven by a SoTimeCounter.

Testing of the implementation of this approach produced *different* animation outputs for the *same* stimulus and afferents chosen: clearly something was not performing correctly. After further testing and posting an inquiry to the OI newsgroup, it became evident that the SoTimeCounter engine is not, by design, *guaranteed* to output each and every value; the actual performance of this engine at a given instant depends on the load on the CPU. This fact clearly explained why the animation outputs were different for each iteration. This performance characteristic was clearly unacceptable: no frames could be skipped. An alternative design of the engine network would have to be developed.

3.3.4 Successful Engine Network

In developing a new design for the animation network, I augmented the initial set of specifications with the following additions:

- Every frame of the animation must be displayed.
- The user must be able to correlate the location in the response array with the corresponding frame.
- The engine network must not be dependent upon CPU load.

Utilization of a *sensor* to drive the animation, rather than an engine, met this set of specifications and produced the quantitative outputs required. Recall that a sensor is an object that calls a user-defined callback function whenever a certain event occurs.[6]

A SoTimerSensor is a sensor that calls its callback function at regular intervals.[7] Recall that a counter is used to output the current index into the response array. By updating the value of the counter *inside the callback function*, each and every frame is guaranteed to be displayed. Since the timer is guaranteed to expire, and its callback is guaranteed to be triggered, incrementing the counter inside of the callback guarantees that *every* array index will be generated. Thus, each frame corresponding to each array element, responseArray[index], will be displayed.

One SoTimerSensor and one SoCounter are needed to synchronize the animation for all neurons. Thus, I added them as members of the class "Viewer", which coordinates the viewing and manipulations of all neurons. The SoCounter is an engine that outputs values from 0 to ARRAYSIZE; each time its field "trigger" is touched, it outputs the next value.[7] The SoTimerSensor's callback is where this "touching" takes place. Thus, each time the timer expires, the callback is called. This function calls the method "trigger.touch()", which causes the counter to output the next value. (See Figure 3.2) The OI code to create the timer, counter and callback is below.

```
SoCounter * animationCounter = new SoCounter;
SoTimerSensor * alarm = new SoTimerSensor;
//the CB to call each time the timer expires:
alarm->setFunction(timerSensorCB);
//the data to pass to the CB:
alarm->setData(animationCounter);
```

```
void Viewer::timerSensorCB(void * data)
{
    SoCounter * counter = (SoCounter *) data;
    counter->trigger.touch();
}
```

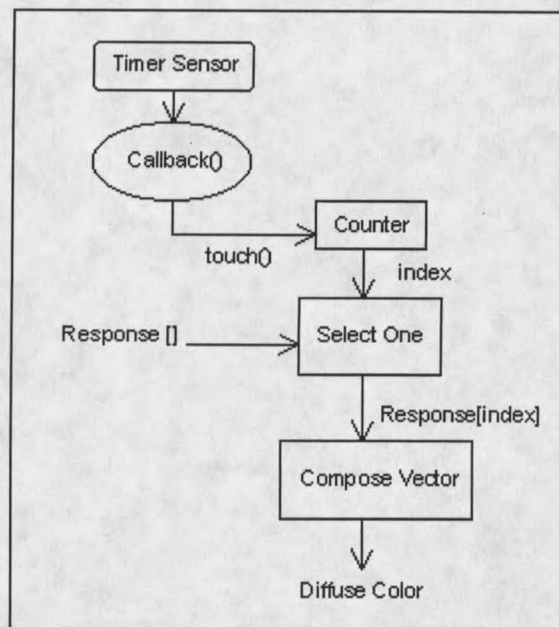


Figure 3.2: Engine network driven by a SoTimerSensor.

A description of the remainder of the engine network with OI code follows. The output of the SoCounter is attached to the "index" of the SoSelectOne. The "input" into the SoSelectOne is the response array. (See Figure 3.2)

```

SoMFFloat * responseArray;
SoSelectOne * selection = new SoSelectOne;
selection->input->connectFrom(responseArray);
selection->index->connectFrom(&myViewer->
                             animationCounter->output);
  
