# GROWING BRAINS *IN SILICO*: INTEGRATING BIOCHEMISTRY, GENETICS AND NEURAL ACTIVITY IN NEURODEVELOPMENTAL SIMULATIONS

by

Rasmus Storjohann

M.Sc., Biochemistry, Simon Fraser University, 1999

B.Sc., Chemistry, Heriot-Watt University, 1995

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

## APPROVAL

| | |
|---|---|
| **Name:** | Rasmus Storjohann |
| **Degree:** | Master of Science |
| **Title of thesis:** | Growing brains *in silico*: Integrating biochemistry, genetics and neural activity in neurodevelopmental simulations |

**Examining Committee:** Dr. Jian Pei
Chair

______________________________

Dr. Robert F. Hadley
Computing Science, Simon Fraser University

______________________________

Dr. Gary F. Marcus
Psychology, New York University

______________________________

Dr. Oliver Schulte
Computing Science, Simon Fraser University

______________________________

Dr. Eirikur Palsson (external examiner)
Biological Sciences, Simon Fraser University

**Date Approved:** Dec 14, 04

# Abstract

Biologists' understanding of the roles of genetics, biochemistry and activity in neural function is rapidly improving. All three interact in complex ways during development, recovery from injury and in learning and memory. The software system NeuroGene was written to simulate neurodevelopmental processes. Simulated neurons develop within a 3D environment. Protein diffusion, decay and receptor-ligand binding are simulated. Simulations are controlled by genetic information encoded using a novel programming language mimicking the control mechanisms of biological genes. Simulated genes may be regulated by protein concentrations, neural activity and cellular morphology. Genes control protein production, changes in cell morphology and neural properties, including learning. We successfully simulate the formation of topographic projection from the retina to the tectum. We propose a novel model of topography based on simulated growth cones. We also simulate activity-dependent refinement, through which diffuse connections are modified until each retinal cell connects to only a few target cells.

*To Cathy and Kristoffer*

# Acknowledgements

I want to thank Gary Marcus for directing the research presented in this thesis. Working with Gary has been a privilege. While clear in his opinions, he is always open to being convinced by argument. He also taught me to always think twice before ruling out possible avenues of research. His encouragement when times were rough was greatly appreciated.

Bob Hadley has been fully involved in the writing of the thesis itself, and his incisive questions have forced me to think more deeply about the nature of this work and how it relates to the biological systems that inspire it. I want to thank Bob for his diligence and attention to detail in commenting on drafts of this thesis. His questions made me consider more closely my assumptions. I also thank Bob for his extensive financial support during my time as his student.

I also thank Eric DeWitt for many discussions regarding the concepts and design of the NeuroGene system. His contributions to the NeuroGene design are described in more detail in section 3.2 of this thesis.

I want to thank friends and fellow students of S.F.U., Cesar Dragunsky, Jim Stuart, Laila Singh and David Arpin, as well as my brother Kristoffer for many fun discussions.

Finally, I thank Cathy for her love, understanding and support throughout these years. I could not have done this without you.

# Contents

# List of Tables

# List of Figures

# List of Algorithms

# Chapter 1

# Introduction

The human brain is arguably the most complex thing in the known universe. How does it come about? What is the role of innateness and learning in its formation? Is this level of complexity beyond the limits of what can be constructed under genetic control? How do genes and environment (nature and nurture) compete or cooperate to build a brain? In this thesis I will try to show how genetic and environmental information together may shape some components of the vertebrate central nervous system. My tool will be a simulator based on current knowledge of genetics and developmental neurobiology. There are three components to my work:

- Develop a set of primitive operations that can encode the genetic information necessary to explain the observed developmental processes. These primitives must reflect the genetic control mechanisms as well as cellular communication and behaviour.

- Build a simulator of neurodevelopmental processes. This simulator takes as input a genome written using the set of primitives. By application of the genetic instructions to a set of cells, these undergo changes in behaviour, and through this behaviour they give rise to structures that may resemble biological tissues.

- Demonstrate the power of the simulator by simulating the development of well understood systems from neurobiology. The systems are chosen for being dependent on both genetic and environmental cues during development.

In the remainder of this chapter, I will first discuss why I believe there are large potential benefits to incorporating as much biological knowledge as possible in connectionist models of neural systems (section 1.1). Section 1.2 is an introduction of the relevant principles of developmental biology, relating to gene regulation, chemical signaling, etc. Section 1.3 is an introduction of the relevant neurobiology, mostly relating to developmental processes and mechanisms which are unique to neural tissues. In section 1.4 I discuss the relationship between learning an innateness, and how an improved understanding of how these two interact may lead to a better understanding of the mind. Finally, in section 1.5 I outline earlier work, some of which forms the basis for the research presented in this thesis.

## 1.1 The need for a biological approach to connectionism

The goal of cognitive science, in my understanding, is to bridge the gap between physiology and psychology — to explain our experience of the mind in terms of the mechanisms of the body and the brain. The problem is approached from two directions, with psychology or biology as starting point. Cognitive science brings together a number of disciplines that use either or both of these approaches. Connectionism, perhaps more than any other cognitive science, attacks the problem from both sides, creating models consisting of neuron-like nodes and synapse-like connections to tackle complex cognitive tasks.

Regardless of whether we chose a "classical" or a neural network approach to artificial intelligence, an ultimate goal is to understand how the mind is implemented in the biological brain. The neural inspiration for connectionism makes this goal more explicit, and perhaps, more attainable. Some connectionist researchers seem to feel that "in time, 'rigid' classical models of higher cognition will yield to more fluid, more realistic, neurally inspired models of which are at least akin to current connectionist models" (Hadley, 1999, p. 198).

Connectionism has suffered from a paucity of detailed knowledge about the properties of biological neural networks. As a result, many important connectionist models have relied on assumptions and guesswork about the entities they model. Recent and rapid progress in experimental techniques within neuroscience, genetics and other areas of biology is now filling many gaps in our knowledge about the structure and function of biological neural networks. The conductance (or "weight") of individual synapses can be measured and manipulated *in vivo* (Zhang et al., 1998) and related directly to behaviour (Oda et al., 1998). The role of chemical communication in the activity-dependent formation and stabilization of synapses is being uncovered (Bonhoeffer, 1996). Particular forms of learning have been disabled by reducing the activity of a particular receptor in parts of the brain of monkeys (Liu et al., 2004), and social memory and recall has been specifically altered by changing the level of particular neural signaling molecules in rats, voles and sheep (Bielsky and Young, 2004). Functional neural networks can be grown and studied *in vitro* (Bi and Poo, 1999). Transient shifts in human perception have been induced in experiments designed based on our improved understanding of synapse plasticity (Fu et al., 2002). New learning algorithms with strong biological justifications have been formulated (Song et al., 2000; Elliott and Shadbolt, 1998a), and new connectionist architectures are constructed based on detailed information about patterns of neural interconnections (Rodriguez et al., 2004).

This rich and rapidly growing body of knowledge represents a treasure-trove for connectionists. However, the incorporation of detailed biological experimental results into a connectionist methodology poses new challenges, not least because the biological understanding of neural systems is still far from complete, and new discoveries continue to surprise. My research is part of an effort to allow new biological knowledge to bear on central unanswered questions in cognitive science: What are the topologies of the neural networks that are responsible for our cognitive processes? How are these topologies formed? What is the role of learning and experience in forming neural networks? To what extent is our mind subject to genetic (i.e. evolutionary) control? What are the biological processes underlying learning? What are the limits to what can be learned?

## 1.2 Principles of developmental biology

Development is the process by which biological tissues grow and differentiate during the life of an organism[1]. Starting with the fertilized egg, developmental processes control cell division, differentiation, migration and programmed cell death to give rise to an organism equipped to survive outside the protective environment of the egg or the womb. After birth, development continues in the form of gradual increase in body size, as well as more abrupt changes such as metamorphosis in insects and puberty in mammals.

[1]See appendix A for a list of terms than may be unfamiliar to non-biologists.

In this review of developmental biology I will focus on developmental mechanisms in animals. While the genetic machinery is identical in plants and animals, plant development is significantly different in several respects, particularly due to the fact that plant cells do not migrate. Within the animal kingdom, the principles of developmental biology appear to be almost universal. For example, many genes and mechanisms discovered in the worm *C.elegans* have since been found to also exist in mammals.

### 1.2.1 Gene expression and regulation

The development of an organism is a process of cellular self-organization under the control of the genetic code. Genes are active units of information, chemically encoded in the form of DNA, and interacting with other genes and with the cellular environment through DNA-protein and protein-protein interactions. The collection of all the genes of a single organism is called the *genome* of that organism.

At one level of description, a gene contains the information needed to make one particular kind of protein. Through the machinery of transcription and translation (often abbreviated to *expression*), information encoded in the 4-letter DNA code (A, T, G and C) is translated into a sequence of amino acids which are linked together to form a protein. The protein code consists of twenty different amino acids. In contrast to the four letters of the DNA code, the twenty amino acids have widely divergent chemical properties, which in turn cause different protein sequences to give rise to proteins with widely divergent, and highly specialized, properties and functions. Examples include structural proteins (e.g. in hair and skin), catalytic enzymes (e.g. digestive enzymes), photo-reactive proteins (chlorophyll and eye photo receptor pigments) and communication carriers (hormones like insulin). Proteins are central to the processes of cellular motion (e.g. in muscle cells), chemical recognition (antibodies), signal transduction (sensory cells related to all our five senses), as well as every other function performed by living bodies.

A particular class of proteins are those that bind to DNA, in particular proteins that are able to recognize and bind to particular DNA code sequences (Davidson, 2001). These are *regulatory proteins* (or *transcription factors*), which means that they regulate the transcription of genes. This gives rise to the second level of description of genes. At this level, genes are units of information that interact with each other. When the gene for regulatory protein

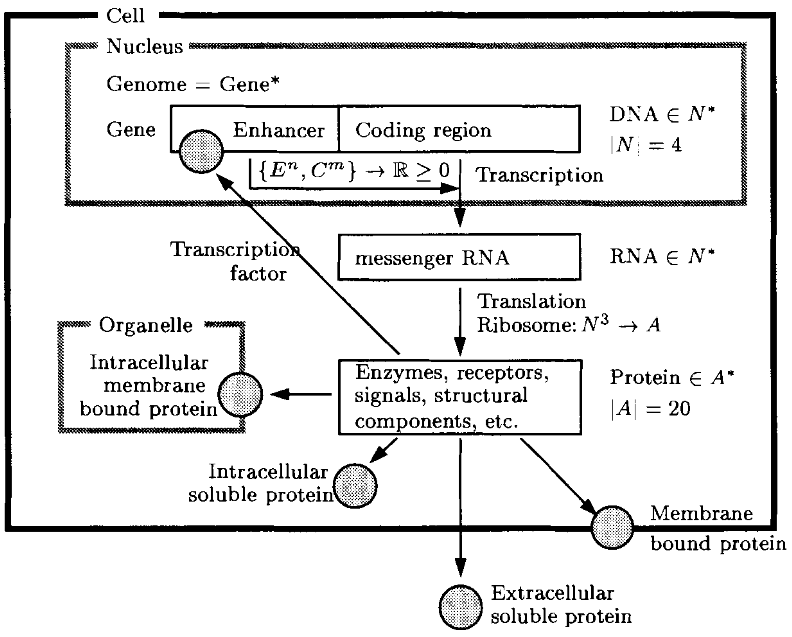


Figure 1.1: Gene expression. The genome is stored in the cell nucleus, of which there is exactly one in every cell. The genome consists of a large number of genes, encoded in DNA. A gene consists of an enhancer region and a coding region. The enhancer consists of regulatory elements. The regulatory elements bind regulatory proteins. The combined states of all the regulatory elements (occupied or free) determines the rate at which transcription occurs. During transcription copies of the gene's coding region are created. These copies are made from RNA, a chemical that is very similar to DNA, and which encodes information over the same 4-letter alphabet. The mRNA copies are mobile and may be transported to the far reaches of the cell. There they are translated into proteins by cellular machinery called *ribosomes*. Proteins are strings over a 20 letter alphabet of amino acids. Depending on the sequence of amino acids in the protein, the protein may bind to cell membranes or remain soluble in the intracellular fluid. The proteins are then transported to their final destination. For Regulatory proteins re-enter the nucleus where they bind to regulatory elements of one or more genes.

"A" is transcribed and translated into a protein, this protein will seek out its specific DNA code sequence and bind to it. This DNA sequence actually forms part of another gene "B" — however, the sequence is never translated into a protein. The sequence is one of perhaps many *regulatory elements* that make up the *enhancer* region of gene "B". Depending on the particular gene in question, the presence of protein "A" at the regulatory element may cause gene "B" to become expressed, or to cease to be expressed. This comes about because the protein may help or hinder the binding of an enzyme to the beginning of gene "B"'s protein-coding region. This enzyme carries out the first step (*transcription*) in the process of gene expression. The expression of "B" will in this case be causally affected by changes in the concentration of the protein "A". Such causal relationships are central to the developmental and genetic processes we aim to simulate. However, the molecular detail underlying these relations (as shown in figure 1.2) will not be explicitly modeled.

There exists a lot of biological data on the expression behaviour of a wide range of genes, to the point where the elucidation of such information is considered as its own discipline, known as *proteomics*, see e.g. Palzkill (2002) and Davidson et al. (2002). For some genes, their expression behaviour can be expressed using boolean algebra, e.g., a gene is expressed only if some protein is present in a certain amount, or only if the cell receives neural activation above a certain threshold, etc. For other genes, the expression profile is better encoded using arithmetic, e.g., the expression rate of some gene is proportional to the square root of the concentration of some protein. For yet other genes, the expression behaviour is best represented by a combination of both.

Genes also play a central role in evolution, the process which over the centuries and generations causes organisms to constantly remain genetically well-adapted to their changing environment. Evolution rests on genetic processes such as point-mutation, chromosome cross-over, sexual reproduction and others. Since we do not aim to model evolution within NeuroGene, these processes fall outside the scope of this thesis.

### 1.2.2 Gene networks and cascades

Many genes are subject to very complex regulatory mechanisms. An example of such a gene is *endo16* in sea urchin, which

> ... is expressed in the early embryo in the progenitors of the endomesoderm,

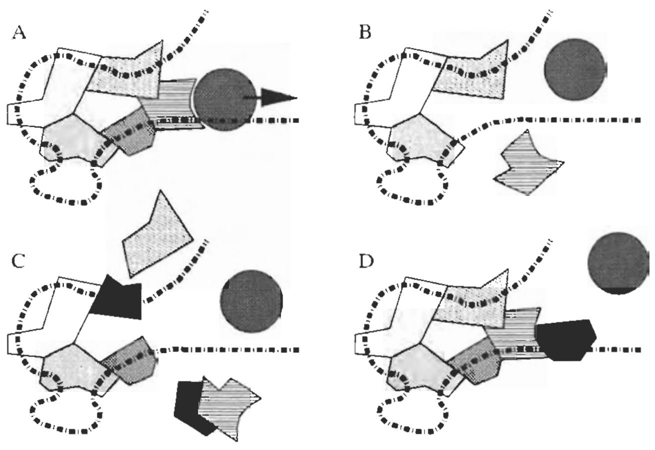


Figure 1.2: Enhancer complex. A: A number of regulatory proteins (polygons) bind to the DNA of the enhancer region (dashed line) in such a way that the transcriptional enzyme (circle) can bind and initiate transcription. The coding region of the gene is off to the right in the direction of the arrow. In the complex proteins bind to DNA as well as to each other through specific interactions, i.e. proteins recognize and bind to specific DNA sequences and specific binding sites on other proteins. The probability that the enhancer complex is formed in a given cell at a particular time is a function of the properties of these DNA-protein and protein-protein interactions, as well as the concentration of all the regulatory proteins. For example, a protein that binds only weakly to the enhancer complex must be present in a high concentration for the complex to be likely to form, while a protein that binds strongly needs only be present in low concentration to ensure that the complex forms. The higher the probability of the complex forming, the more frequently the transcriptional enzyme will bind and copy the gene's coding region, and the higher the expression level of the gene. The expression rate of the gene is therefore a function of the properties of the enhancer and the regulatory proteins (both ultimately encoded in the genome as inherited information) and the current state of the cell, expressed as the current concentration of all relevant regulatory proteins. B: Transcription is disrupted because a required regulatory protein is absent. C: Transcription is disrupted because the presence of an inhibiting regulatory proteins (black), either on bound or free proteins, impede the formation of the enhancer complex. D: Transcription is disrupted because a regulatory protein (black) binds where the transcription enzyme should bind.

> then throughout the gut, and finally only in the midgut, a not very elaborate temporal sequence. But its control system turns out to be an elegantly organized and complex information processing device that responds to both positive and negative inputs to set the boundaries of expression. Early and late expression phases are controlled by two different sub-regions of the regulatory sequence, or modules, each several hundred base pairs long. Together these are serviced by nine different DNA sequence-specific transcription factors.[2] [...] The functions that the *endo16* regulatory system performs are conditional on the inputs, and they include linear amplification of these inputs, but also many non-linear operations such as an intermodule switch that transfers control from the early to the late module, detection of input thresholds, and various logic operations (Davidson et al., 2002, p. 1670).

Depending on the abundance of different regulatory proteins in the cell, some of the regulatory elements will be occupied, and some will be empty. Depending on which proteins are present and which are absent, the enzyme that initiates gene expression may have a higher or lower affinity for the gene's coding region; stated simply, the enzyme may have a harder or easier time doing its job. If the enzyme has an easy time, it will succeed often. As a result, the protein that the gene encodes will be produced in large quantities. If the enzyme has a hard time getting to the gene, the protein is produced at a lower rate or not at all.

The abundance of a regulatory protein affects the states (occupied or free) of regulatory elements of other genes. This affects the facility with which the enzyme can help express these genes, which in turn affects the rate of production of the corresponding proteins. In other words, the abundance of one protein affects the rate of production of another — a more common way of saying the same thing is that one gene affects the expression of another gene.

The genes of an organism define a network in the form of a directed graph. The nodes of the graph are the genes, the arcs are causal their interactions which determine the expression behaviours of each gene. While no such network has ever been constructed in its entirety for any organism, pieces of it has been discerned in many cases.

Sometimes the graph takes the form of a tree, with a "master gene" at the root. Such gene networks are often called cascades. An example is the gene *Pax6*, which is involved

[2] *Transcription factor* is another term for *regulatory protein.*

in eye formation in fruit flies. Through experimental means, Halder et al. (1995) caused Pax6 to be expressed in parts of the fly where it would normally be silent. This caused anatomically complete eyes to form at places where limbs and antennas normally form. The Pax6 gene sequence consists of about 21,000 DNA base pairs, representing approximately 42 kbits of information (GeneCards, 2004). This gene by itself cannot possibly contain all the information necessary to build fruit fly eyes. Instead, Pax6 regulates a number of other genes, which themselves regulate more genes, until all the genes required to build photo receptor cells, pigment cells, cone cells and bristles are activated in the proper spatial and temporal pattern required to build the eye. It is believed that a total of approximately 2,500 genes are involved in eye formation in the fruit fly (Halder et al., 1995, 1998; Gehring, 1998).

### 1.2.3 Receptors and inter-cellular signaling

The description above limits gene-gene interactions to those that occur within a single cell. However, developmental processes rely on communication *between* cells, both between neighbouring cells and over longer distances. The carriers of information are still chemical substances, either attached to the surface of cells or released into the extra-cellular fluid.

Chemically mediated inter-cellular communication relies on cells carrying receptors that are sensitive to the presence of specific signal molecules. The receptors are bound to the cell membrane, and carry a binding site for signal molecules. This binding site is exposed to the extra-cellular space. When a molecule binds to the receptor on the outside of the cell, it sets off a chain of events inside the cell leading to the activation of some protein that affects the expression level of one or more genes. In this way the expression level of genes in one cell may affect extra-cellular signal concentrations which again affect the expression levels of genes within other cells.

Competition as developmental principle in neuroscience has found extensive support in experimental observations (van Ooyen and Ribchester, 2003; Swindale, 1996). The most studies example of such competition is the neuro-muscular junction (Rasmussen and Willshaw, 1993). Ligand-receptor binding is a purely chemical process by which biological systems may implement competition (Harding and Chowdhry, 2001). Different cells, carrying differing amounts of receptor bound to their surfaces, will attract different amounts of a soluble ligand which is present in the space surrounding the cells. The amount of receptor

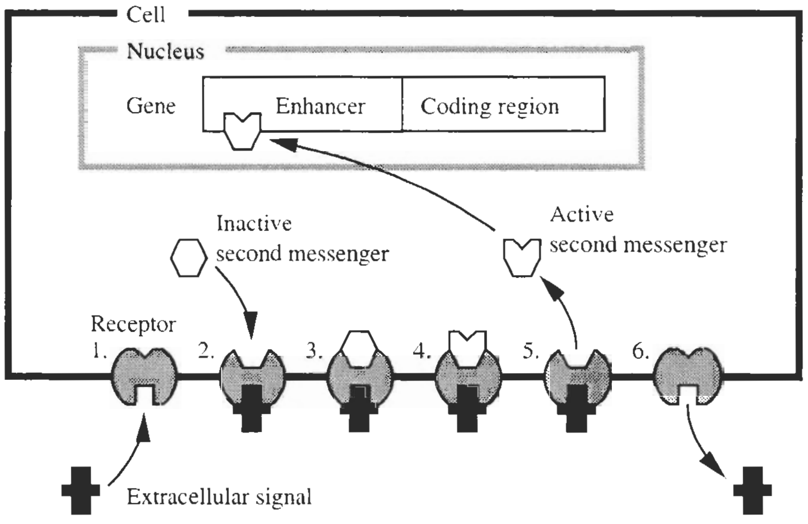


Figure 1.3: Intracellular signaling. The cell initially carries receptors on its surface, and contains a "second messenger" protein in an inactive form. When an extracellular signal binds to the receptor (1), the receptor changes shape, exposing on the intracellular side an "active site" (2) that was previously inaccessible. The inactive form of the second messenger molecule binds to this active site (3). The receptor then acts as an enzyme (or catalyst), converting the inactive form of the second messenger into its active form (4). This activation often consists of attaching a negatively charged phosphate group ($-PO_4^{2-}$) to the second messenger. Enzymes that activate other proteins through phosphorylation are called ***kinases***, and they are central to almost all known intracellular information processing systems. The activated second messenger then breaks free from the receptor, and travels to the nucleus, where it acts as a regulatory protein, binding to the enhancer region of genes, turning them on or off. The receptor is now free to activate a new second messenger molecule (5). A single extracellular signal molecule can thus cause the activation of a large number of second messenger molecules, thereby causing a form of *amplification*. Eventually the extracellular signaling molecule is destroyed or dislodged from the receptor, and the receptor returns to its inactive form (6).
Note that both the inactive second messenger and the receptor are proteins encoded in genes, and that these genes must have been expressed in preparation for the encounter with the extracellular signal molecule — the sensitivity of cells to extracellular signals is therefore under genetic control.

carried by each cell then becomes the competitive strength of each cell. The cells compete for a limited resource, which is the ligand present in the extracellular space.

### 1.2.4 Gradients, domains and patterns

The bodies of all complex animals are divided into tissues and organs which consist of cells with different properties and activities. The main goal of developmental biologists is to understand how these patterns of cell differentiation arise.

The simplest mechanism by which cell differentiation can arise is through a concentration gradient established though the diffusion of a soluble[3] signaling molecule. Typically, the signaling substance is produced by cells within a small area, and diffusion causes a monotonic decrease in the concentration of the signal with increasing distance from the source. The range of diffusion and the shape of the resulting gradient depend on the rates of production, diffusion and breakdown of the chemical signal. Given two or more dissimilar gradients, sensitive cells may react reliably in a position-dependent manner, forming new domains in what was previously uniform tissue. Some cell types may react by producing other diffusible chemicals, thus establishing new gradients. In this way, wide ranging gradients may give rise to new gradients with narrower range, each time dividing domains of uniform cells into differentiated sub domains of steadily increasing complexity (Davidson et al., 2002).

Not all signal molecules are free to diffuse through the embryo. Some remain bound to the surface of the cells that produce them. These signals are only detected by cells in the immediate neighbourhood. Membrane-bound signals may act as tags which identify cells or tissues, or for constructing fine patterns in which single cells may assume roles that differ from those of their immediate neighbours (Muller, 1997).

### 1.2.5 Developmental mechanisms

Genes interacting with other genes is a means to an end, which is to affect the size and shape and activities of the cells, organs and organisms they are part of. The size and shapes of cells is commonly referred to as the cell *morphology*. In contrast to the wide repertoire of gene-gene interactions within and among cells, genes have a limited set of ways in which they can affect cell morphology. These include cell division, cell migration, and programmed cell death, all

[3]"Soluble" is used in this text to distinguish from "membrane-bound".

of which are under genetic control. As will become apparent in the following section, nerve cells have more complex shapes than most other cells in the body, and a correspondingly wider repertoire of developmental mechanisms is used to build neural organs.

Biological cells contain an internal network of stiff fibres which maintain the cells' shape. This network is known as the *cytoskeleton*. The cytoskeleton is highly dynamic, constantly being broken down and rebuilt. It is attached to the cell membrane, and changes in the structure of the skeleton may lead to changes in the shape of the cell. Through the cells' ability to adhere to their surroundings, such changes may again be translated into cellular migration. Similar intracellular scaffolding is also involved in cell division. Concerted cytoskeletal changes in large numbers of cells may lead to large-scale changes in shapes of tissues and organs. Very similar mechanisms underlie the migration of axons and dendrites. Like all other processes in the cell, changes in the cytoskeleton are under genetic control, through genes expressing proteins which in turn cause skeletal fibres to break down or be rebuilt. See e.g. Browder et al. (1991, chapter 9) for an introduction to cellular adhesion and motility. The intracellular skeletal structures are also involved in intracellular trafficking of proteins to specific target areas within the cell. The targeting information is encoded in the gene along with the protein sequence. The target area may be one of the various intracellular structures or, in the case of neurons, different part of the neuronal structure, such as synapses, axons, dendrites, etc. Through this mechanism, the cell nucleus, containing the genome of the organism, acts as a central coordinator of all the cell's activities.

### 1.2.6 Studying genes

The study of genes involves a number of interdependent techniques. This typically imposes a certain ordering on the study of the various aspects of the structure and function of a particular gene, since the information originating from the application of one technique is required for the use of another. However, the fact that many genes are shared among organisms often allows for the required information discovered in one organism to be applied to the study of another. Below are listed a number of questions that can be posed relating to a given gene, and short outlines of the techniques used to answer these questions:

1. What does a gene do? That is, what is the effect of changes in the expression rate of the gene? Statistical analysis of the variability of a trait across individuals can suggest

the existence of one or more genes contributing to that trait. The discovery of mutant organisms with grossly affected phenotypes, such as altered body-shape, behaviour, etc., can also lead researchers to the discovery of the mutated gene which can then be directly correlated to the observed mutant phenotype.

2. How is the gene regulated? How is the gene expression rate affected by changes in the state of the cell and its environment? Where and when is the gene expressed? These question can not be answered without a certain understanding of question 1, since researchers need to be able to determine whether the gene is active or not in any given situation. Important techniques include knockout and over expression of other genes causing reduced or elevated concentrations of regulatory proteins, thus affecting the expression of the gene in question. Detecting either the mRNA or protein produced from a gene requires the sequence of the gene or that of a related gene, possibly from a different organism. This can be used to determine where in the organism a gene is active.

3. Where is the gene located within the chromosomes of the genome? This is often a difficult and time-consuming problem to solve, however it leads to the sequence of the gene, including regulatory elements, which is required for further progress. Again, it relies on knowing the answer to question 1, and may be assisted by knowing sequences of related genes from other species.

4. How is the gene related to other genes in the same and other species? This requires the sequence of the gene to be known, and involves statistical analyses of similar but non-identical sequences. If the gene is found to be closely related to other genes (perhaps in other organisms) whose structure and function is well understood, rapid progress can be made in understanding the novel gene.

5. How does the gene regulation work? This requires the identification of regulatory protein binding sites within the DNA sequence of the regulatory elements of the gene. Often the gene sequence alone can give many answers, but if a novel regulatory protein with an unknown DNA binding sequence is involved, more detailed investigation of molecular interactions between the DNA and the novel protein is required. A good understanding of question 2 is very helpful in this work.

6. How does the protein produced by the gene work? That is, what other protein does the protein in question interact with, how are these interactions modulated, etc. (see figure 1.3). Here knock-out and over expression of putative interacting proteins is an important tool, similar to what was used in question 2, but here applied to the discovery of protein-protein interactions. An extension of this question is: what is the detailed molecular mechanism of the proteins' action? This requires either the molecular structure of the protein, or that of a protein produced by a closely related gene.

7. How can drugs be created which would alleviate disorders caused by malfunctions in the protein? This is often the ultimate goal of genetic research. This requires a detailed understanding of question 6.

Each of these analyses produce an increasingly detailed picture of the structure and function of a particular gene. As will be outlined in detail in chapter 2, genes within the NeuroGene system are represented at a level of abstraction where much of this information is omitted. In fact, only the information discovered from points 1–2 above are represented within the NeuroGene data structures.

## 1.3 Neurobiology

A neuron is a cell highly specialized for processing information in the form of electrical pulses. The central part of the cell consists of the *soma*, which contains the cell nucleus. This is the site of gene transcription, translation can also occur in other parts of the neuron (see figure 1.1). Extending from the soma are long fibrous extensions collectively known as *neurites*, which fall into two classes, *dendrites* and *axons*. A cell commonly carries a large number of dendrites but only one axon. The axon usually extends far away from the soma while the dendrites form a highly branched network close to the soma.

Neurons form connections among themselves through synapses. At the synapse the electric signal (*action potential*) in the presynaptic cell is converted into a pulse of chemical signaling molecules (*neurotransmitter*) which diffuses across the synaptic cleft. At the postsynaptic side of the cleft, the chemical signal is converted back into an electric signal in the postsynaptic cell. As a rule, synapses connect the axon of the presynaptic cell to on of the

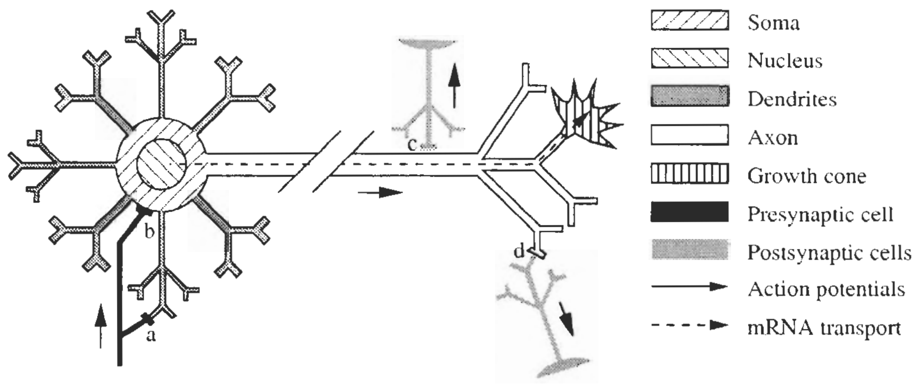


Figure 1.4: Schematic of a neuron. a–d indicates different kinds of synapses: a and b are incoming synapses through which the cell receives neural activity from other cells not shown. c and d are outgoing synapses through which neural activity is transmitted to other neurons. a and d are synapses formed between the axon of one cell and the dendrite arbor of another. b is an example of a synapse on the soma of a cell, while c is an *en-passant* synapse formed on the main shaft of the axon.

dendrites of the postsynaptic cell. This implies that dendrites act as the cell's input device, while the axon is the output device.

### 1.3.1 Neural activity

Neurons are non-linear in their information processing activity. They pass on information to their postsynaptic cells in the form of discrete action potentials. Information is carried in the frequency and timing of these action potentials, not in their shape or amplitude which tend to change little. This means that a neuron's computation consists of determining when to "fire", i.e., cause an action potential in the axon. It is believed that this computation takes place at the *axon hillock*, an enlarged part of the axon at the point where it joins the soma.

A neuron that has not received any action potentials for a while, will be at *resting potential*. This means that the potential (voltage) across the cell membrane will be around -70 mV. As the neuron receives action potentials from its afferent cells, the membrane potential increases. The amount of increase will depend on the number of action potentials, the conductance of synapses (also referred to as *synaptic weight*), the proximity of the synapses to the soma and properties of the cell membrane. When the membrane potential reaches a ***threshold potential***, typically between -65 and -55 mV, the cell fires. This means that a new action potential is generated in the axon hillock, after which the membrane potential drops down to the resting potential. The neuron is typically insensitive to further input for a period after firing, called the *refractory delay*. When a cell receives action potentials on inhibitory synapses, it may become hyper-polarized, bringing the membrane potential below the resting potential and making the cell less likely to fire.

### 1.3.2 Growth cones and neuronal development

A correctly interconnected nervous system is the result of massive exploratory growth, with the growth cone as the explorer *par excellence* (Gordon-Weeks, 2000). A dynamic structure at the tip of extending neurites, the growth cone is largely responsible for neuritic organization at all levels, from long-range axon guidance to the exact positioning of synapses. It consists of a bulbous structure at the tip of a growing neurite from which extends large numbers of fibers called *filopodia*. These fibers probe their environment and direct axon growth and synaptogenesis. While most of the research on growth cones has focused on axonal path finding mechanisms, a more recent study has shown that dendrite growth occurs though similar processes (Furrer et al., 2003). Given the central role of growth cones in the development of neural tissues, it follows that simulation of growth cone function will also be central role to the NeuroGene simulation system.

Growth cones contain mitocondria and ribosomes, indicating that they are self-sufficient with respect to energy and protein synthesis (*gene translation*). Growth cones even remain functional for up to three hours after the axon has been severed from the cell body (Halloran and Kalil, 1994). The cell presumably exerts control over the growth cones by transporting mRNA of appropriate genes along the extending neurites. However, the translational machinery in the growth cone (which produces protein according to instructions carried by mRNA) constitutes a second level of control over protein synthesis. The "local authority" of the growth cone may give it the ability to react rapidly to changes in its environment, something that would not be possible if it was "micro-managed" by the distant nucleus.

The structural integrity of axon fibres are maintained by intracellular skeletal fibres called microtubules (see section 1.2.5). These extend into the central portion of the growth cone. The peripheral part of the growth cone is high in another fibrous substance called actin. See

Hely (1998, chapter 3.) for a good review of growth cone biology.

The ability of growth cones to act as path finders relies on filopodia's ability to sense their environment. The filopodia are able to detect exceedingly shallow concentration gradients of signaling molecules (1% over 25 $\mu$m (Baier and Bonhoeffer, 1992; Goodhill, 1998)). They are also able to integrate multiple such signals simultaneously in order to decode combinatorial signaling (Thor et al., 1999; Goodhill, 2003). Axons are often observed to grow along physical features in their environment, such as grooves (Tai and Buettner, 1998). In nature, axons commonly grow in tight bundles, possibly due to growth cones navigating along similar physical features.

Both actin and microtubules are constantly broken down and rebuilt. These dynamic processes form the basis of growth cone motility (Buck and Zheng, 2002). For example, an increase in rate of buildup and a concurrent decrease in the rate of breakdown of actin within a particular filopodium will cause that filopodium to grow, conversely when the rate of breakdown is higher than the rate of build-up, the filopodium will retract. Interaction between the actin and the microtubules is believed to similarly affects the rate of extension or retraction of the growth cone as whole. Interestingly, growth cone motility is based on the same mechanisms that of other motile cells, such as white blood cells (Li et al., 1994).

For sensing to affect growth cone behaviour, contact between filopodia and the environment somehow affects the rate of actin/microtubule buildup and breakdown within the growth cone, so that contact with an attractive cue stabilizes filododia and contact with repulsive cues destabilize them. This phenomenon can be observed when growth cones are about to turn, at which point a large number of filopodia extend in the direction of future extension (Zheng et al., 1996). However, growth cone behaviour is complex, often characterized by high degree of seemingly random motion which only over longer time spans emerge as directed growth toward a target structure. There is also a high degree of variability among axons within a single class of neurons, with as much as a 3-fold difference in propagation speed (Halloran and Kalil, 1994).