```

Depending on the type of afferent, either the X, Y, or Z component of a SoComposeVec3f is connected to the output of the SoSelectOne. Thus, depending on whether the afferent is attached to a long, medium, or short hair, the red, green, or blue component of its color will be modified. This is accomplished by attaching the

output of the SoComposeVec3f to the material node for the density clouds, called "dynamicMaterial". (See Figure 3.2)

```
SoComposeVec3f * rgbVector = new SoComposeVec3f;
SoMaterial * dynamicMaterial;

hairLength = myNeuron->getHairLength();
switch (hairLength)
{
    case ivNeuron::Long:          // x = red
        rgbVector->x.enableConnection(TRUE);
        rgbVector->x.connectFrom(selection->output);
        break;
    case ivNeuron::Medium:        // y = green
        rgbVector->y.enableConnection(TRUE);
        rgbVector->y.connectFrom(selection->output);
        break;

    case ivNeuron::Short:         // z = blue
        rgbVector->z.enableConnection(TRUE);
        rgbVector->z.connectFrom(selection->output);
        break;
}

dynamicMaterial->diffuseColor.connectFrom
    (&rgbVector->vector);
dynamicMaterial->ambientColor.connectFrom
    (&rgbVector->vector);
dynamicMaterial->specularColor.connectFrom
    (&rgbVector->vector);
dynamicMaterial->emissiveColor.connectFrom
    (&rgbVector->vector);
```

The output from this approach to the engine network produced the desired quantitative results and met the modified set of specifications. By utilizing the expiration of a timer to trigger the connection of the next array element to the color of the density clouds, the engine network became independent of CPU load. Thus, the "every frame guarantee" was met. Also, functionality to enable the user to correlate

his/her position in the response file with the corresponding frame was included. This will be described in Section 4.2.

Chapter 4

Visualization

NAPA's visualization capabilities include rendering digitized neurons, displaying static attributes across a neural map's 3D structure, and animating predicted firing patterns for a given stimulus.[5] Numerous graphical user interface components enable the user to select and manipulate the different rendering options.

4.1 Static Functionality

NAPA creates a working model of a neural map, which enables static spatial relationships to be visualized and analyzed. A biologist can choose a *functional mask* to apply to an ensemble, which enables him/her to visualize and analyze the 3D spatial relationships of different functional properties of neurons.[5] The length of the hair a neuron innervates and its peak tuning angle are two such functional properties. The response that a certain stimulus *direction* would elicit, given the neuron's peak tuning can also be visualized. Each functional property is represented by a color mapped onto the 3D structure of the afferent. The color for each afferent is determined by the application of a particular functional mask to the ensemble.[5]

Below is a rendering of an ensemble composed of four afferents. (Figure 4.1)

This map is color-coded with the functional mask of "hair length". The afferents that

innervate long hairs are colored red, and the afferents that innervate medium hairs are colored green.



Figure 4.1: Four afferents color-coded based on length.

Below is the same ensemble color-coded with each afferent's peak tuning angle, based on the 360 degree color wheel. (Figure 4.2)



Figure 4.2: Four afferents color-coded based on peak tuning.

For the image below, a wind direction stimulus of 290 degrees was chosen. (Figure 4.3) Then, a yellow to blue color gradient was used to color each neuron: yellow represents that a neuron is excited by that direction, and blue represents that the neuron is inhibited. As seen in Figure 4.2, the peak tuning for the afferent on the far-left side of the image is 297 degrees. So, a stimulus from 290 degrees will certainly excite the neuron. Thus, the neuron is rendered yellow in this color-coding scheme.

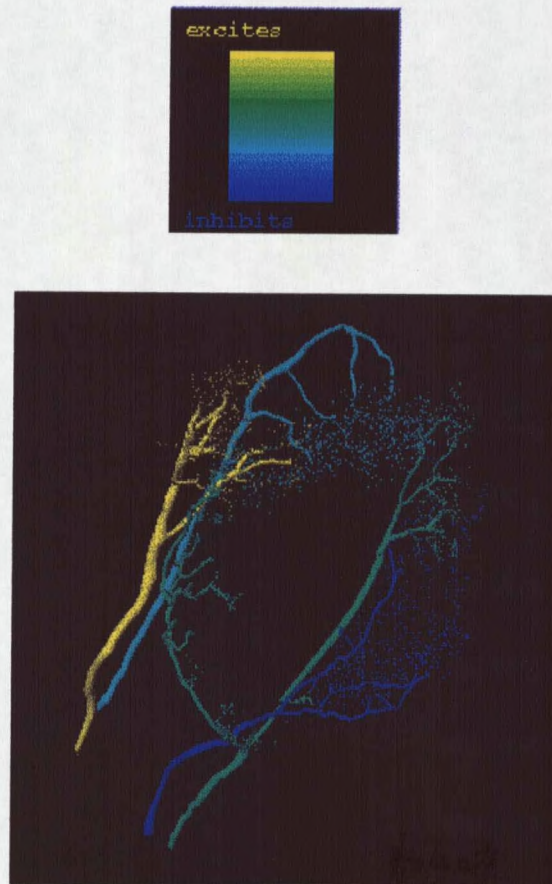


Figure 4.3: Four afferents color-coded according to activity level in response to a wind stimulus from 290 degrees.