The shape and behaviour of growth cones often vary as they traverse different tissues: When a growth cone appears to follow a simple cue such as a concentration gradient, it is typically small and stream-lined, and progresses at a quick and even pace without pauses. At "choice points", where the growth cones are about to change direction, the growth cone is typically larger with a more complex shape, it may extend axonal side branches with their own growth cones. At such points, the growth cone makes frequent pauses of highly variable duration, and often goes through repeated phases of advancement and rapid retraction (Halloran and Kalil, 1994).

Growth cone behaviour may be completely altered after a single filopodium makes contact with a specific cell, such as the case of the guide-post cells which cause extending axons to go through a $90^\circ$ turn (O'Connor et al., 1990; Palka et al., 1992). Such abrupt changes in behaviour indicates that the internal state of the growth cone determines how it reacts to external signals. It has been shown that changes in the internal concentration of the important signaling molecule cAMP causes some growth cones to switch from being attracted to being repelled by the same signal (Song et al., 1997). Intracellular $Ca^{2+}$ plays a similar role altering growth cone behaviour (Gomez and Spitzer, 2000; Zheng, 2000).

Growth cones transform into a synaptic termini during synapse formation. Growth cones have several features in common with synaptic termini: They commonly secrete neurotransmitters, which also play a role as guidance molecules (Hentschel and van Ooyen, 1999). Growth cones also carry ligand-gated ion channels, including NMDA receptors. In synapses, NMDA receptors are involved in the transduction of the glutamate neurotransmitters signals into action potentials at the postsynaptic terminus. In growth cones, the same receptors allow the growth cone to detect extracellular glutamate, causing growth cone turning (Zheng et al., 1996).

### 1.3.3 Synaptogenesis

Compared to axon guidance, synaptogenesis is less well understood. It is known that axonal growth cones select targets for synaptogenesis with extremely high specificity (Brose, 1999). Given the large number of different neuronal cell types in the target areas of many axonal pathways, the selection is probably mediated through combinatorial signaling. This means that each cell type is recognized by the absence or presence of $N$ different signals, in principle allowing up to $2^N$ different cell types to be distinguished. Important examples of such combinatorial signaling is the *Robo*-code (Goodhill, 2003) and the *LIM-hd* system (Bachy et al., 2001).

After the initial contact between the two cells, the axonal growth cone is converted into a presynaptic terminal. Tight binding between the two cells are mediated by a number of *cell* *adhesion molecules* (CAMs). The presynaptic terminal expresses signals that interact with receptors of the target cell, thereby inducing the target cell to form a postsynaptic terminal directly opposing the presynapse. Both the presynapse and the postsynapse are highly specialized structures, each containing cellular machinery for neurotransmitter secretion and reception, respectively.

In many cases it is not the main trunk of the axon that forms most of the synapses, but rather side branches (*collaterals*) which extend well behind extending the tip of the main axon. One study has found that the growth cone pauses and increases in size at the point where such collaterals will form after the main growth cone has continued on (Kalil et al., 2000). This may mean that while the collaterals form from the axon shaft, the growth cone has marked the spot where this is to occur, again highlighting the importance of the growth cone in forming neural topologies.

*En passant* synapses, which are also common in the central nervous system, are synapses that form directly on the main shaft of the axon. These form through a process that resembles the formation of collaterals. The axonal growth cone pauses while forming a structure that will become the presynapse. The axonal growth cone then continues on its way, while the synapse is completed (Hatada et al., 1999).

CAMs may be involved in more than binding the synapse termini together. There are a sufficient number of different CAMs that they may be used in combinatorial signaling. The trans-synaptic signaling inducing the formation of the cellular synaptic machinery may also be mediated by CAMs. There is even evidence for the involvement of CAMs in long-term potentiation. (See Brose, 1999 for a review of the role of CAMs in synaptogenesis.)

Synapses are highly dynamic structures. Early in development they are formed and destroyed at a very high rate as part of the exploratory growth of the nervous system (Innocenti, 1995). They are also believed to be the site where learning occurs. Activity dependent synaptic strengthening can cause the conductance of a single synapse to increase up to 50-fold (Chen and Regerh, 2000). Strong and stable synapses stabilize the structures that support them, while the loss of synapses can lead to retraction of axonal and dendritic branches (Innocenti, 1995; Cline, 2001).

## 1.4 Learning and innateness

The first indications of the existence of genes came from every-day observations of the phenomenon of inheritability — offspring tends to be more similar to their parents than to other individuals of the same species. At the same time it was clear that individuals are also influenced by the conditions under which they live. This gave rise to the central dichotomy of biology, the perceived opposites of *nature* and *nurture*, or in psychological terms, of the innate and the learned.

This dichotomy reflects the two components of most connectionist models of neural networks — the initial network topology represents nature, while the learning process represents nurture. Connectionist models span the spectrum from simple two-layer "perceptrons" that may need extensive learning (depending on their task) to elaborate hand-crafted networks that do not incorporate learning at all, see e.g. Hummel and Biederman (1992).

The primary focus of connectionism has often been on models of learning. One reason for this may have been the lack of biological information about the processes that shape neural networks. The early Hebbian model of learning (Hebb, 1949) has also fueled a lot of research into learning phenomena. The model is simple and powerful, and explains a wide range of observed learning phenomena.

### 1.4.1 Nurture

Neuronal tissues are, in comparison to other animal tissue types, highly complex. Neurons are by far the largest cells in the body[4], and they form precise connections with other neurons over large distances. Is the developmental system that build the rest of our body able to also build the brain? If not, what alternative processes might be involved?

One possibility is that neurons' unique information-processing abilities lend a helping hand, i.e., the developmental process is dependent on neural activity, and maybe on experience. The brains ability to recover from injury seems to support this view — following brain injury, areas of the cortex can to a large extent take over functions from other, damaged areas of the cortex (Vicari et al., 2000). In one experiment, the visual nerve was made to innervate the auditory cortex (Sur et al., 1990), and the resulting stimuli are interpreted

[4]When measured by their extent, "tip-to-tip", not when measured by volume.

by the animal as visual (von Melchner et al., 2000). It seems that either visual or auditory cortex can *learn* to process visual information simply by the fact that they receive input from the eyes.

However, there is a paradox in this flexibility: if the cortex is so flexible in how functionality are allocates to different regions, then why is it that the location of cortical centres are the same in all normal, healthy individuals? While experience clearly influences the structure of the nervous system, there must also be a genetic component to the arealization of the cortex.

### 1.4.2 Nature

If we suspect that there is a genetic component to the developmental control of neural tissues, it is natural to look at what is known about the development of other types of tissues. This raises two questions:

- Are the mechanisms known from other areas of developmental biology sufficiently powerful to produce the complexity of the nervous system?
- Is it likely that a separate, parallel developmental system has evolved just for neural tissues, and is there any evidence for the existence of such a system?

Investigations into the development of the vertebrate central nervous system have indeed revealed familiar mechanisms: domains of differential gene expression, concentration gradients of diffusible and membrane-bound signal molecules, and complex patterns of differentiated cells arising from uniform cell populations in discrete steps of domain subdivisions guided by signal gradients (Krubitzer and Huffman, 2000; Stern, 2001). All the elements of the developmental system shared among all animals are also found in the developing vertebrate central nervous system. Not only brain structure, but also function, is sometime genetically determined: In one case a congenital language deficiency has been traced to one mutant gene (Lai et al., 2001). This is a strong indication that in this case, genetic information is necessary, though not sufficient, for the development of a specific, well defined cognitive ability. These findings clearly indicate that conventional developmental mechanisms are involved in the nervous system.

However, perhaps there are other, parallel mechanisms at work which are unique to the brain and which account for the unparalleled sophistication of neural structures? To try to address this issue, we might consider the evolutionary challenge involved in developing new developmental mechanisms — the assumption being that if a novel developmental mechanism appeared to construct the brain, novel mechanisms should also have appeared at other times to account for the development of other organs, perhaps in other species. The fact is that novel developmental mechanisms do not seem to have appeared with any appreciable frequency during the evolution of the species. In fact, as far as a very large group of animal species called the bilaterians[5] are concerned, a single, shared developmental mechanism is used to form an enormous variety of body shapes, from flounders to praying mantids. To quote Davidson et al. (2002, p. 1677),"[t]he bilaterians all have more or less the same genetic toolkit, and in particular rely on essentially the same repertoire of regulatory genes to control the developmental organization of their body plans".

When a novel structure appears during the evolution of some species, the development of that structure may come about either through a novel developmental system, or else through the modification of an existing developmental mechanism. Current evidence appears to support the latter alternative as the more common mode of evolutionary change. This, together with the lack of any direct evidence of a special brain-building mechanism, supports the hypothesis that the brain is formed through the same developmental processes under genetic control as the rest of the body.

### 1.4.3 Activity dependent development

If a genetically driven developmental program is necessary for the development of the CNS, is it sufficient? Verhage et al. (2000) created mutant mice that lacked neurotransmitter secretion, meaning that during their entire prenatal development, there was no neural activity in their central nervous system. They found that the brains of the mutant mice at birth were anatomically indistinguishable from those of normal mice, including "formation of layered structures, fiber pathways, and morphologically defined synapses" (*ibid.*, p. 864).

While Verhage and coworkers observed that synapses had formed in their mutant mice, they could not tell whether the *correct* synapses where formed. Experiments indicate that

[5]That is, all animals with bilateral or left/right symmetry, including vertebrates and insects.

neural activity is indeed required for the correct formation of neural topologies such as ocular dominance columns and topographic maps (Crair, 1999). In these systems, it appears that that the activity-independent developmental system can only get it *almost* right, and that activity-dependent processes come in at a relatively late stage to fine-tune the neural connections.

### 1.4.4 A unified theory of neurodevelopment

The developmental process is the outcome of a tightly integrated collaborative effort of the old "adversaries", nature and nurture. The following quote is instructive as to how such questions might be approached:

> "[T]o try to determine how much of a trait is produced by genes and how much by the environment is as useless as asking whether the drum sounds that we hear in the distance are made by the percussionist or his instrument. On the other hand, if we pick up a *changed* drum sound, we can legitimately ask whether the difference is due to another drummer or another drum." (Kummer (1971), quoted in de Waal (2001), p. 8, author's emphasis).

Nowhere is this more true than in the nervous system. Through improved insight into the developmental processes that form the nervous system, we may come to understand the interplay between our genes and our environment that shape our brain and mind (Marcus, 2004).

## 1.5 Previous related work

Our goal is to build a software system which is able to simulate neurodevelopmental processes to a relatively high degree of accuracy. This means concurrently simulating the genetic control mechanisms of development, protein signaling which is used to coordinate these mechanisms in time and space, and neural activity which shapes neural connections through learning-based processes. In each of these areas, computer simulations have been used as research tools for a long time. This has given rise to a wide range of approaches to implementing these various mechanisms.

When it comes to simulating genetic regulatory systems, Hidde de Jong (2002) lists a number of different approaches, including Bayesian networks, a range of differential equation-based approaches, stochastic equations, boolean networks, rule-based formalisms and others. Differential equations as a way to represent genes has a strong appeal, since a gene's expression rate is an important component of the time-differential of the corresponding protein's concentration. However, as will be apparent from the set of actions listed in the remainder of this chapter, NeuroGene simulations encapsulates many processes which, at the level of abstraction at which we operate, are instantaneous (such as cell death, cell division, etc.), making such an approach unsuitable. Another problem with a differential equation based approach is the large additional memory requirement of an adaptive step Runge-Kutta routine for solving a very large number of coupled equations. See also Stanley and Miikkulainen (2003) for a more recent review.

We have instead decided to create a rule-based system, where genes are rules with an antecedent (called *regulation*) which defines the expression behaviour of the gene, and a consequent (called *effects*) which encodes the changes caused by the gene's expression. The genes are encoded in a dedicated genetic programming language. The simulation proceeds in discrete time steps, and each gene is evaluated within each cell during each time step.

In order to put our system into the context of other similar or not so similar computational systems, our concepts of genes and cells need to be compared to other conceptualizations of the same biological entities within other computational frameworks.

### 1.5.1 Computational conceptions of genes

#### Genetic algorithm

Genetic algorithm (GA) comes readily to mind as a computational approach which draws on the biological concept of genes. GA simulates the biological process of evolution, i.e., the gradual genetic change in a population over many generations. It is used as a search algorithm to find near-optimal solutions to (typically) NP-hard problems. The "genes" used in GA are therefore represented in ways which cater to a straightforward implementation of evolutionary processes such as point mutation, chromosome cross-over and sexual reproduction. Genes in GA also need to encode properties of biological organisms (the *phenotype*). It is the phenotype that is optimized through the GA. The translation from the GA genes to the phenotype corresponds in principle to the developmental process by which genes determine the structure and function of an individual organism. However, in practice, this translation rarely bears any resemblance to the biological process of development.

By comparison, we do not seek to simulate evolutionary processes. Instead, our representation of genes is shaped by our goal of biologically accurate simulations of developmental processes.

### Gene sequence analysis

A wide range of computer software exists for the analysis of actual gene sequences, such as those discovered by the Human Genome Project. Such genetic sequences consists of very long strings of the four letters A, G, T and C. Such sequences are meaningless to human readers without such analysis, which can be used to determine where genes start and end, what the amino acid sequences of the corresponding proteins are, and what the structure and function of those proteins might be.

By comparison, we seek an encoding of genetic information that is easily readable by human users, and which is able to represent causal relationships between genetic and developmental processes as outlined above. To achieve this goal, we select a level of description of genes which does not include DNA sequence data or explicit molecular interactions, since these are not easily grasped by human readers. By abstracting away the details of the molecular interactions we arrive at a representation of genetic information which only retains the causal relationships between the expression of genes and cellular changes. This is consistent with how biologists represent some forms of genetic information, as was outlined in section 1.2.6: It is the level of description which emerges from questions 1–2 (page 12).

### Gene network analysis

Gene network analysis is the attempt to elucidate the causal relationships between the expression of large number of different genes. The problem of elucidating the structure of gene networks is closely related to our goal, which is to simulate such networks and their effect on development. A common approach to elucidating such networks is to collect a number of "snapshots" of an organism in different states of gene expression. From each snapshot the expression rates of a very large number of genes are determined. In the software used for the analysis of such gene expression data, genes may be represented by a vector, where each vector element specifies how each other gene affects the expression rate of the gene in question (Dutilh and Hogeweg, 1999). Alternatively, each gene is represented by a boolean function. The function takes as parameters the presence or absence of the proteins of all other genes, and it is evaluated to determine whether or not the gene is expressed in a particular situation. Finally, differential equations may be used to encode the rate of change in the concentration of each protein as a function of the current concentration of all other proteins, as in the system Ingeneue (von Dassov et al., 2000; Meir et al., 2002).

These representations of genes have the same expressive power that we will need for the encoding of the expression behaviour of genes within NeuroGene. Our general level of abstraction is also close to what is used in gene network analysis, where the interactions between genes are represented using logic and/or algebra. However, we do require the additional ability to encode the *effects* of gene expression in terms of developmental processes such as cell division, migration, etc.

**Algorithmic gene languages**

As our understanding of the mechanisms of gene regulation and expression has improved, a picture has emerged which may be described as "algorithmic", where each gene is a potentially complex computational unit, and gene expression causes changes in the structure and function of the cell according to the "algorithms" encoded in the genome. This algorithm is implemented through linear and non-linear processes which construct enhancer complexes and thereby causally affect gene expression (see figure 1.2, and also the quote on page 6). These causal relationships can be straightforwardly represented using algebra and/or boolean logic, with no need to refer to or even understand the underlying molecular interactions. It is therefore natural to conceive of encoding genetic information in the form of algorithmic programming languages. A number of different systems have taken this approach, with different degrees of adherence to biology.

Agarwal (1995) has created a cellular programming language (*CPL*), in which a genome consists of multiple programs, each related to a particular cell or cell type. Cell types are also explicitly represented, including instructions which cause a cell to change cell type. The system by Astor and Adami (2000), discussed below, also uses an algorithmic representation of genetic information.

### 1.5.2 Computational conceptions of cells

#### Lindemayer systems

L-systems (Lindenmayer, 1968) are used for the simulation of developmental processes, primarily in plants (Prusinkiewicz et al., 1995), but they have also been applied to neural structures (DeVaul and McCormick, 1996). L-systems use a grammar to define how parts of the organism changes through development. At any point in time, the organism consists of a number of components, some of which are *terminal* and some which are *non-terminal* with respect to the grammar. The components of the organism at time $t$ which are non-terminals in the grammar, are expanded according to appropriate productions to form a new structure at time $t + 1$. This new structure may consist of both terminal and non-terminal components, and the process of production expansion is repeated. The conversion of grammar elements into three-dimensional shapes usually involves a turtle based system (Logo Foundation, 2004). Individual cells may or may not be explicitly modeled in L-systems — in the case of neural structures, the components generated by the L-system are actually sub-cellular structures such as axons, dendrites and synapses.

L-systems have the ability to generate complex tree- or plant-like structures with potentially high levels of control. While the mechanism by which L-systems create these structures is likely very different from the mechanism which forms these structures in nature, from a pragmatic stand point, L-systems represent a well established methodology for producing these complex shapes.

#### Two-dimensional hexagonal cell systems

Simulators used to study gene networks need to represent cells in a way which allows them to accurately model cell-cell interactions. The system Ingeneue (von Dassov et al., 2000; Meir et al., 2002) represents a two-dimensional static array of uniform, hexagonal cells. This data structure allows the representation of interactions between receptors on the surfaces of adjacent cells. However, it does not allow for cell migration, cell division or cell death. A slightly more flexible system (Agarwal, 1995) uses a similar 2D hexagonal lattice, but each cell may occupy more than one hexagon, with the condition that each cell constitute a contiguous area of the lattice. Under this generalization, cell migration and division are possible. However, the restriction to 2D remains, and the data structure is poorly suited to incorporate axons, dendrites and other small cellular components.

### Representing cell types

Biologists have for a long time recognized the existence of *cell types*, such as liver cells, muscle cells and many different kind of nerve cells, etc. Cells of a particular type may have recognizable shapes, act in particular ways, and will express genes that are specific to the cell type. However, it is also recognized that cell types are artifacts of the researchers categorizing their discoveries — as far as nature is concerned, cells differ only by which genes they express and the structural and functional consequences of particular patterns of gene expression.

Simulation systems differ by whether they explicitly represent cell types, and what role the cell type plays in controlling development. The systems by Agarwal (1995) and Cangelosi et al. (1994) both represent cell types — genes may cause cells to change cell type (a process called *differentiation*). The genetic instructions which determine the behaviour of cells is specific for each cell type. The system created by Astor and Adami (2000) also represent cell types through a mechanism based on special cell-type genes. Here the genetic instructions are shared among cell types, but individual genes may be labeled as being expressed only in certain cell types.

As stated above, cell types do not represent biological reality. Since the effect of cell-type specific gene expression is simple to achieve without explicitly representing cell types, we will not do so in implementing NeuroGene. Instead ordinary genes which remain expressed through the lifetime of cells will be used to identify the cell in terms of its cell type.

## 1.5.3 Simulating growth cones

The central role of growth cones in the formation of the nervous system have made them interesting subjects of computer simulations. Hentschel and van Ooyen (1999) simulated the movements of growth cones under the influence of soluble and membrane bound attractive and repellent proteins. A goal of that study was to improve the understanding of axon fasciculation or bundling. They did not include in the model aspects of growth cone structure or function, such as filopodia. Instead, the signals both produced and sensed by the growth cones are focus of this study.

Meinhardt (1999) looked at the internal amplification that must necessarily occur within a growth cone in order for shallow external concentration gradients to result in highly asymmetric growth cone structures. In this study the concentration gradients of signaling molecules within the growth cone are explicitly modeled. These gradients are central to how the growth cone transduces and responds to external directional cues. This model may explain why filopodia tend to form preferentially in the direction of axon extension (i.e. toward attractive and away from repellent signals).

Buettner et al. (1994) developed a model of growth cone motion based on explicit modeling of filopodia. The model consists of the following features: The initiation timing and lengths of filopodia are stochastic, while the speed of extension and retraction occur with constant speed. Values for all the parameters of the model where taken from statistical analyses of the motion of real growth cones *in vitro*. Using this model, the authors modeled the movement of growth cones over a substrate consisting of areas of permissive and non-permissive surfaces. The assumption was made that the growth cone would only cross bands of non-permissive surface if at least $N$ filopodia extended across it and touched the permissive surface on the other side. Based on this assumption, the model does fairly well in reproducing the probability of growth cone crossing the non-permissive gaps, at various gap widths and various concentration of permissive substrate.

In nature, the structural framework of growth cones consists of two types of fibres: Microtubules give structural integrity to the axon, and extends into the growth cone, while actin fibres constitute the more dynamic structure within the growth cone itself. Li et al. (1994) have simulated the actin dynamics of growth cones, modeling a 2D slice through the three-dimensional growth cone as a hexagonal network of contractile fibres. They are able to compare the effects of different aspects of the membrane and actin dynamics, such as membrane adherence to the substrate, the rate of growth and breakdown of actin, the pressure exerted by the intracellular fluid on the membrane, and a constrictive force acting on the cell membrane at the neck of the growth cone.

Finally, Tim Hely (Hely, 1998, chapter 5 and Hely and Willshaw, 1998) has constructed a model of growth cone dynamics which includes both microtubules and actin. The actin is not modeled explicitly, but the effect of actin dynamics on the growth and shrinkage of microtubules are simulated. The model shows how the encounter of a growth cone with a target cell can stabilize the actin within the contacting filopodium. This in turn causes selective extension of the axon toward the target cell through interactions between actin and microtubules within the growth cone.

### 1.5.4 Simulating neural activity

#### Precise neural models

Two important software packages exist for the precise modeling of neurons, *GENESIS* (Bower and Beeman, 1995) and *Neuron* (Hines and Carnevale, 2003). These are capable of modeling single neurons with very high accuracy, to the point of explicitly modeling individual ion channels. They are also able to model neural networks consisting of tens or hundreds of neurons using more abstract neural models. However, neither of these incorporate developmental changes in the network structures. They are also computationally very expensive, making them unsuitable for our use.

#### Connectionist neural networks

The investigation of connectionist neural networks is an important field of cognitive science. Such networks represent abstract neural circuits, consisting of *nodes* (representing neurons) and *connections* (representing synaptic connections between neurons). By mimicking neural activity in the form of either action potential (in "spiking networks") or time-average firing rates (the more common approach), combined with learning rules which adjust synaptic conductivities (or "weights"), such networks can replicate important phenomena of learning and other cognitive processes.

While the higher level of biological accuracy of the *GENESIS* and *Neuron* type simulations is appealing, the effort involved in implementing corresponding functionality within the NeuroGene system is prohibitive. The neural models which have been incorporated into NeuroGene are thus closely related to the simpler models used in connectionist neural nets. The neural activity is integrated with the genetic regulation of cellular activity. This is important, since it allows the *bidirectional coupling* between genetic and neural activity, allowing each to affect the other.

### 1.5.5 Neurodevelopmental simulators

Several software systems have been developed for the simulation of neurodevelopment. The system by Cangelosi et al. (1994) uses genetic algorithm (GA) to discover genomes that guide the development of neural networks which are able to solve a simple task. The system defines 16 different cell types, each of which are guided by a different gene. The cells are able to migrate through the 2D simulation space, divide and grow axonal arbor structures of regular tree-like geometries. In order to evaluate the organism fitness value for the GA, an external space is also simulated, through which a virtual robot controlled by the neural network can move, with the goal.

Fleischer and Barr (1993) have created a system for developing neural networks. It consists of cells which are free to move about in a 2D environment. The genome is modeled as a set of conditional differential equations of the form

$$\textbf{if}\ \mathsf{Condition}(state, env)\ \textbf{then}\ \frac{dstate[i]}{dt} = \mathsf{Consequent}(state, env)$$

where *state* is a vector of variables defining the state of the cell, and where the terms of the **Condition** and the **Consequent** depend on the current *state* and the environment *env*. Through this mechanism, this system captures both the continuous and discontinuous aspects of gene expression. Physical processes such as diffusion in the extracellular space are modeled as well. They simulate growth cones, but do not explicitly represent filopodia.

Another system (Rust et al., 1999, 2000) again uses genetic algorithm to optimize the genetic information which controls the developmental process. Neurons are constructed through processes which are inspired by biological developmental process, relying on extracellular protein gradients, simulated growth cones, interactive branching of extending axons, etc. A system created by Astor and Adami (2000) uses genes encoded as algorithmic programs, simulates certain physical and chemical processes, and using a hand-coded genome, can build a neural circuit which can "learn" Pavlov's conditional response reflex. Finally, there are other systems for the simulation of neurodevelopmental processes which have more practical applications, e.g., the one by Peter Eggenberger, which is directed toward the design of robots (Eggenberger, 1997a,b).

### 1.5.6 Properties of NeuroGene

The system developed by Astor and Adami (2000) embodies many of the features that are also present in NeuroGene: separate representation of regulation and effects of gene expression, a common genome for all cells, a genetic programming language, developmental control of the formation of neural circuits, representation of certain physical and biochemical processes, including diffusion. Like some other systems, this simulator operates on *relative* protein concentrations which are restricted to the range from 0 to 1, with 1 representing the *saturation level* of each protein.

Our system improves on that of Astor and Adami in several important ways. What sets our system apart from earlier works is the range of developmental mechanisms supported, the close correspondence between genetic control structures and our genetic programming language, the simulation of ligand-receptor binding which is integral to many competitive developmental interactions (see e.g. van Ooyen and Willshaw, 1999) and the coupling of neural activity to the developmental system. All of these elements are central to furthering the understanding of the development of biological neural networks.

NeuroGene has a more powerful and general representation of protein concentrations, including distinct membrane bound and diffusible proteins. This allows a more powerful (and readable) genetic programming language which reflects more closely biological reality. To further gain biological plausibility, we dispose of the concept of cell types. Instead, we accommodate asymmetric cell division, which allows a parent cell to give rise to daughter cells which are arbitrarily different from the parent. All genes are potentially active within all cells, subject to the expression behaviour of each gene. Absolute protein concentration levels are used — these are more intuitive, and allow the straightforward implementation of ligand-receptor binding. Explicit modeling of growth cones, including the competitive interactions among filopodia allows accurate simulation of axon guidance. We also use a 3D rather than a 2D simulation space. Powerful and flexible visualization systems may be used to show the current state of the simulation, both for presentation of results and also as a tool during the development of new simulations.

Within the NeuroGene data structures, we explicitly represent the 3D spatial relationships between different cell components of the simulation. This sets NeuroGene apart from the majority of connectionist systems in which "nodes", representing neurons, are typically fully characterized by the connections they make to other nodes (i.e. the topology), and no spatial arrangement of the nodes in three-dimensional physical space is given. The spatial representation of neurons allows biologically plausible representations of sensory input which often have spatial components. In addition to the obvious case of the visual images, aural (Kandel et al., 2000, p. 1127), olfactory (from the sense of smell, *ibid.*, p. 630ff), somatosensory information (from the sense of touch, *ibid.*, p. 458) and sensation from internal organs, such as hunger (*ibid.*, p. 974) are known to be represented in a topographic manner in the vertebrate nervous system. Even the sensation from whiskers in animals such as cats and rodents are transferred in a topographic manner to the brain (*ibid.*, p. 707ff). This means that sensory information from adjacent locations in the body are directed to adjacent target locations within the brain. In each of these cases, each cell in a population of neurons receive dissimilar neural input based in their distribution through the 3D space of the brain.

The remainder of this thesis consists of two chapters on methodology, followed by two chapters outlining experimental results, and finally a chapter on conclusions and future work. A number of appendices are also included. In chapter 2, I outline the biological mechanisms which are simulated by NeuroGene, and the approaches used to implement these aspects of the simulator. In chapter 3, the software engineering aspects of the implementation of NeuroGene are laid out, including the design of the NeuroGene language parser and the graphical user interface. In chapter 4 results are given which verify the correctness of simulations of physical processes, while the results of three simulations of developmental processes are presented in chapter 5. Conclusions and perspectives on future work is given in chapter 6.

# Chapter 2

# Methods I: Biological principles and algorithmic analysis

In this thesis I divide the description of the methodology into two chapters. This first methods chapter outlines the biological processes that NeuroGene simulates. I also discuss how these mechanisms have been captured in algorithmic form, and the approximations and simplifications used to make simulations practical. In the subsequent chapter I describe implementational issues, including the data structures which represent the genome and the state of the simulation, as well as the system architecture, user interface, file I/O etc. In simple terms, this chapter is about the biology of NeuroGene, and chapter 3 is about the software engineering. I will briefly introduce elements of the system design in this chapter only when it is required to understand the context of the implementation of the biological processes described.

In section 2.1, I discuss how best to represent genetic information in data structures within NeuroGene. This includes discussions of which aspects of genes' structure and function are relevant to the type of simulations we envisage NeuroGene to be able to carry out, and how these aspects can be abstracted. Given that supplying genetic information for NeuroGene to work on is an important mode of user interaction with the system, mechanisms of input of genetic information into NeuroGene, and the form of that input, must also be considered. In sections 2.2 and 2.3 the representation of cells and the space in which they exist is discussed. Here the requirements of biological accuracy and computational efficiency need to be reconciled. In sections 2.4–2.6 the implementation of genetic control over the simulated developmental process is outlined. This includes the mechanisms of gene regulation, protein production and simulated growth cones. In section 2.7 the implementation of simulated neural activity is discussed — here we adhere quite closely to common approaches from connectionism. However, the integration of neural activity with the genetic system is, to our knowledge, novel. In section 2.8 the overall architecture incorporating the cells, simulation space, genome and other components is described. Section 2.9 describes how simulations are initialized, and finally section 2.10 explains how the different elements are executed and coordinated during NeuroGene simulations.

## 2.1 Representing genetic information

Early in the planning phase of the NeuroGene simulation system, it became apparent that we would need a way to encode genetic information that was powerful enough to capture a wide range of genetic interactions, as well as a wide range of developmental mechanisms. The requirements of the system were:

- Genes should be readily human-readable, more specifically they should be readily understandable by somebody with a biology background and less knowledgeable about programming and computer science in general.
- Simulations should run as efficiently as possible, primarily with respect to CPU time, but also to memory. Since simulations would access genetic information frequently, this requirement puts demands on the internal representation of genetic information.

Alternative approaches we considered to representing genetic information include:

### Encoding based on the DNA code

A possible encoding of genetic information might resemble the DNA encoding. This solution was never seriously considered, mainly because the DNA code is inherently obscure. Indeed, while the complete sequence of the human genome is now known, it is an enormous challenge for the biological community to decode this text. Using some form of nature's genetic code to encode the genes in NeuroGene therefore violates the language design principles of expressiveness and maintainability (Louden, 1993), since genomes encoded in this way would be very hard to read and write.

However, certain elements of the encoding of genetic information in DNA would be carried over to the representation used in NeuroGene. Genes consist of distinct regulatory elements and coding elements, which are encoded in distinct sections of the genes' DNA. Similarly in NeuroGene, the regulatory elements of each gene will be encoded independently of the part of the gene which determines the effects of the gene's expression. Other elements of the biological encoding of genes will *not* be carried over, including the fact that most cells carry two copies of each gene, as well as the intron/exon structure of genes (see e.g. Muller, 1997).

#### GUI-based rule editor

One possible approach was to design a graphical user interface (GUI), which would allow the user to build genes from simpler building blocks. This approach would necessarily be combined with some file format for storing the genomes, but the file format need not be human-readable. From a human-computer-interface (HCI) point of view, a programming language and a graphical user interface can be seen as two different types of user interfaces. However, a programming language is much more scalable with respect to the complexity of what it encodes than a GUI. Consequently, we decided against the GUI approach.

#### Using an existing programming language

We might also encode the genes in an existing programming language. In our case, Java would be the natural choice, since the rest of the system would be written in Java. There are two problems with this approach:

Firstly, it was essential that NeuroGene simulated genomes be readily understood by researchers with a biological background. This meant that the simulated genes should be encoded at a level of abstraction which is also used by biologists themselves (section 2.1.3). We chose a level of abstraction where only the causal relationships between gene expression, signal transduction, neural activity and cellular change are represented, and the molecular interactions underlying these relationships are omitted. At this level of abstraction, genes have constituent parts which define their expression behaviour and their effects on the cell. Existing programming languages do not have programming constructs that are similar to genes in this way. Using such a language to represent genetic information would therefore require significant amounts of programming code not related to the genetic information itself, probably rendering the resulting genome code obscure to biologists.

Secondly, we would not have easy access to the internal representation of the genome, since this representation would be compiled Java code. Future development of NeuroGene may include the simulation of mutations in the context of a genetic algorithm (GA) optimization of the genome. Implementing mutation without access to the internal representation of the genome would not be possible. However, since GA is not implemented yet, and only may be implemented in the future, this requirement is not absolute.

On the other hand, the advantages to using Java to encode the genetic information are undeniable. These include the ability to write libraries of genes, as well as easy access to existing tools such as compilers, editors, etc. It would also remove the need for the NeuroGene parser system, which would lead to a significant reduction in the complexity of the NeuroGene system. There are therefore compelling arguments both for and against this approach, and it is possible that a future version of NeuroGene will use Java to encode the genome.

### Production system

The modern model of gene regulation has a lot in common with the methodology of production systems (Luger and Stubblefield, 1998, ch. 5) commonly used in artificial intelligence and expert systems research.

Production systems consist of a set of *rules* which embody the knowledge which will be used to solve a particular problem. A *working memory* is initialized with the problem state. Through the application of rules, the working memory is altered until its state is recognized as a valid solution to the problem. Control of the system comes from a *recognize-act* loop: In the *recognize* phase, all rules whose conditions match the current state of the working memory are selected. In the *act* loop, one of these rules is selected (based on some heuristic) and the working memory is altered according to the action which is associated with that rule.

To relate production systems to gene regulation and developmental control in nature, genes correspond to rules, where the condition of the rule corresponds to the regulatory elements which determine whether or not the gene is expressed in a given cell at a given time. A rule's action corresponds to the changes caused by the gene's expression, considered either narrowly as the production of a certain amount of a particular protein, or more widely as including the changes induced by expression of other down-stream genes, these proteins' interactions with cellular structure, etc. The working memory corresponds to the state of the cell, including the concentrations of all regulatory proteins, and the recognize-act control cycle corresponds to the gene expression machinery of mRNA transcription and protein translation. There are therefore many similarities between our approach to the simulation of gene expression and developmental control and production systems.

### Situation calculus

Another AI approach with similarities to NeuroGene is situation calculus (Levesque et al., 1998). Within this formalism, a ***situation*** $s$ may give rise to a new situation $s'$ by the application of some ***action*** $a$. A situation $s$ is characterized by relational and functional ***fluents***. These have values that depend on the situation, such as in a given situation $s$ the light may be on (relational fluent), and the temperature may be 25°C (functional fluent). A ***fluent action*** is a "meta-action" which is evaluated in a given situation $s$ to give some action $a$. Which action results depends on the situation $s$. In NeuroGene, both genes and growth cone functions would be fluent actions.

For each type of action we can ask two questions: Can the action $a$ occur in situation $s$, and, if yes, how is the situation changed by this action. The former question is handled by ***precondition axioms***, of which there are one for each type of action, e.g. to turn on the light (action), the light must be off and you must stand by the light switch (precondition). The answer to the latter is encoded in ***successor state axioms***, for which there is one for each fluent/action pair, such as: the room will be lit (fluent) after you have turned on the light (action).

A challenge for situation calculus is the frame problem (Reiter, 1991), relating to how to specify all the fluents which are not affected by any given action (turning on the light does not affect the temperature). With the introduction of concurrent actions (Reiter, 1996) comes the additional problem of precondition interactions: This problem occurs when two concurrent actions are individually possible but mutually exclusive. For example, standing on the curb (situation $s$), I can catch a taxi (action $a_1$) or a buss (action $a_2$), but I can not do both concurrently. In developing a situation calculus formulation of NeuroGene[1], the frame problem did not appear to pose serious problem. However, some problems that had already been encountered and solved were identified as precondition interactions.