Finally, the same neural map is rendered in a density cloud representation, as described in Section 3.2.1, in the next image. (Figure 4.4) The afferents are color-coded based on their peak tuning angles.



Figure 4.4: Four afferents rendered as density clouds.

4.2 Dynamic Functionality

NAPA's working model of the neural map enables dynamic attributes to be animated and simulated.[5] A user selects a direction, velocity, and waveform of an air current stimulus to apply to the ensemble. Then, the predicted response of each neuron type is retrieved. Next, for each afferent, either the long, medium, or short predicted response is scaled. This scaling consists of multiplying each array element by the difference between the cosine of the stimulus angle and the cosine of each afferent's peak tuning. Thus, an afferent whose peak tuning is *close* to the stimulus direction

will have larger values in its array, than an afferent whose peak tuning is far from the stimulus direction. The results of this calculation are arrays containing the predicted response for each individual neuron. Finally, the collective ensemble response is visualized by temporal changes in the color intensities within the map, as the values in each individual array are encoded by the color intensity for each afferent.[5]

What follows is a series of still images taken from a run of the animation. (Figure 4.5) The wind stimulus was a 100 HZ sine wave from 220 degrees.

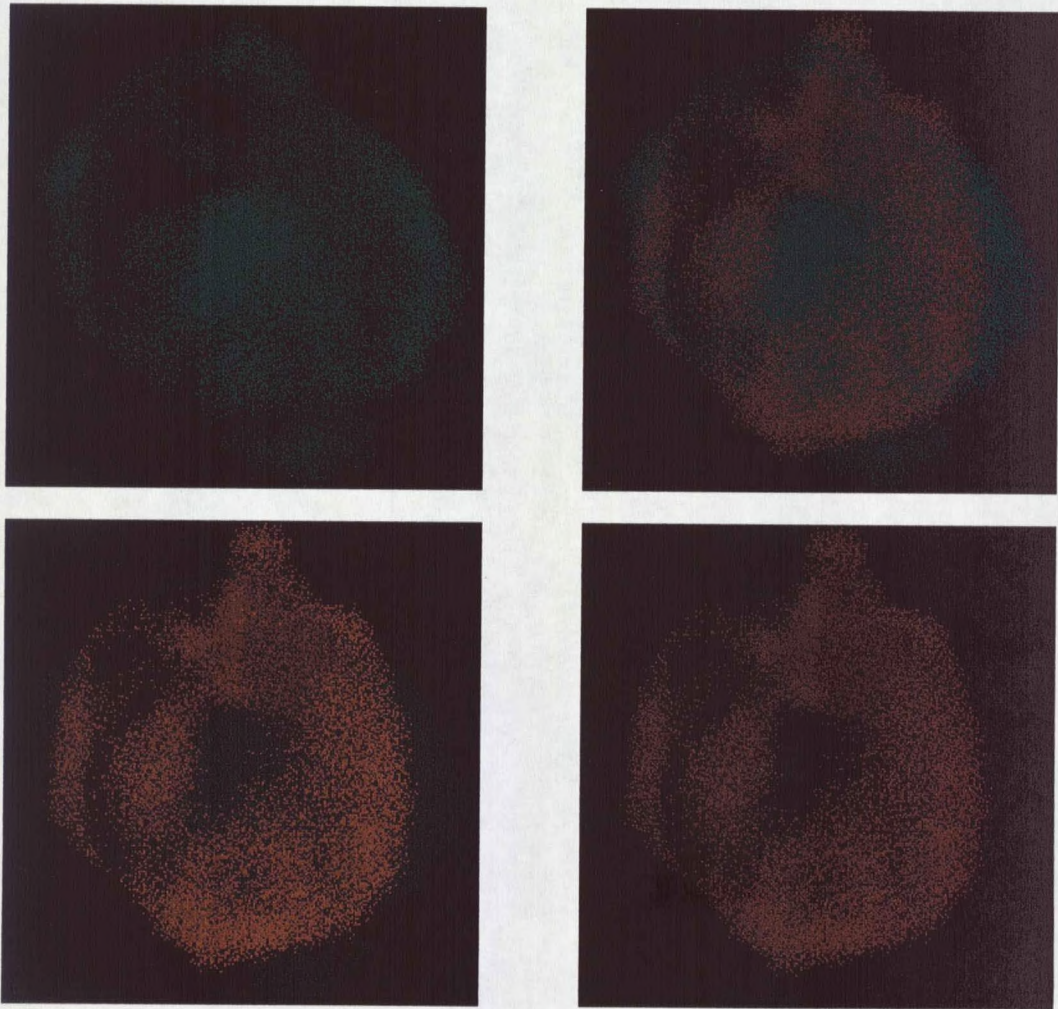


Figure 4.5: Four frames generated from a 100HZ sine wave from 220 degrees.

The neural map in Figure 4.5 is composed of 13 "long" afferents colored red, and 13 "medium" afferents colored green. The afferents are rendered as density clouds. Each image is a different time-step as the animation progresses.

4.3 Graphical User Interface Components

Graphical user interface (GUI) components enable biologists to manipulate several parameters in the engine network, and correlate the current location in the response file with the corresponding frame. These capabilities allow users to set up the specific animation they desire, step through the animation one time-step at a time, or play the resulting animation automatically.

The user manipulates the animation using the Animation Controls window. (See Figure 4.6) First, the user must select a direction for the wind stimulus. The slider labeled "Stimulus" in Figure 4.6 allows the user to easily choose the direction. Then, the response files must be selected. By clicking the "Browse" button, a file dialog box appears containing the response files from which to choose. The user must pick a file for each type of neuron rendered, long, medium, and short, using the drop down button in the bottom left corner of Figure 4.6. Lastly, the user should render the neurons as density clouds, by clicking the third button on the left-hand-side of the Animation Controls window. The initial animation parameters are now set; by pressing the Play/Pause button, the engine network described in Section 3.3.2 is assembled and the animation of the predicted response of the ensemble begins.

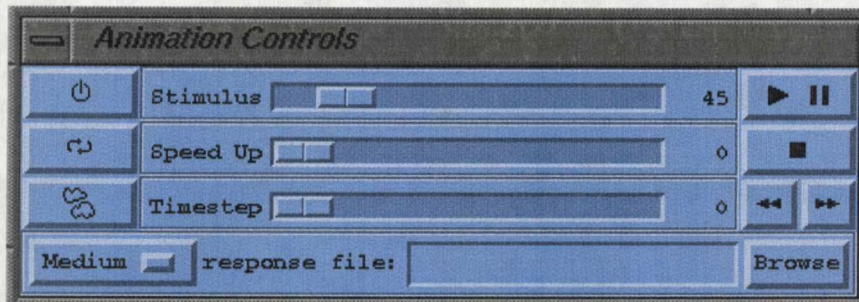


Figure 4.6: The Animation Controls window.

Another window appears as the animation begins. (See Figure 4.7) The Response Files window contains a graphical representation of the response files selected for the animation. The vertical white bar in this window moves in synchronization with the animation of the predicted response, since its translation is attached to the SoCounter engine's attribute "index". Thus, as the SoCounter outputs values from 0 to ARRAYSIZE, the horizontal translation of the bar is automatically updated with the same value. This functionality enables the user to coordinate the precise location within the response file with the corresponding frame. This visual feedback enables biologists to use the animation in a more quantitative fashion: by allowing the user view to each response waveform simultaneously with the predicted animation, the ensemble response is much easier to analyze and quantify.

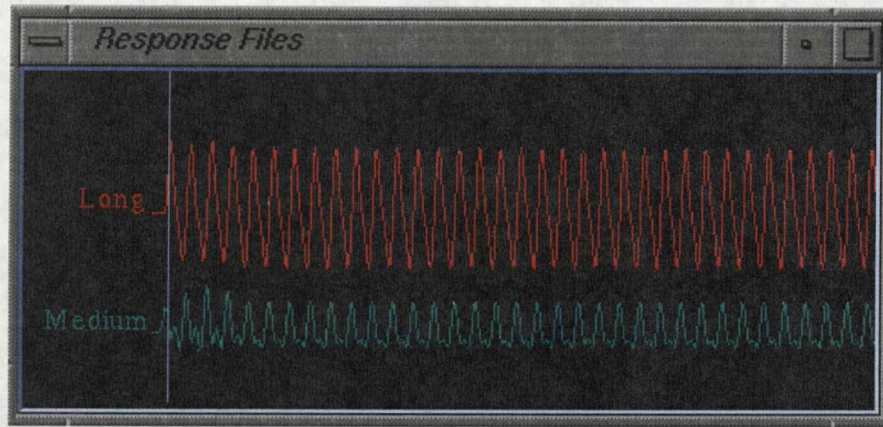


Figure 4.7: The Response Files window.