### Markup language

Given that the choice of approaches has been narrowed down to developing a new programming language, a natural choice might be to base this new language on the widely adopted XML (DuCharme, 1999) standard for encoding structural data. While XML was initially designed for storing data, it has recently also become common to use it to store program code, such as XSL. By adhering to the XML standard, we would be able to take advantage of a number of existing tools that recognize the XML format, including sophisticated text editors, existing Java XML parsers, etc. However, code written in markup languages tends to be very verbose and difficult to read, which made us decide against this approach.

### Dedicated genetic programming language

By method of elimination, the choice fell on designing a new programming language for encoding genetic information. Certain superficial syntactic elements where copied from Java, including using curly braces ("{" and "}") for delimiting blocks, and the function call syntax. However, the gene control structure is unique to NeuroGene. It is designed to capture biologists' description of causal interactions between genes and the cellular system at a particular level of abstraction. Several other language features, such as cell initialization blocks (see section 2.9 below) also do not have counterparts in general-purpose programming languages.

[1]Available upon request.

**Chronology of system development**

NeuroGene draws on a prototype system created by Gary Marcus. This system used the GUI rule editor approach. For NeuroGene, we initially decided on a dedicated and novel programming language for encoding genetic information. This language underwent significant changes throughout the early development of NeuroGene, including a complete rewrite of the parser using a different parser generator tool (the first versions used the parser generator JavaCUP, Hudson et al., 2003). The scope of the language has also grown over time, with the addition of a symbol table and lexical scoping, the growth cone mechanism, etc.

Alongside the development of the NeuroGene simulator, we have also developed several developmental simulations, three of which are presented in this thesis. These simulations have served as tools for debugging the NeuroGene source code. They have also revealed shortcomings in the capabilities of NeuroGene, and thereby served as guides for the further work on the simulator itself. This means that the current set of capabilities of NeuroGene has to some extent been determined by the requirements of these particular developmental simulations. However, we have endeavored to keep NeuroGene as general as possible, mainly through two approaches: Firstly, the simulations we have chosen to explore cover a wide gamut of developmental mechanisms. Secondly, most primitive operations have been designed with user-definable parameters that allow them to be used in a range of circumstances.

### 2.1.1 Biological justification of actions and queries

Genes in nature can be seen as computational units with input and output channels. The input channels are the regulatory elements of the gene, through which the abundance of relevant regulatory proteins determine the expression rate of the gene (see figure 1.2). The output channel is the rate of production of the protein encoded by the gene, and indirectly, the effect the increased concentration of this protein has on the structure and function of the cell. In NeuroGene, this same input/output causal system is modeled using an abstract representation of both the regulatory elements, the produced protein, and its effect on cellular properties and behaviour.

However, gene regulation and expression as simulated in NeuroGene represents more than just the regulation of gene transcription in nature. The NeuroGene gene mechanism represents both transcription and translation of genes. By virtue of the fact that simulated genes are evaluated throughout cells, and a gene's expression rate can vary across the cell, the gene mechanism also represents the effects of intracellular active transport of mRNA and protein, without modelling the transport explicitly. Furthermore, since simulated gene expression can be directly affected by extracellular events and conditions using a wide array of queries, the gene model also represents intracellular signalling pathways which communicate such events and conditions to the sites of protein transcription and translation. Finally, through the large set of actions listed below, simulated gene expression may directly cause changes in the state of cells. This means that the NeuroGene gene model also represents the consequences of molecular interactions involved in bringing about such changes, again without modelling the molecules themselves or their interactions. In cases where actions represent changes that in nature involve expression of other genes, a simulated gene which invokes such actions would represent biological master control genes (such as Pax6 described in section 1.2.2) whose protein affects the expression rate of these other genes. In the remainder of this thesis, this sense of the term is always intended when referring to NeuroGene simulated genes.

The regulation of a particular simulated gene is represented by a mathematical formula, specified using the genetic programming language, which is evaluated within the context of a cell component at a point in time to give the expression rate of the gene in that cell component at that time. The mathematical formula is defined in terms of *queries*. When the expression rate is evaluated, any queries within the formula are also evaluated to give values which depend on the state of the cell. Through this mechanism, the influence of the state of the cell on the gene expression rate is modeled.

In this chapter, a large number of queries will be presented. Some of these, like **internalConcentrationOf**($p$) which returns the intracellular concentration of some protein $p$, can be used to represent mechanisms of gene regulation which are very close to what occurs in nature — intracellular regulatory proteins directly affect gene expressions through binding to gene regulatory elements. Others, such as **insidePreSynapse**() which returns **true** in presynaptic cell components and **false** in all other types of cell components, are more abstract. To justify such queries, we assume that there exists some mechanism by which expression of certain genes only affects presynaptic terminals. Importantly, data from biological research allows us to judge whether such assumptions are warranted. In this case, there is evidence that intracellular transport of mRNA and/or protein to specific sub cellular areas, such as synaptic termini, does occur in nature (Lodish et al., 2000, pp. 809–817). Where available, relevant biological data will be presented along with the queries. Using these queries, the expression of some gene as a function of the concentration of some regulatory protein $Q$ can be specified as (for example)

```
if(insidePreSynapse())
    expressInternally(pow(internalConcentrationOf(Q), 2)).
```

Here *pow*() function was included to show the potential for modeling complex expression behaviour. *pow*($x$, $y$) evaluates $x^y$, and a square relationship as shown here is common in nature (Smolen et al., 2000b), and may be explained by two molecules of $Q$ binding to the gene's regulatory elements. The action ***expressInternally***(), which specifies intracellular expression, will be described in section 2.5 below. Please refer to appendix B for examples of genomes encoded using the NeuroGene language.

The existence of certain queries, such as **cellIncomingWeight**() which returns the total weights of all synapses coming into a cell, does not at present receive strong support from experimental data. However, including them may still be justified: **cellIncomingWeight**() is required for NeuroGene to be able to implement connectionist models of learning which commonly rely on various forms of normalization of synaptic weights. The inclusion of such unsubstantiated queries within the genomes of simulations makes underlying assumptions readily apparent. The validity and biological relevance of a NeuroGene simulation can therefore to a significant degree be estimated from an inspection of the NeuroGene genome controlling the simulation.

The effects of the expression of a simulated gene is represented by *actions*, of which a number are also presented in this chapter. Actions model the effect of the presence of proteins in the cell. Changes in protein concentrations in the cell cause changes in the structure and function of the cell by affecting systems within the cell, which as a rule consist of proteins. Such mechanisms are represented by actions. Similarly to queries, there are actions at different levels of abstraction. However, actions are generally more abstract than most queries, since they commonly represent cellular changes which involve many interacting proteins. For example, the action ***endocytose***($p$, $x$) causes to an amount $x$ of some protein $p$ to be brought inside the cell from its environment. This is a common cellular process which is reasonably well understood (Lodish et al., 2000, pp. 727–733), however, modeling it in detail including all the proteins involved is beyond the scope of NeuroGene. An example of a more abstract action is ***buildPreSynapse***(*bid*, *weight*), which causes an axon to form a new synapse with weight *weight* with a given other cell component. (The parameter *bid* will be explained in section 2.6 below on simulated growth cones.) In using this action, we assume that growth cones are capable of forming synapses, that the axon initiates synapse formation, and that the synaptic weight is determined when the synapse is formed. The first two find support in experimental data (reviewed in section 1.3.2). We can infer that there must be some biological mechanism which determines what the synaptic weight of the nascent synapse is. Since we do not know this process, the parameter *w* allows us to specify the initial weight. In simulating synapse formation, we also abstract away complex changes in cell morphology and cell membrane specialization that are involved in synapse formation.

Like many actions, ***buildPreSynapse***(*b*, *w*) takes numerical parameters. The values passed to the actions may be computed from queries, such as for example

```
buildPreSynapse(1, 1/internalConcentrationOf(Q)).
```

The statement above models a causal relationship between the concentration of some protein $Q$ and the synaptic weights of nascent synapses. The large number of queries and actions supported by NeuroGene allows for a wide range of such causal relationships to be represented. Some such relationships will be biologically plausible, while many will not. In the interest of being able to support a wide range of possible simulations at varying levels of abstraction, a large number of primitives has been included, many of which take parameter values that further extend their scope. The biological plausibility of a NeuroGene simulation must always be judged based on how actions and queries are used within the genome controlling the simulation, rather than based on the queries and actions themselves.

The NeuroGene gene model can be used to represent biological genes, but is also capable of representing a part of a biological gene (such as transcription or translation, allowing these processes to be modelled individually), or multiple genes (such a master control gene and all its subordinate genes). There are also situations in which simulated genes are used to represent biological entities or processes which are not genes. This includes genes which represent receptor-ligand complexes. While such complexes exist in nature, they are not the direct product of gene expression (their component elements usually are), and the NeuroGene genes which represent ligand-repeptor complexes therefore do not correspond to genes in nature. Also, simulated genes may represent the consequences of cellular changes (e.g. cell division) which are triggered when a cell and its environment have particular properties. In future versions of the NeuroGene system, changes may be made so that such processes are modelled using a different mechanism from the gene model, see section 6.4.

### 2.1.2 On the term *protein*

This is a short aside on the use of the term protein in this thesis. The term ***protein*** is descriptive of the chemical composition of a class of chemicals that are important in all biological systems. Since all proteins are made in the same way ***in vivo***, the term is also descriptive of a particular mechanism of production. This mechanism is directly related to the encoding of information in the genes of all biological organisms, see figure 1.1.

An alternative term would be ***signal***. This describes the role which many, but far from all proteins play. I will at times refer to chemicals which are not proteins, and which act as signals in controlling developmental processes. In cases where chemicals are used as signals, their composition are irrelevant, much in the same way the form of words in human languages are arbitrary with respect to their meaning — the word "cat" has no relation to furry animals except for the fact that all English speakers agree that "cat" refers to furry animals. Similarly, it is not generally relevant to a discussion of the function of insulin to know that it is a small protein, and similarly irrelevant that dopamine is a biphenol, cAMP a nucleotide, etc. — what matters is how cells' behaviour is affected by these signaling molecules. A term used in the biological literature is ***morphogen***, which similarly emphasizes function, since these chemicals affect the ***morphology*** of biological structures.

While proteins by definition are two degrees of separation (transcription and translation) from the genetic code, other signaling compounds are also created under genetic control, albeit more indirectly: For dopamine to be produced, the protein (in this case an enzyme) *DOPA decarboxylase* must be present — the regulation of the gene encoding the protein *DOPA decarboxylase* therefore indirectly also regulates the level of dopamine. Consequently, all chemical substances discussed in this thesis are produced, directly or indirectly, under genetic control. Almost all of them play the roles of signals, coordinating developmental processes over time and space.

I have chosen to use the term *protein* throughout this thesis when referring to signaling molecules of all kinds. While this is somewhat inaccurate use of language, it emphasizes the close relationship to genetic information, which plays a central role in this thesis. It also avoids a possible misinterpretation of the term signal, which might be understood as immaterial. All signaling molecules are of course material, as opposed to e.g. action potentials, which is an immaterial representation of information.

### 2.1.3 Enforcing biological plausibility

All connectionist models raise the question of whether or not they are biologically plausible, i.e., whether their neural networks could conceivably exist as part of the human brain. Our goal is to try to settle such questions by allowing the faithful replication (at a certain level of abstraction) of the biological processes which form neural circuits in the brain, and to form connectionist models through the simulation of biological developmental processes.

There are two ways in which this goal can be achieved: Either design the simulator in such a way that biologically implausible processes are precluded, or design the genetic encoding in such a way that it relates directly to biological processes, making implausible mechanisms readily apparent. While the first approach may seem more sound, it raises several issues: To what extent can we characterize what "biologically plausible" actually means? If we fail to properly characterize plausibility, we run the risk of needlessly limiting the ability of the system to simulate interesting processes. Such failure is made all the more likely by the fact that our understanding of biological systems is far from complete.

Secondly, to what extent can a software system be designed so that its abilities coincide exactly with some characterization of biological plausibility? If either the abilities go beyond what is plausible, or else the system is unable to express or simulate one or more plausible processes, the system will not meet the requirement of enforced plausibility.

And finally, what is the cost in increased complexity of implementing such restrictions? This includes complexity of the software system itself, as well as complexity of the user's experience using the system.

The solution we have chosen is to limit the functionality of NeuroGene only when it is simple to do so, both implementationally, and as seen by the user. Biological plausibility is enforced primarily by the available set of primitives, since most primitives are justified by inferred or actual biological evidence that corresponding mechanisms exist in nature. Further restrictions are built into the data structure and execution model: Cell components can not communicate with each other except through chemical signaling and neural activity. Similarly, the state of the cell component is encoded in protein concentrations and timer values only.

However, in most cases we have not attempted to implement such limitations. Consequently, biologically implausible NeuroGene simulations are conceivable. The biological plausibility of NeuroGene simulations must therefore be justified by the researcher who designs the simulations and writes the genomes. This justification must cover simulated gene regulation and expression, including intracellular transport of proteins, simulated or implied signal transduction processes such as communication from the cell membrane to the nucleus (see figure 1.3), the invocation of actions implementing cellular changes, and the interaction between simulated genetic and neural processes. The justifications can take different forms, since the simulated processes may be known to occur in nature, or it may be inferred that a certain process must occur, or the process may merely be judged plausible, e.g., by reference to similar processes occurring in different systems. Certain types of mechanisms are so pervasive throughout biology that explicit justification may be considered superfluous, such as intracellular proteins affecting gene expression.

It is important to note that where, throughout the remainder of this thesis, I state the capabilities of NeuroGene in terms of being able to simulate genetic control of developmental processes, I assume that the simulation in question is biologically justifiable by the standards outlined here. The value of NeuroGene is this: Since NeuroGene simulated genomes are expressed in terms of gene expression, signal transduction and other biological mechanisms, it is possible to estimate whether a particular NeuroGene genome is justified, whether the corresponding simulation is biologically plausible, and consequently whether a given connectionist model might conceivably form part of the brain.

### 2.1.4 The design of NeuroGene

The basic requirements of NeuroGene are: 1. To represent genetic information (conjectured or based on experimental data) in a form which is human-readable and encoded in terms of biological concepts. 2. To simulate the developmental processes of neural tissues under the control of this genetic information. 3. To represent the biological mechanisms of gene regulation and developmental control as accurately as possible. 4. To simulate neural activity, including learning. These specifications have given rise to a software system consisting of the following concepts:

### Cells and cell components

Cells and cell components represent the biological cells that make up part of a developing organism. NeuroGene is designed to simulate the developmental process of such organisms. Cells are described in section 2.2.

### Growth cones

We simulate growth cones, which are central in axon guidance and synapse formation. Growth cones are transient structures, however, they have permanent effects on the simulation as they affect the growth of axons and dendrites. The NeuroGene model of growth cones is described in section 2.6.

### Genes

Genes as represented in NeuroGene contain logic which determines under which conditions each gene is expressed. They also contain specifications of the changes which are triggered when the genes are expressed. Genes are described in section 2.5.

Genes are executed in the context of a given cell component. During such an execution, it is determined whether or each gene is expressed in that cell component. The NeuroGene execution model is given in detail in section 2.10.

### Genome

Together the genes of an organism constitutes the organism's genome. In NeuroGene, the genome will also consists of other entities which together determine the behaviour of the cells, thereby determining the progress and final outcome of the simulations. These other elements are described throughout this chapter: timers (section 2.5.4) and receptor-ligand relations (section 2.4.4). Implementational details of the genome are outlined in section 2.8.1. The input and internal representation of genomes is described in section 3.1.

### Actions

Actions cause changes to cells under simulated genetic control. Actions result from the expression of simulated genes. NeuroGene contains a wide range of possible actions, and is designed to be relatively complete with respect to the range of cellular behaviours observed in development of biological organisms.

### Queries

Queries is a mechanism by which the properties of the current state of the cell component and its environment is made accessible to genes, which again allows genes to respond to these properties. Queries, like actions, cover a wide range of cellular mechanisms, and should ideally be able to account for all ways in which genes may directly or indirectly respond to cellular environments.

### Proteins

Proteins are produced by cells according to the "instructions" of the genome. To each gene corresponds one protein. A certain amount of this protein is produced by a given cell when the corresponding gene is expressed in that cell. Details about gene expression are given in section 2.5 and about proteins in section 2.4.

### Physical/chemical environment

The environment represents the space in which the cells exist. It represents the concentrations of proteins in the medium surrounding the cells (the *extracellular space*). As part of the environment is implemented functionality which simulates certain physical and chemical processes (as distinct from the biological processes controlled by the genome). These are protein diffusion and decay, as well as ligand-receptor binding, outlined in sections 2.4.1 and 2.4.4 respectively.

### Neural activity

The information processing abilities of neural networks need to be represented. For the sake of computational tractability, simple models of neural activity will be used, as outlined in section 2.7.

### Execution control

The simulation proceeds in discrete time steps. Each time step consists of a set sequence of sub-steps, including genome execution, protein diffusion and several others. This is described in detail is section 2.10.

## 2.2 Cells and morphology

Neurons are represented in NeuroGene in a way that supports much of the range of morphological variability seen in neurons in nature. In NeuroGene, a neuron consists of the central cell body (called the *soma*). From the soma extends zero or more ***dendrites*** and *axons*. These, as well as the soma, may carry synapses. A synapse is represented by a pair consisting of one ***presynapse*** and one ***postsynapse***. All cell components which make up a cell form a tree data-structure, of which the soma is at the root node. Subtrees which have an axon as root consist of axons and presynapses only. Subtrees which have a dendrite as root consist of dendrites and postsynapses only. Presynapses and postsynapses are always leaf-nodes, axons or dendrites may be leaf-nodes or internal nodes. Internal nodes, including the soma, may have any number of child-nodes. These relationships are shown in figure 2.1.

Biological cells have a high level of internal complexity, including sub-cellular bodies (called *organelles*) fulfilling a wide range of functions from energy conversion to photosynthesis, protein production and modification, etc. This complexity is completely left out of the NeuroGene simulator — the cell components have no internal structure except for intra-cellular protein concentrations.

Cells in nature respond to the direct physical interaction with other cells. Detecting such interactions in a dynamically changing computational representation, known as ***collision detection***, is computationally expensive. In optimizing NeuroGene simulator for computational speed, we have chosen not to carry out collision detection except when it is explicitly needed. When direct cell-cell interactions are needed, such as during synapse formation or axon guidance, the simulated growth cone mechanism (described in section 2.6) is capable of collision detection. This leaves chemical signaling as the means by which cells may detect the presence of surrounding cells, i.e., a cell may respond to the presence of proteins carried on the surfaces of surrounding cells, rather than the presence of the cells themselves.