The user can also manipulate several animation parameters using the GUI components in Figure 4.6. For example, the animation is paused and restarted using the Play/Pause button, and stopped using the Stop button. This is accomplished by scheduling or unscheduling the SoTimerSensor. Recall from Section 3.3.2 that the SoTimerSensor drives the animation for all neurons. Also, the speed of the animation is controlled with the "Speed Up" slider: as the user moves the slider farther to the right, the *interval* attribute of the SoTimerSensor is decreased, thereby speeding up its rate of firing. Finally, the "TimeStep" slider controls the current location within the response files. By moving this slider, the "index" attribute of the SoCounter engine is changed to the value selected. Thus, both the displayed frame and the location of the vertical bar in the Response Files window are updated with that value. This slider can also be used to step frame-by-frame through the animation. Each time the user clicks on the slider, the "index" of the SoCounter is incremented or decremented by one,

depending on whether the user clicked on the right or left side of the slider's bar. The SoTimerSensor is also paused. This functionality allows the user to study areas within the animated response of particular interest more carefully, since the user is stepping from frame to frame at his/her own pace.

Chapter 5

Biological Research Using NAPA

NAPA's functionality has led to new insights that demonstrate the validity and experimental power of the 3D graphical map as a tool for analyzing sensory processing in neural systems. This new ability to project interactively static and dynamic functional masks onto the 3D map has enabled biologists to visualize the temporal relationships between different categories of neurons, which was impossible before.[5] For example, a direct outcome of using NAPA was a quantitative prediction that the afferents innervating long hairs and the afferents innervating medium hairs respond out-of-phase with each other at higher frequencies.

Current and future research in CCB has a new direction because of NAPA's visualization capabilities. Insights into the nature of the encoding performed by the system are now possible, since researchers are able to visualize the application of time-series data to neural ensembles. As stated before, this is the first time that researchers have the capability to view a quantitative ensemble response to a stimulus based on the predicted individual responses of the ensemble's constituent neurons.[5] This capability enables researchers to formulate hypotheses on how the ensemble *encodes*, and thereby, *differentiates* between different stimuli. Such insight is crucial to an understanding of the mechanisms underlying sophisticated neural computations.

Members of CCB are currently using this method to formulate a mathematical model of the filtering properties of *interneurons* as they process information from many sensory afferents. Interneurons are the next level in the cricket's nervous system; they receive inputs from many afferents and integrate that input before transmitting information onto higher levels in the nervous system. The anatomical arrangement of the afferents with respect to a given interneuron is thought to be a deterministic factor in the functionality of the interneuron.[3] We plan to extend the visualization capabilities of NAPA to include a "graphical intersection" of an interneuron's 3D structure with the afferent map. We will represent static attributes by mapping the numerous color-coding masks applied to the ensemble onto the interneuron's branching structure. Also, dynamic responses will be encoded by temporal changes in the color intensity or transparency along the interneuron's 3D structure. This functionality will allow biologists to formulate a quantitative model of the transfer of information between different levels of the nervous system.

Chapter 6

Conclusions

6.1 Future Improvements

In its current state, NAPA suffers from several limitations, which we plan to address. First of all, the application of both a static and dynamic functional mask *simultaneously* is not supported; instead, the dynamic mask overrides the static mask. This means that the user cannot view the predicted response of the ensemble when the ensemble is color-coded based on peak tuning direction, or any other static functional mask.[5] This characteristic is a result of the fact that the animation is designed to function with the map color-coded based on hair length. In its current state, each neuron is colored red, green, or blue, depending on whether the neuron innervates a long, medium, or short hair; its color intensity is then dynamically scaled by the engine network. The OI code needs to be modified so that the intensity of *any color* in which a given neuron is rendered can be scaled by the engine network. This will provide an additional way to study the data by enabling the user to view predicted responses while the map is color-coded with a static functional mask.

A second limitation that we are addressing is the fact that low activity levels appear very dim when animated. Since there are no additional mathematical transformations applied to the probability of firing at each time-step, if every

value in a given response array is within the range of 0.0 to 1.5, the intensity of the density clouds will also be within that range. Thus, the clouds will barely be visible when animated. This is a problem that involves both visualization and mathematical issues: the user would like to be able to *see* predicted responses even for low activity levels, yet the mathematics dictates that the data should not be altered simply for visualization purposes without careful consideration of the implications of such a change. We must be sure that if we make dimmer animations appear brighter that we are not skewing all of our research results. Therefore, further mathematical modeling is required before a decision on this issue can be made.