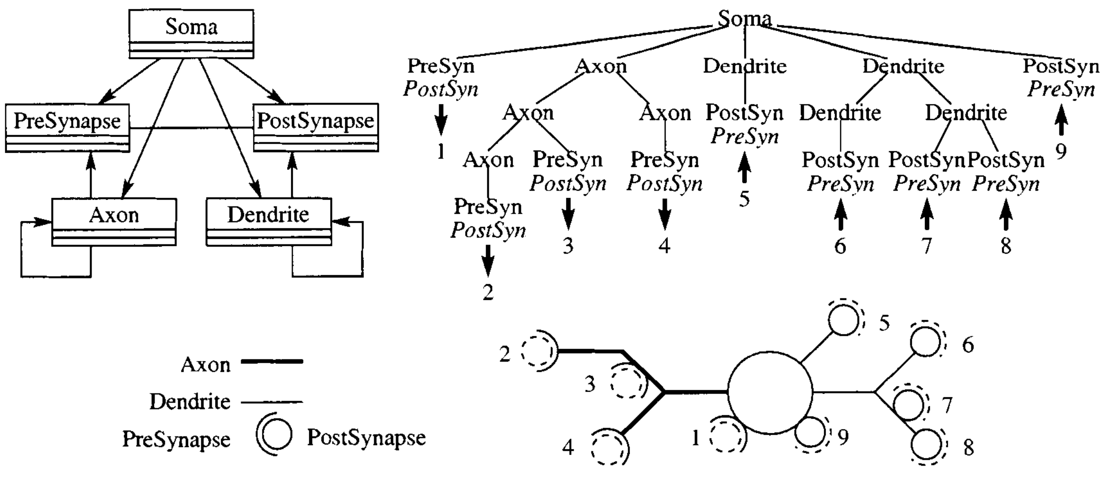


Figure 2.1: UML diagram (Liberty 1998, left) showing the possible forms of interactions between the classes used to represent neurons. Arrows indicate "one-to-many" relationships, e.g., a soma may be associated with many axons, but each axon is associated with one soma only. Lines indicate "one-to-one" relationships; each presynapse is associated with one postsynapse, and *vice versa*. Right: an example of a data structure created according to the UML diagram. The tree-topology of the data structure is evident. Arrows indicates the direction of neural activity flowing across synapses. Below: The cell which is represented by the example data structure. Synapses are numbered according to the scheme above. Synaptic termini belonging to other cells than the one shown are displayed in italics (top right) and broken lines (bottom).

### 2.2.1 Cell naming

When simulated neurons are created either during simulation initialization, or during cell division, cells may be given names. Cells' names, which need not be unique, do not change during the existence of the cells. Cell naming allows subsets of cells within a simulation to be specified using regular expression matching, see appendix C. Assuming that cells within a simulation are named according to some sensible scheme, a subset of cells can be defined as those cells whose names match a given regular expression. Several display modules that form part of the NeuroGene graphical user interface (GUI, outlined in chapter 3) are used to visualize different aspects of simulations. This technique is used to allow the user to specify which cells are to be visualized. Many examples of images created using these modules are shown in chapter 5.

When a cell divides, the parent cell ceases to exist, and both daughter cells may be given arbitrary names. If names are not specified, the daughter cells are given names which are derived from the name of the parent cell. The form of this derivation makes it possible to trace the ancestry of any given cell (see appendix C).

### 2.2.2 Actions

NeuroGene is designed to simulate developmental changes of cellular structures. These changes are under the control of genetic information which is supplied by the user in the form of a genome. Listed below are the primitives (actions and queries) which may be used to alter cell morphology. Each of these applies to a single cell or cell component. The format used to show each primitive is similar to that used in structured programming languages to show the signature of functions:

returntype **name**(parameterlist)

where *parameterlist* is a comma-separated list of zero or more parameters, each of which has a data type, *name* is the unique name of the primitive, and *returntype* is the data type of the value returned by the primitive, e.g., some primitives return numbers, others return lists of cell components, etc. The different data types which may be used in the genome scripts are shown in table 2.1 . Actions always have **void** as their return type[2].

These actions will be used, along with other primitives given in the remainder of this chapter, to specify the genome in the form of text file. The form of this text file, and the internal representation of the data contained in the file is given in detail in chapter 3. The actions relating to the cellular structure are:

| | | |
|---|---|---|
| void ***migrate***(direction dir) | cell ***branch***() | cell ***growAxon***() |
| cell ***divideCell***(direction dir) | void ***die***() | cell ***growDendrite***() |
| cell ***divideCell***(direction dir, string name1, string name2) | | |

Note that action names are printed in ***bold italics***, while query names are printed in **bold**. The action ***migrate*** causes a soma to move the distance specified by the value of dir, or, in

[2]Exceptions are actions used to build new cells, axons and dendrites — these actions return references to the new cell components they create.

| Type | Referent |
|---|---|
| number | A real signed number, double precision. |
| direction | A vector in 3D space, representing displacements or locations. |
| string | An 1D array of characters. |
| boolean | True or false. |
| void | No value, indicates that the primitive does not return any value. |
| protein | The name of a protein or the gene which expresses that protein. |
| growthconeFunction | The name of a growth cone function. |
| timer | The name of a cellular timer. |
| cell | A cell component. |
| cellList | A list of cell components. |

Table 2.1: Data types recognized within NeuroGene. The void data type is used to indicate functions that do not return a value. Attempts to use the return value from void functions cause a parser error when the gene is read by NeuroGene. For example, the code snippet `variable x = 3.4 * arborSize()` is valid, since the primitive **arborSize**() is a query with a return type number (see below) and 3.4 times a number is meaningful. However the code snippet `variable x = 3.4 * die()` is not, since the primitive ***die***() is an action and has return type void (see below), indicating that it does not return any value, and 3.4 times "no value" is meaningless and therefore invalid.

the case of axons and dendrites, to grow the same distance. Two forms of the ***divideCell*** operation cause a soma to divide, either specifying the names of the daughter cells, or using default names as outlined above. ***branch*** causes axons or dendrites to branch; ***growAxon*** and ***growDendrite*** cause a soma to sprout a new axon or dendrite; ***die*** causes an axon or dendrite to retract and disappear, or, if the operation is applied to a soma, the cell as a whole to be destroyed.

The following actions are used to help in debugging genomes:

| | |
|---|---|
| void ***red***(number r) | void ***green***(number g) |
| void ***blue***(number b) | void ***print***(string s) |
| void ***colour***(number red, number green, number blue) | |

***red***, ***green*** and ***blue*** alter the colour with which the cell is drawn in the NeuroGene GUI; ***colour*** changes all three colour channels at once. The colour of cell components has no biological significance, but may be useful in giving feedback to the user about the simulated genetic activity and may be used as a debugging tool when designing simulations. ***print*** causes a string to be printed to standard-out.

### 2.2.3 Queries

The following queries relate to cell morphology:

| | | |
|---|---|---|
| cell **parent**() | cell **partner**() | |
| cellList **allAxons**()[1] | number **numberOfAxons**()[1] | boolean **insideAxon**()[1] |
| cellList **allChildren**() | number **distanceToSoma**() | number **length**() |
| number **arborSize**() | number **cellSize**() | string **cellName**() |

[1] Other queries exist for each of the other cell component types.

The query **parent** when evaluated returns a reference to the parent cell component. A cell component's "parent" here has the usual meaning in the context of tree data structures, i.e., the cell component directly above in figure 2.1 top right. This reference can be used with other actions or queries which operate on other cell components. There are several of these listed in the following sections, examples are actions which cause protein to be transported between connected cell components, etc. **partner** is only valid for synapses, and it returns a reference to the opposing synaptic terminal, i.e., from a presynapse, the query returns a reference to the corresponding postsynapse and *vice versa.* **allAxons** returns a list of all the axons belonging to some soma. The language contains a special **foreach** loop construct with which one can iterate over such a list, either to query each element of the list, or to execute actions on them. Similar queries **allDendrites**, **allPreSynapses**, etc. are also available, one for each type of cell component. **allChildren** returns a list which is the union of all these cell component type-specific lists, containing all the cell components of which the current cell component is parent. **numberOfAxons** return the number of axons which have the current cell component as parent, i.e., the length of the list **allAxons**; **insideAxon** returns true if the current cell component is an axon, false otherwise. Also for these queries there exists one version for each cell component type (**numberOfDendrites**, **allPreSynapses**, **insideSoma**, etc).

**length** returns the length of an axon or dendrite (see section 2.10.8 for units of measurements). **distanceToSoma** returns the sum of the lengths of all dendrites from the current cell component to the soma. **cellSize** returns a measure of the total size of the whole cell (the soma contributes 1.0, the axons and dendrites contribute the values of their respective **lengths**, and synapses contribute 0.01). **cellName** returns the name of the cell.

## 2.3 Simulation space

The simulation space in NeuroGene is represented by a three-dimensional array of small cubic elements, all of the same size. These elements are called WorldNodes. The simulation space as a whole is referred to as the World, since the NeuroGene system does not represent anything that is outside this space. Each WorldNode contains zero or more cell components. This data structure is described in more detail in chapter 3.

### 2.3.1 Queries

The following queries give information about how cell components are arranged within the simulation space.

| | |
|---|---|
| direction **towardsCellDensity**() | number **cellDensity**() |
| direction **awayFromCellDensity**() | number **cellXCoordinate**()[2] |

[2] Other queries exist for the Y and Z dimensions.

**towardsCellDensity** returns a vector which is pointing in the direction in which the cells are the most densely packed. **awayFromCellDensity** returns the opposite vector, pointing toward where the cells are the most sparse. **cellDensity** returns a number representing how many cell components are close to the querying cell component. These queries return values that are computed relative to a particular point in the simulation space — this point is always the location of the cell component that performs the query. Since collision detection is not implemented, these queries can be used to implement *contact inhibition*, the process by which cells' behaviour changes when they come into direct physical contact with other cells (Browder et al., 1991, p. 366).

**cellXCoordinate** returns the x coordinate of the cell component. Similar queries **cell-YCoordinate** and **cellZCoordinate** return the other coordinates.

## 2.4 Proteins

In nature, proteins play central roles in maintaining the cell's structure and shape, and are also involved in all ongoing processes within the cell, including regulation of gene expression. Proteins make up the cytoskeleton which determines the cell's shape. Ion channels, which determine cells neural properties, are proteins, as are receptors which detect chemical signals in the cell's environment, etc. In NeuroGene, proteins' roles in neural activity, morphological change, etc., has been abstracted away. This abstraction is necessary because mechanisms by which changes in proteins concentrations affect changes in neural activity patterns, cell migration or other cellular properties are too complex to model explicitly. Instead, genes in NeuroGene may affect such changes directly through actions. Consequently, the function of proteins within NeuroGene is primarily the regulation and coordination of gene expression. Proteins are also useful as they can be visualized within the NeuroGene GUI, thus allowing the user to monitor gene expression during simulations.

In NeuroGene, proteins are divided into two classes, *soluble* (or diffusible) and *membrane bound*. Membrane bound proteins are attached to some membrane that is part of a cell. This may be the cell membrane surrounding the cell, or it may be other membranes that are part of the cell's interior structure, but which are not explicitly represented in NeuroGene. Proteins that are bound to the cell membrane are exposed to the space outside the cell, and may interact with other cells. Proteins that are bound to intracellular membranes are not exposed, and are "invisible" to other cells. Membrane bound proteins remain bound to the cells in which they were produced.

Soluble proteins are those that are not bound to membranes. These proteins are free to diffuse away from their source, much like cigarette smoke spreads through a room. Cell membranes are impermeable to these proteins.

Protein concentrations are represented for three distinct types of locations: The first is the extracellular space, which surrounds all cells in the simulation. Only soluble proteins can be present in this space, since there are no membranes for membrane bound proteins to attach to. Soluble proteins are free to diffuse through this space. The second is the surfaces of cell components. Only membrane bound proteins can bind to cell surfaces. These proteins remain bound to the cell component, and if the cell component migrates through the simulation space, the proteins move with them. The last is the interior of cell components.

The interior may contain both membrane bound and soluble proteins — the membrane bound proteins are imagined bound to intracellular membrane structures, of which there are many in nature, but which are not represented in NeuroGene. Protein may be transported between adjacent cell components using certain actions for internal transport, these will be outlined below.

Extracellular proteins and proteins bound to cell surfaces can be perceived by other cell components, while intracellular proteins cannot. All soluble and membrane bound proteins are subject to decay according to protein-specific first-order decay rates, regardless of their location, while only soluble protein in the extracellular space are subject to diffusion.

### 2.4.1 Protein decay and diffusion

Diffusion and decay are two simple chemical processes which play central roles in how protein concentrations evolve over time. Decay applies both to membrane-bound and soluble proteins. It is a process by which the protein molecules are gradually broken down and disappear. The rate of decay is characterized by the *decay rate* $k$, a non-negative real number, defined according to the equation of first-order decay (Atkins and de Paula, 2002)

$$\frac{dC}{dt} = -kC \tag{2.1}$$

where $C$ is the concentration of the protein.

Diffusion does not apply to membrane bound proteins which remain bound to the cell components that initially produced them. Diffusion within cell components, which does occur in nature, is also ignored in NeuroGene. Soluble proteins in the extracellular space tend to spread through this space, away from the cell components that produce them. The rate of this spread is characterized by the diffusion rate. The diffusion in 1D is governed by the equation (Kreyszig, 1988, p. 661)[3]

$$\frac{\partial C}{\partial t} = D\frac{\partial^2 C}{\partial x^2} \tag{2.2}$$

where $C$ is the concentration of the protein, and $x$ is the one spatial dimension and $D$ is the diffusion rate, again a non-negative real value. The form of this equation can be rationalized as shown in figure 2.2A. In 3D the equation becomes

[3]This equation describes diffusion, heat flow and electrostatic potentials, and it is introduced in most general textbooks on differential equations.

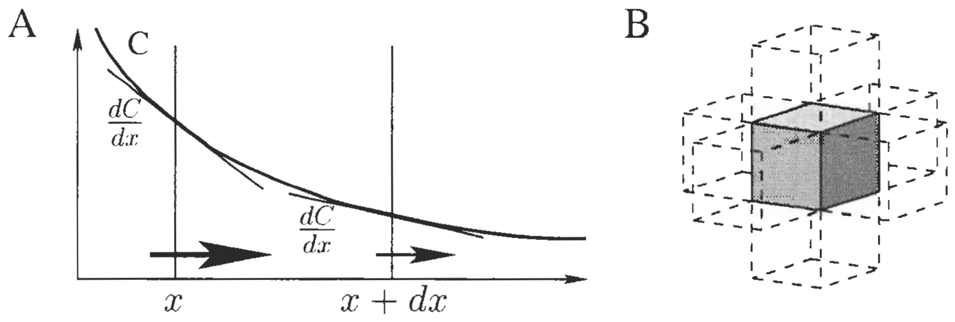


Figure 2.2: A: Informal graphical derivation of the diffusion equation. The net flow of protein at any point is proportional to the slope $dC/dx$ of the concentration $C$ with respect to the spatial coordinate $x$, the coefficient of proportionality being the rate of diffusion $D$. The flow across the line at $x$ will therefore be larger (since the slope of $C$ is greater) than that across the line at $x + dx$, as indicated by the sizes of the two arrows. The net change in concentration $C$ in the region between $x$ and $x + dx$ is the difference between the flow across the boundary at $x$ and the flow across the boundary at $x + dx$, which is then proportional to the difference between the slope at $x$ and the slope at $x + dx$. When $dx$ is very small this difference is equal to the curvature $d^2C/dx^2$. B: Arrangement of the six face-sharing WorldNodes.

$$\frac{\partial C}{\partial t} = D\nabla^2 C = D\left[\frac{\partial^2 C}{\partial x^2} + \frac{\partial^2 C}{\partial y^2} + \frac{\partial^2 C}{\partial z^2}\right] \tag{2.3}$$

In order to simulate diffusion within the NeuroGene simulation space, a numerical solution of this equation is needed. This solution makes the following assumptions: Within a single time step, diffusion only occurs between face-sharing WorldNodes, see figure 2.2B. It is also assumed that the rate of diffusion is relatively slow, specifically that the numerical value of $D$ is less than $\frac{1}{7}$ in whatever units of time and concentration are used (see section 2.10.8)[4]. This gives rise to the following simple expression of the concentration $C^0_{t+1}$ in some WorldNode 0 at time $t + 1$(Cruise, 2001):

$$C^0_{t+1} = C^0_t + D\sum_{n=1}^{6}\left(C^n_t - C^0_t\right) \tag{2.4}$$

[4]Consider equation 2.4 in the situation $C^0_t = 1$, $C^n_t = 0$ for all $n \neq 0$. For $D > \frac{1}{6}$, $C^0_{t+1}$ becomes negative. For $D > \frac{1}{7}$, $C^0_{t+1} < C^n_{t+1}$ for all $n \neq 0$, which means that more has diffused out of the central node than what remains. Both of these situations are physically impossible. Consequently the algorithm is only stable for slow diffusion for which $D < \frac{1}{7}$.

$C_t^n$ for $1 \leq n \leq 6$ is the concentrations of the proteins in the six face-sharing WorldNodes.[5]

Special cases arise when the WorldNode 0 is at the edge of the simulation space. Here, one or more of the six face-sharing nodes will be missing, since these would represent points in space which fall outside the simulation space. Two different solutions to this problem has been implemented in NeuroGene. In the first, which is called *open*, the concentrations $C_i^n$ of such missing WorldNodes are set to zero, in the other (*closed*) case, these concentrations are set equal to $C_i^0$. In the open case, the simulation space is effectively surrounded by an infinitely large volume where the concentrations of all proteins is zero. This means that proteins are free to diffuse out of the simulation space, hence the designation *open*. In the closed case, the simulation space is effectively enclosed in a barrier which is impermeable to the diffusing protein.