We plan to extend NAPA's functionality to provide two additional capabilities. First, we will graph the *stimulus* files in the same manner that we currently graph the *response* files. Once mathematicians in CCB convert the current stimulus files into the format of a function or set of points, this functionality will be trivial to implement since we currently have the capability to graph either of these data formats.

The second functionality enhancement we plan for NAPA, as described in Chapter 5, is the ability to map static and dynamic ensemble characteristics onto the anatomy of an interneuron. Recall that the spatial regions where the afferent ensemble overlaps with the interneuron are the areas where sensory information is transmitted. The application of a static functional mask will result in the rendering of the interneuron's 3D structure where each segment is color-coded based on the area of the afferent map with which it overlaps. In a similar fashion, temporal changes in the

diffuse color intensity or transparency along the interneuron's 3D structure will encode the cumulative dynamic response of the regions of overlap. This functionality will enable biologists to formulate a mathematical model of the filtering performed by the interneurons as they process information from hundreds of sensory afferents.

6.2 Strengths of the NAPA Approach

The application of dynamic properties to a 3D neural map is a generic approach that holds much promise for the entire neuroscience community. The dynamic techniques described here hold promise in two major areas. First, they allow a visual and quantitative analysis of how temporal computations on sensory input are performed within a neural system. As stated previously, without NAPA's functionality researchers are restricted to interpreting optical recordings that measure the ensemble's response as a *whole*. NAPA enables the *individual* responses to be combined to predict the global response, thereby generating a more quantitatively correct ensemble response. This is a general approach that is applicable to all neural systems, provided the anatomical data can be obtained.[5] Secondly, by comparing NAPA's predictive activity patterns in real time with optical recordings coming from live animals given the same stimulus, we have developed a generic technique for analyzing complex spatio-temporal patterns of activity within a neural map.[5] Thus, the "dynamic map" approach will further research in how stimulus processing is controlled by the anatomical and functional properties of the constituent neurons.

REFERENCES CITED

- [1] Jacobs, G.A. "Detection and analysis of air currents by crickets: A special insect sense", *Bioscience*, 45, 1995, 776-785.
- [2] Stiber, M., Jacobs, G.A., Swanberg, D., "LOGOS: A computational framework for neuroinformatics research", *Proceedings of the Ninth International Conference on Scientific and Statistical Database Management*, Olympia, Washington, USA, 1997.
- [3] Jacobs, G.A. and Theunissen, F.E., "Functional organization of the neural map in the cricket cercal sensory system", *J. Neuroscience*, 16, 1996, 769-784.
- [4] Roddey J.C. and Jacobs, G.A., "Information theoretic analysis of dynamical encoding by filiform mechanoreceptors in the cricket cercal system", *J. Neurophysiol.*, 75, 1996, 1365-1476.
- [5] Hodge, K.M., Jacobs, G.A., Starkey, J.D., "NAPA, The Neural Activity Pattern Animator", *Proceedings of the Computer Graphics and Imaging Conference*, Halifax, Nova Scotia, 1997.
- [6] Wernecke, J., *The Inventor Mentor*, Addison-Wesley, 1994.
- [7] Wernecke, J., *Open Inventor C++ Reference Manual*, Addison-Wesley, 1994.
- [8] Neider, J., Davis, T., Woo, M., *OpenGL Programming Guide*, Release 1, Addison-Wesley, 1993.
- [9] Pressman, R.S., *Software Engineering, A Practitioner's Approach*, Third Edition, McGraw-Hill, Inc., 1992.
- [10] Sommerville, I., *Software Engineering*, Addison-Wesley, 1996.
- [11] Jacobs, G.A. and Nevin, R., "Anatomical relationships between sensory afferent arborizations in the cricket cercal system", *Anatomical Record*, 231, 1996, 563-572.
- [12] Foley, J.D., van Dam, A., Feiner, S.K., Hughes, J.F., Phillips, R.L., *Introduction to Computer Graphics*, Addison-Wesley, 1994.
- [14] SCULPT, Interactive Simulations, Inc., <http://www.intsim.com>.
- [15] Bower, J.M., Beeman, D., *The Book of GENESIS*, Second Edition, TELOS: The Electric Library of Science, 1998.

MONTANA STATE UNIVERSITY - BOZEMAN



3 1762 10421613 8