### 2.4.2 Queries

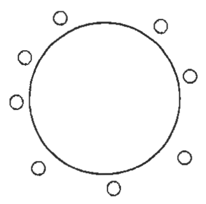
Essential to the simulation of genetic regulation in NeuroGene is genes' ability to sense the concentrations of proteins within cells as well as in cells' external environment. The following queries can be used to retrieve current concentrations of proteins, or the slopes of concentration gradients, within or in the vicinity of cell components:

| | |
|---|---|
| number **internalConcentrationOf**(protein s) | direction **towardsSignal**(protein s) |
| number **surfaceConcentrationOf**(protein s) | direction **awayFromSignal**(protein s) |
| number **externalConcentrationOf**(protein s) | |

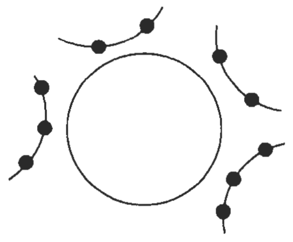
Figure 2.3 shows the definitions of various actions and queries. It is important to note that many of the actions and queries have different meaning when they are applied to soluble and membrane-bound proteins. When the protein $s$ is soluble, the queries from some cell component $C$ return the following values:

- **externalConcentrationOf** returns the concentration of the protein in the WorldNodes computed for the location of the cell component $C$ (figure 2.3A). This computation

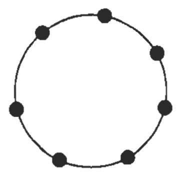
[5] A more stable algorithm exists, known as DuFort-Frankel, which can handle faster diffusion rates. Computing the concentrations at time $t+1$ involves the concentrations at both times $t-1$ and $t$. This approach can not deal with changes in concentrations that are due to other processes than diffusion, such as protein expression, and it is therefore not suited to our needs.



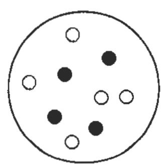
A: The external concentration of soluble protein is the concentration of protein dissolved in the extracellular fluid. This concentration changes through external expression or consumption of soluble proteins.



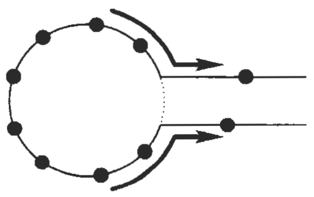
B: The external concentration of membrane bound protein is the concentration of protein bound to the surface of nearby cell components.



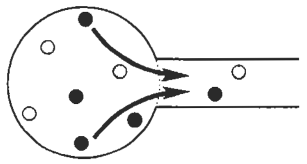
C: The surface concentration of protein. This applies to membrane bound proteins only. This concentration changes through external expression or consumption of proteins.



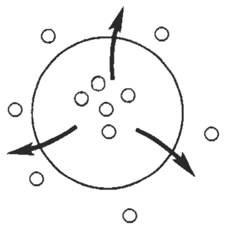
D: The intracellular concentration of protein. Both membrane bound and soluble proteins may be carried intracellularly. These concentrations change through internal expression or consumption of proteins.



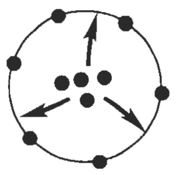
E: Export of membrane bound proteins along the cell surface between cell components belonging to the same cell. Import causes transport in the opposite direction, i.e., towards the cell component which executes the action.



F: Intracellular export of membrane bound or soluble protein to a neighbouring cell component. An import action causes the transport to occur in the opposite directon.



G: Exocytosis of soluble protein from the interior of the cell component to the extracellular space. Endocytosis represents the transport of the protein in the opposite direction.



H: Exocytose of membrane bound protein from the intracellular space to the cell membrane. Endocytosis causes transport in the opposite direction.

Figure 2.3: Definitions of protein concentrations and mechanisms of protein transport. Soluble protein shown as ○, membrane bound proteins as ●.

involves interpolation between the values of several WorldNodes close to this location. The implementation of this interpolation is given in chapter 3.

- **surfaceConcentrationOf** always returns zero, since soluble proteins do not associate with the cell surface (figure 2.3C).
- **internalConcentrationOf** returns the intracellular concentration of the protein in the cell component$C$ (figure 2.3D).
- **towardsSignal** returns a vector pointing in the direction in which the extracellular concentration (defined as for **externalConcentrationOf**) increases the most. Again interpolation is used.

When the protein is membrane bound, the same queries compute the following values:

- **externalConcentrationOf** computes the amount of the membrane bound protein bound to cell components in the vicinity of the cell component $C$ (figure 2.3B). For each WorldNode close to the location of $C$, the sum of surface concentrations of the protein for all cell components contained in that WorldNode is computed. These values are then used to interpolate the value at the location of $C$.
- **surfaceConcentrationOf** returns the surface concentration of the protein in the cell component $C$ (figure 2.3C).
- **internalConcentrationOf** returns the intracellular concentration of the protein in the cell component$C$ (figure 2.3D).
- **towardsSignal** returns a vector pointing in the direction in which the extracellular concentration (defined as for **externalConcentrationOf**) increases the most. Again interpolation is used.

**awayFromSignal** returns the vector which is the inversion of that returned by **towardsSignal**.

### 2.4.3 Actions

The wide range of biological mechanisms that involve proteins requires a wider range of actions which manipulate protein concentrations values in ways that reflect commonly observed biological phenomena. This involves consumption of protein, as well as transport of protein under genetic control. Protein production results from gene expression, and is described in section 2.5 below. The following actions are used to manipulate protein concentrations within cell components or their immediate environment.

```
void consumeExternally(protein s, number r)
void consumeInternally(protein s, number r)
void exportInternally(cell c, protein s, number r)
void exportSurface(cell c, protein s, number r)
void importInternally(cell c, protein s, number r)
void importSurface(cell c, protein s, number r)
void endocytose(protein s, number r)
void exocytose(protein s, number r)
```

These actions all take as a parameter the name of the protein whose concentration will be affected by the action, as well as a number $r$, indicating the amount of change in concentration which will be caused by the action. The value for $r$ is specified as part of the gene program, and can be either a constant value or an expression composed of queries, mathematical functions (such as square root or logarithm) and constants.

The external concentration of soluble proteins (figure 2.3A), the surface of membrane bound proteins (figure 2.3C) and internal concentrations of both types of protein (figure 2.3D) are all reduced by the appropriate *consume* primitives. The following actions also affect one or more of these concentrations: Transport of protein between neighbouring cell components that form part of the same cell is done using the *export* and *import* actions (figure 2.3E–F). Transport between the intracellular space and the cell surface or the extracellular space is done using the *endocytose* and *exocytose* commands (figure 2.3G–H).

For all these actions, it is assumed that the value given for the parameter $r$ is greater or equal to zero. For each primitive, if $r$ evaluates to a negative value, the action does not occur. We have made this design decision in order for the genetic encoding to be as clear as possible. We hereby avoid the potentially confusing situation where a statement like `export(NT,x)` might actually represent an **import** action if `x` happened to be negative.

### 2.4.4 Ligand binding

Receptor-ligand binding is a chemical process on which many important developmental processed depend, particularly those which allow cells to respond to their environment. The mechanism usually involves two different proteins, one which is called the *receptor*, which is bound to the surface of the cell. The other is called the *ligand*, and it may either be a soluble protein, or else bound to the surface of another cell. The receptor has a *binding site*, facing the outside of the cell, in which the ligand fits the way a key fits in a lock. When the ligand binds to the receptor, this causes changes in the receptor, which in turn causes signaling events inside the cell, thereby communicating the extracellular presence of the ligand to the interior of the cell (see figure 1.3).

While in nature, the ligand proteins may be either soluble or membrane-bound, in NeuroGene ligands must be soluble. As implemented in NeuroGene, competitive ligand-receptor binding can be used to model a wide range of biological phenomena, however, processes involving membrane-bound ligands can not be simulated.

The chemical reaction in which one molecule of ligand $L$ binds to one molecule of receptor $R$ to give one molecule of complex $C$ is described by the following equilibrium

$$C \rightleftharpoons R + L \tag{2.5}$$

The double arrow $\rightleftharpoons$ indicates that the reaction is reversible, i.e., once $L$ binds to $R$ it may fall off again. The strength of binding, or the relative tendency for $L$ to bind to and to fall off $R$, is described by the equilibrium constant $K_d$, given by the equation (Atkins and de Paula, 2002)

$$K_d = \frac{[R][L]}{[C]} \tag{2.6}$$

where the square brackets indicate "concentration of". The smaller the value of $K_d$, the more tightly the ligand binds to the receptor. Given some arbitrary initial concentrations $[L]_0$, $[R]_0$ and $[C]_0$ of the three chemical species, the equilibrium concentrations can be computed from the equation

$$K_d = \frac{([R]_0 + \Delta)([L]_0 + \Delta)}{[C]_0 - \Delta} \tag{2.7}$$

where $\Delta$ may be positive or negative. Solving for $\Delta$, we get

$$\Delta^2 + \Delta([R]_0 + [L]_0 + K_d) + [R]_0[L]_0 - K_d[C]_0 = 0 \tag{2.8}$$

**Algorithm 1** Simulation of ligand-receptor binding for one WorldNode instance $W$.

```
1: $[L]_0 \leftarrow$ the extracellular concentration of ligand in the WorldNode $W$
2: $[R]_0 \leftarrow$ the total surface concentration of receptor on all cells in the WorldNode $W$
3: $[C]_0 \leftarrow$ the total surface concentration of complex on all cells in the WorldNode $W$
4: $B \leftarrow K_d + [L]_0 + [R]_0$
5: $C \leftarrow [L]_0[R]_0 - K_d[C]_0$
6: $\Delta \leftarrow (-B + \sqrt{B^2 - 4C})/2$
7: set extracellular concentration of ligand in the WorldNode $W$ to $(L_0 + \Delta)$
8: $f_R \leftarrow ([R]_0 + \Delta)/([R]_0 + [C]_0)$
9: for all cells $c$ in the WorldNode $W$ do
10:     $\beta_c \leftarrow$ total concentration of receptor and complex carried by cell $c$
11:     set surface concentration of receptor on cell $c$ to $\beta_c f_R$
12:     set surface concentration of complex on cell $c$ to $\beta_c(1 - f_R)$
13: end for
```

Since $[R]_0$, $[L]_0$, $[C]_0$ and $K_d$ are all non-negative, and the equilibrium concentrations also must all be non-negative, the correct solution of the quadratic equation can be determined *a priori*: With two solutions $\Delta_1$ and $\Delta_2$ such that $\Delta_1 \leq \Delta_2$, both the terms $([R]_0 + \Delta_1)$ and $([L]_0 + \Delta_1)$ in equation 2.7 are negative, giving a mathematically valid but physically invalid solution. With the solution $\Delta_2$, both these terms are positive, and the solution is valid.

The algorithm used to compute the equilibrium concentrations of receptor, ligand and complex for a set of cell components within a single WorldNode instance is shown in algorithm 1. In lines 1–3 the values $[L]_0$, $[R]_0$ and $[C]_0$ as defined above are computed. $[L]_0$ can be retrieved directly from the WorldNode $W$, $[R]_0$ and $[C]_0$ are the sums of contributions from each of the cell components in $W$. Lines 4–6 compute the value of $\Delta$ by solving the quadratic equation 2.8 above. With the value for $\Delta$, the equilibrium values for the concentrations of receptor, ligand and complex can be computed from equation 2.7, i.e., $[R]_{eq} = [R]_0 + \Delta$, $[L]_{eq} = [L]_0 + \Delta$ and $[C]_{eq} = [C]_0 - \Delta$. On line 7 the concentration of ligand as stored in the WorldNode is updated to this new value.

When updating each of the cell components in the WorldNode, the following two principles apply: The total concentration $\beta_c$ of receptor and complex for each cell component (see line 10) should remain unchanged, and the relative amount of receptor and complex should be the same for all cell components. $f_R$ is the fraction of the combined amount of receptor and complex which is in the free (i.e. receptor) form at equilibrium.

$$f_R = \frac{[R]_{eq}}{[R]_{eq} + [C]_{eq}} = \frac{[R]_0 + \Delta}{[R]_0 + [C]_0} \tag{2.9}$$

$f_R$ is computed on line 8. While iterating over all cell components in the WorldNode (line 9), the total amount $\beta_c$ of receptor and complex carried by the cell component is computed (line 10), and the relative amounts of each is adjusted according to the value of $f_R$ (lines 11–12).

The algorithm iterates over all cell components twice, once to compute the values in lines 2 and 3, and then to update the values in lines 9–13. The remaining steps of the algorithm (lines 1 and 4–8) run in constant time. The algorithm therefore has time complexity $O(n)$ where $n$ is the number of cell components in the WorldNode.

The algorithm makes two assumptions: The first is that the ligand-receptor binding reaction is at equilibrium. This approximation is valid if the binding reaction is fast relative to the duration of a simulation time step. The other assumption is that the total number of molecules involved is large, meaning that stochastic effects deriving from single molecular events do not play a significant role. The binding reactions we simulate usually occur within very small regions of space with low concentrations of ligand and receptor, meaning that the stochastic nature of interactions between individual molecules may be significant. However, the computed concentrations represent time-averages. If the duration of the simulation time step is sufficiently long, the computed values will be a correspondingly precise estimates of the true average concentrations. See section 2.10 for a discussion of the significance of the simulation time step duration.

The algorithm has the limitation that the same protein can not be involved in more than one receptor-ligand-complex relation. This limitation does not exist in nature, where e.g. the same ligand may be detected by several different receptor each with a different affinity. However, by excluding this possibility, NeuroGene is still able to simulate a wide range of competitive processes while keeping the mathematics tractable.

## 2.5 Genes and the genome

The concensus among biologists is that the vast majority of cellular behaviour is governed by genetic mechanisms. NeuroGene has been designed in such a way that all cellular behaviour be governed by simulated genes. This requires that simulated genes be capable of representing causal relationships such as those described in the quote on page 6 and figure 1.2. Furthermore, the effect of neural activity on gene expression must also be representable. Finally, the effect of gene expression on neurons, including their growth and morphology, neural properties and biochemistry must also be representable. This section outlines the NeuroGene genetic system which makes this possible. The equally important genetic roles in mutation and inheritance do not come into play in the developmental processes NeuroGene is designed to simulate, and are not supported.

Gene expression in nature causes the production of the protein encoded by the gene. Often the proteins are transported to specific target locations within the cell, such as dendrites or presynaptic termini. In NeuroGene, we need to be able to simulate such targeting, and we should be able to do so without having to simulate the details of intracellular transport. The solution is to have all genes being active in *each* cell component, i.e., all parts of the cell are "created equal" under the NeuroGene gene mechanism. For genes to affect, for example, synapses, all that is required is to specify as part of the gene's expression behaviour that it is "expressed in the synapse". In nature this would signify that protein produced from the gene is transported to synapses through complex intracellular transport mechanisms. In NeuroGene, the gene's expression rate is zero in all cell component types except synapses. The targeting of proteins to the intracellular space, the cell membrane, or the extracellular space is similarly simplified with respect to the biological model, with such targeting also explicitly included as part of the genes' regulation behaviour.

### 2.5.1 Genes

The gene data structure is used within NeuroGene to represent biological genes. By virtue of being an abstraction, it can also represent multiple genes or biological entities or processes which are not genes, as outlined above (section 2.1.1). The data structure maintains the information needed to model the causal relationships which govern the expression and cellular effects of biological genes, including some select properties of the protein which is produced when the gene is expressed. The genes have the following properties:

- A name which identifies both the gene and the protein produced when the gene is expressed.
- The gene's expression behaviour:
  - A symbolic representation of the gene's expression rate as a function of the state of the cell component. This is evaluated to give both an expression rate (a real non-negative number) and an expression location (either INTERNAL or EXTERNAL).
  - The cell component types in which the gene is expressed. This may be one or more of the following values: SOMA, AXON, DENDRITE, PRE_SYNAPSE and POST_SYNAPSE.
- Properties of the protein produced by the gene:
  - Whether the protein is soluble or membrane-bound.
  - The decay rate, which determines how quickly the protein is broken down after it is produced (see section 2.4.1).
  - Diffusion rate, which determines how fast a soluble protein spreads through the simulation space away from the producing cells. Membrane-bound proteins do not have a diffusion rate.
- The *effects* of gene expression, i.e., the actions which result from the expression of the gene. One implicit effect is always the addition of a given amount of protein to the appropriate expression location, where both the expression rate and location have been computed by the gene regulation. Additional effects may also be specified as part of the gene definition. These may include actions which change the cell's morphology (grow axons, form synapses, migration through the simulation space, etc.), or any of the other actions which will be described throughout the remainder of this chapter.

See appendix B for numerous examples of NeuroGene genomes. The details of the internal representation of the regulation and effects portions of genes will be described in chapter

**Algorithm 2** Simulated gene expression.

```
1: evaluate the gene's expression behaviour in a given cell component to give the expression
   rate $R$ and protein target location. The expression rate is a real non-negative number.
   Target location is either $INTRACELLULAR$ or $EXTRACELLULAR$.
2: if $R > 0$ then
3:   if target location is $INTRACELLULAR$ then
4:     add an amount $R$ of protein to the intracellular space
5:   else if target location is $EXTRACELLULAR$ then
6:     if protein is membrane bound then
7:       add an amount $R$ of protein to the cell membrane
8:     else
9:       add an amount $R$ of protein to the extracellular space
10:    end if
11:  end if
12:  evaluate the gene's effects, entering the actions on the ActionQueue.
13: end if
```

3. The execution of a gene within a given cell component in a given simulation time step follows algorithm 2. Note that genes are stateless, in that none of their properties change during the simulated gene expression as outlined. This reflects genes in nature, which never change during the normal operations of cells.

The algorithm is run for each gene in each cell component each simulation time step. In line 1, it is determined from the gene's regulation behaviour what the expression rate of the gene is. Depending on the expression behaviour of the gene, this may depend on the type of cell component, the concentrations of various proteins in the cell components or in its environment, its neural state, etc. If the expression rate is greater than zero, the gene is expressed (line 2).

Depending on the target area of the protein and whether or not the protein is soluble or membrane-bound, the appropriate amount of the gene's protein will be added to the appropriate target area (lines 3–11). Finally the other effects of the gene are executed in line 12. The role of the ActionQueue will be explained in section 2.10.

### 2.5.2 Justification of the gene algorithm

The biological processes of gene expression is understood quite well in general (Smolen et al., 2000a), although the particular expression behaviours of many individual genes have yet to be elucidated. The algorithm described above attempts to capture three aspects of gene expression: The regulation of expression, which determines the rate of protein production, the transport of protein to the proper cellular target area, and the effects of gene expression on the structure and function of cells.

In the introduction, an outline was given of the stages of elucidation of the structure and function of a gene (section 1.2.6). It is important to note that the internal representation of genes within NeuroGene does not include all these aspects of gene and protein structure and function. However, this abstraction and consequent omission of detail in the representation of genes does not lead to loss of accuracy, since the omitted detail does not affect those aspects of gene regulation and developmental control which we are interested in. For instance, we represent the fact that some regulatory protein affects the expression of a particular gene (see question 2 on page 12), but we do not represent the molecular mechanism by which this is accomplished (question 5). Similarly, we represent the fact that a particular protein causes a particular change in the structure or function of a cell component (question 1), but again the molecular mechanism of this interaction is omitted (question 6). This allows the relevant aspects of the genetic information to be represented in symbolic form which is easily accessible to biologists.

### Gene regulation

The range of cellular properties which may influence gene expression have been outlined in section 2.1.1. The genetic programming language allows queries of these properties to be compiled into arbitrary mathematical expressions, which may contain conditionals. Such expressions are able to capture the kind of linear and non-linear phenomena of gene regulation that has been observed in nature, see e.g. Davidson et al. (2002), quoted on page 6. The expression rate is evaluated on line 1 of the algorithm. If this rate is greater than zero, the gene is expressed (line 2). Within this general outline defined by the algorithm, the particular encoding of the expression behaviour of any particular simulated gene has to be justified by its designers.

### Protein expression and sub cellular targeting

Evaluating the gene expression function in the context of a given cell component gives the rate of production of the corresponding protein within that cell component. At the same time the destination of the protein (intracellular or extracellular) is determined. This represents an abstraction with respect to protein production in nature, and conflates five distinct processes: 1. regulatory information flow to the cell nucleus, which in a simulated neuron corresponds to the soma (see figure 1.3), 2. gene transcription into mRNA, 3. transport of mRNA to the target area of the cell (i.e. back to the given cell component), 4. translation of mRNA into protein (figure 1.1), and finally 5. possible transport of the protein to the cell membrane or out of the cell.

The justification for this abstraction is two-fold, and follows a familiar pattern used throughout this thesis: In most cases the abstraction do not influence the outcome, while at the same time conferring significant gains in computational efficiency and ease of use. Secondly, in cases where the added detail needs to be included, this is possible within NeuroGene at the cost of a more complex simulation. There are actions available which can be used to simulate intracellular transport of mRNA and protein, and separate simulated genes may be used to independently simulate the transcription and translation of genes, see below.

### Gene effects

Simulated gene expression may cause changes to occur to the state of the cell component or its environment (line 12). In nature, it is not gene expression in itself, but rather the presence of protein resulting from gene expression, which causes such changes. Our approach correctly simulates genes whose expression and protein-mediated effects on cellular function have identical time course. Our model does not capture genes whose proteins persist and affect cell function after gene expression has ceased.

This limitation has been kept in mind when developing the simulations presented in this thesis. It is likely that a future version of the NeuroGene simulator will incorporate a more accurate model of gene expression in which the role of proteins in shaping cell structure and behaviour is more explicitly modeled.

### Modeling transcription and translation separately

The production of proteins in nature is regulated both at transcription (production of an mRNA copy of the DNA gene) and at translation (production of protein based on the information in the mRNA). Such independent regulation can be modeled in NeuroGene by representing the gene by two simulated genes. The first gene represents the regulation at transcription, and the product of that gene represents the gene's mRNA copy. The second simulated gene represents translation, the product of which is the protein. The expression of the second gene requires the presence of the mRNA produced by the first gene, thereby allowing the regulation of the first gene to indirectly affect the regulation of the second. This technique will be used in one of the simulations presented in chapter 5.

### Modeling post-translational modification

In nature, many proteins are produced in an inactive form, and require activation through some form of chemical change, usually involving enzymes. One common example is phosphorylation, in which an enzyme attaches a phosphate group ($-PO_4^{2-}$) to the inactive form of a protein to activate it. Such processes can also be modeled up to a point using simulated pseudo-genes. One gene produces the inactive form of the protein, and a simulated pseudo-gene which represents the activation reaction, produces the active form. For the active form to be produced, the presence of the inactive form and the catalyzing enzyme are both required. This is simply captured in the gene regulation of the pseudo-gene. However, in nature the production of the activated form of the protein causes a depletion in the level of the inactive form. While this can be captured if necessary using the **consume**() actions (see section 2.4.3), it is not as straightforward. Alternative mRNA splicing, a process by which a single gene may give rise to multiple different proteins, is also not supported.

## 2.5.3 Actions and queries

The actions listed in the box below are those used to specify the expression rate of a gene. A number of different actions are given, each specifying a different target location for the expressed protein. Each action takes a numeric parameter *rate*, which is the expression rate. Like all parameters of type number, the expression rate may be a constant value or an expression involving queries. Only if the computed rate is greater than zero is the gene expressed.

| | |
|---|---|
| void ***expressInternally***(number rate) | number **geneExpressionRate**() |
| void ***expressExternally***(number rate) | |
| void ***runOneGeneOnly***(protein p) | void ***dontRunGenes***() |

Unlike most other actions, these have the semantics of "return"-statements in languages such as C, C++ and Java, i.e., no actions following one of these actions are executed. ***expressInternally*** causes a given amount of protein to be added to the interior of the cell component; ***expressExternally*** adds the protein to the cell surface (if the protein is membrane bound) or to the extracellular space (if the protein is soluble).

The action ***dontRunGenes***, when executed in the context of some cell component, ensures that the genome will not be executed in the context of that cell component. This can be used to improve the speed of simulations. ***runOneGeneOnly*** similarly causes only one gene to be executed in a given cell component, namely the gene which encodes the protein p.

### 2.5.4 Timers

Some genes are known which have expression behaviours with complex time dependence. Davidson et al. (2002) report that the expression of the gene *foxa* in sea urchin oscillates, driven by persistent positive inputs from other genes combined with auto-repression. In some cases such oscillating gene interaction networks have been created from scratch (Elowitz and Leibler, 2000; Judd et al., 2000). Smolen et al. (2000b) is a review article on the simulation of oscillating and other dynamic gene regulatory systems. While the mechanisms driving many such systems may be well understood, encoding such behaviour in a NeuroGene genome is greatly simplified by the existence of a timer primitive. Zero or more named timers may be defined as part of a genome, and the expression of genes may depend on the values of these timers. Each cell component has its own value of each timer.

| | |
|---|---|
| void ***setTimer***(timer id, number ticks) | number **timerValue**(timer) |

The action ***setTimer*** allows a named timer to be set to some positive value. Each time step, all non-zero timers in all cell components are decremented. The query **timerValue** may be used to query a given cell component for the current value of some named timer.

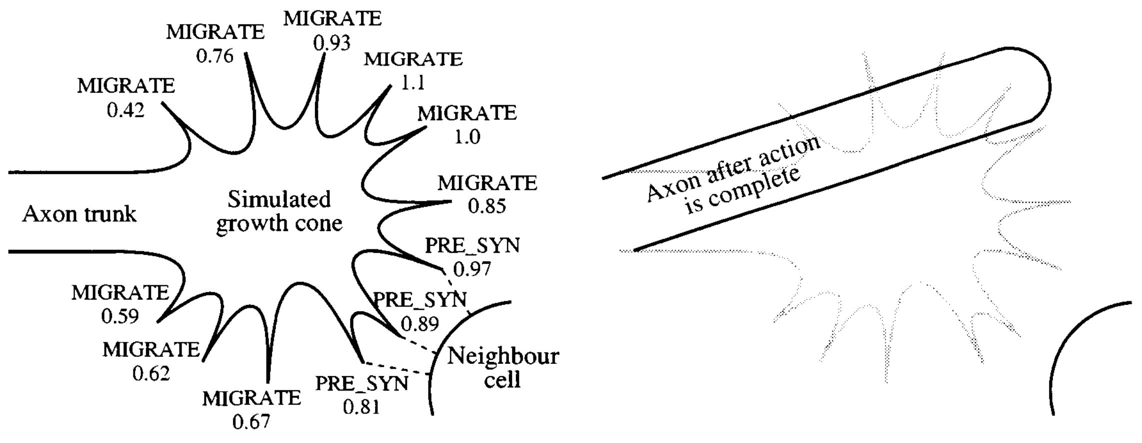


Figure 2.4: Example showing the auction mechanism of a growth cone consisting of twelve filopodia. The bid value and action is shown for each filopodium. These have been chosen arbitrarily for the purpose of illustration. The winning filopodium has a bid of 1.1, and causes a MIGRATE action to give the situation on the right.

## 2.6 Growth cones

A growth cone is a bulbous structure on the tip of an extending axon or dendrite. From this dynamic structure extends numerous fine fibres known as filopodia, which appear to sense and probe their environment, and thereby guide the advancement of the growth cone. Growth cones are instrumental in wiring up the nervous system, from navigating long distances between tissues to forming specific connections based on chemical cues. The details of the mechanisms by which the growth cone accomplishes this process are not well understood. We have tried to capture the general outline of the process using an auction paradigm among filopodia, see figure 2.4. In the auction process, all the filopodia that make up one growth cone compete through a bidding process. The winner of the auction is the filopodium with the highest bid value. The action resulting from the growth cone process is determined by the winning filopodium alone, and the location of that filopodium is used in executing the action. For example, the action may be to MIGRATE, in which case the axon migrates to the location of the tip of the winning filopodium. The other possible action is to form synapses with other, nearby cell components.

In order to form synapses with other cell components, filopodia are able to detect the presence of other cell components in their environment. They are also able to sense the surface properties of these cell components, including the concentration of membrane bound proteins, which allows for simulation of specific synaptogenesis processes. This is the only mechanism by which collision detection is available in NeuroGene.

Each filopodium computes its bid value and action based on a *growth cone function*, which is defined by the user. The bid value may be computed based on all of the information perceived by the filopodium, and as such, may define a wide range of behavioural patterns of the growth cone. A genome may contain zero or more growth cone functions, each identified by a unique name.

## 2.6.1 Invocation of growth cone mechanism

Growth cones in nature form in response to genetic activity. In NeuroGene, this means that for an axon or dendrite to form a growth cone, a gene which is expressed in that axon must cause the growth cone to form, i.e., that the effect of gene expression (see line 12 of algorithm 2) is the formation of a growth cone governed by a particular growth cone function. Such an action is encoded in the genome using this primitive:

void ***growthcone***(growthconeFunction gcf)

Note that this action is not added to the queue of pending changes. Instead, the growth cone is constructed immediately. The action resulting from the growth cone (such as cell migration) is then added to the queue, as will be explained in the section on the ActionQueue below. ***growthcone***() is arguably a meta-action, since it is evaluated within a cell component to give an action, which is subsequently applied to the cell component.

## 2.6.2 Growth cone functions

A growth cone function defines the behaviour of a type of growth cone. It does this by defining how the bid value and corresponding actions are computed. A growth cone function has the following properties:

- A name which identifies the growth cone function.

- A function which is used to compute the bid value as well as the resulting action. The bid value is used in the auction among all filopodia. This is similar to the way genes' expression behaviour are represented, where the expression rate (corresponding to the filopodium bid value) and the protein target location (corresponding to the action) are computed concurrently.
- The geometry of the growth cone, including the number of filopodia, their length, etc.
- Parameters for the search for neighbouring cell components, including maximum search range, etc. This is explained in more detail in section 2.6.4.

See appendix B.2.2, lines 131–137 and 150–205 for two examples of growth cone functions. The algorithm for simulating growth cones is outlined in algorithm 3. *location* is a point in 3D space which represents the location of the tip of a filopodium. This value is initialized on line 1 with the location of the parent axon, i.e., the axon which is carrying out the growth cone action. This means that the first filopodium's location coincides with the location of the axon or dendrite. Subsequent filopodia will have locations that are generated stochastically (line 25). On line 2, *bestBid* is initialized to zero. For other bids to be considered in the auction, they must be greater than *bestBid* — this means that all negative bids are disqualified in the auction. If there are no qualifying bids, the growth cone mechanism does nothing.

The for-loop in lines 3–26 then generates a set number of filopodia, computes the bid value for each, and decides the winner of the auction. In line 4, the region of the simulation space close to the filopodium location is searched for other cell components. This is the collision detection part of the algorithm. The cell components found are added to the set *Neighbours*. Line 5 defines a branch point in the algorithm depending on whether any neighbouring cell components were found or not.

If no neighbour cell components were found, lines 6–13 will run. On line 7 the bid value and action is computed from the user-defined growth cone function described above. If the bid value is higher than the highest bid value encountered so far (line 8), then the resulting action and bid value are stored (lines 9–12). Since the *currentNeighbour* is not defined (line 6), the winning filopodium also does not define a corresponding neighbouring cell component (line 12).

**Algorithm 3** Growth cone simulation.

```
1: location is initialized to the location of the parent axon
2: $bestBid \leftarrow 0$
3: for number of filopodia in the growth cone do
4:   Search the simulation space for cell components whose locations are close to location.
     These cell components make up a set called Neighbours, which may be empty.
5:   if no neighbours were found, i.e., $Neighbours = \emptyset$ then
6:     currentNeighbour is undefined
7:     Evaluate the growthcone function using the current value of location. This computes
       a bid and an action. bid is a real number, and action is one of MIGRATE,
       PRE_SYNAPSE or POST_SYNAPSE.
8:     if $bid > bestBid$ then
9:       $bestBid \leftarrow bid$
10:      $bestLocation \leftarrow location$
11:      $bestAction \leftarrow action$
12:      bestNeighbour becomes undefined
13:    end if
14:  else
15:    for all cell components currentNeighbour in the set Neighbours do
16:      Evaluate the growthcone function as above, computing bid and action.
17:      if $bid > bestBid$ then
18:        $bestBid \leftarrow bid$
19:        $bestLocation \leftarrow location$
20:        $bestAction \leftarrow action$
21:        $bestNeighbour \leftarrow currentNeighbour$
22:      end if
23:    end for
24:  end if
25:  location set to a randomly selected point close to the location of the parent axon.
26: end for
27: if $bestAction = MIGRATE$ then
28:   Parent axon migrates to bestLocation.
29: else if $bestAction = PRE\_SYNAPSE$ then
30:   Parent axon migrates to bestLocation and forms a synapse with bestNeighbour so
      that the presynaptic terminal belongs to the axon and the postsynaptic terminal to
      the neighbour.
31: else if $bestAction = POST\_SYNAPSE$ then
32:   Same as for PRE_SYNAPSE, except that the synapse has the opposite orientation,
      with the postsynaptic terminal belonging to the axon and the presynaptic terminal to
      the neighbour.
33: end if
```

If one or more neighbours were found in the search on line 4, then lines 15–23 will run. We iterate over all the neighbours found (line 15), and evaluate the growth cone function as before (line 16), evaluating the resulting bid value (line 17) and storing the results of the leading bidder (lines 18–21), including the neighbouring cell component (line 21).

The user-defined function used to evaluate the bid value (lines 7 and 16) may make references to the *currentNeighbour*, including queries to surface protein concentrations of the neighbour, its distance from *location*, etc. This is part of the simulation of growth cones' ability to sense their environment, including signaling molecules on the surfaces of surrounding cells, functionalities which are important in both axon path finding and synapse formation. On line 7, the neighbour cell component is not defined, and queries will always return zero for surface concentrations of, and infinity[6] for the distance to the neighbour.

After the neighbours (if any) have been processed and the corresponding bid values computed and compared, the location of another filopodium is computed through a stochastic mechanism (line 25). Here the length of the filopodia, as defined in the growth cone function, is used to select this new location, specifically the distance from location of the next filopodium to the parent axon will be equal to this length. The outer loop then runs again, once for each filopodium in the growth cone.

After all filopodia have been processed, the resulting action is stored in *bestAction*, the location of the filopodium which won the auction is stored in *bestLocation*, and the neighbouring cell component (if any) which gave rise to the highest bid value is stored in *bestNeighbour*. This last field may or may not be defined at this point, depending on whether the auction winner was selected at line 8 or line 17. In either event, a final set of if-statements (lines 27–35) execute the winning action. Note that the actions *PRE_SYNAPSE* and *POST_SYNAPSE* require that *bestNeighbour* be defined, while the other action does not.

### 2.6.3 Justification of growth cone algorithm

Our approach to simulating growth cones is based on observed growth cone structure and behaviour, rather than a detailed understanding of the molecular mechanisms which underlie this behaviour. Consequently, we do not explicitly model any elements of the intracellular

[6] Java's `double` data type can represent positive and negative infinity.

skeleton underlying growth cone structure and motility.

The growth cone model as outlined in algorithm 3 consists of the following elements: Generation of filopodia at stochastically selected locations, computation of bid values and following auction, and implementation of action according to the winning filopodium. The generation of filopodia is justified from the physical structure of a growth cone, which consists of a core from which a number of fibres extend in more or less random directions. In our model, the locations of the filopodium tips are chosen so as to be at a constant distance from the centre of the growth cone — this is also based on the appearance of growth cones in nature. The remaining parameters determining the physical manifestation of the simulated growth cones, such as the length and number of filopodia, can be set by the user. The choice of values for these parameters must therefore be justified by the user.

The behaviour of growth cones in nature suggests that the filopodia, as they probe the environment of the growth cone, each represents different potential actions that the growth cone might take. In nature, growth cone migration occurs when one filopodium becomes enlarged while the others are retracted, indicating that the growth cone has somehow selected one action among several potential actions, discarding the others. Synapse formation is similarly initiated by initial contact between filopodia and the target cell. We use an auction mechanism to model this process of "winner-takes-all" competition. The behavioural characteristics of the simulated growth cone is then expressed through the computed bids and the associated actions (see lines 9, 18 and 11, 20 respectively).

Our model of growth cone function is not closely related to any of the earlier models outlined in section 1.5.3. However, there are some similarities to the model by Meinhardt (1999). In this model, chemical signals within the growth cone are explicitly represented. Through the simulation of activation and inhibition of the production of these chemicals, a "winner-takes-all" mechanism is produced as an emergent property. This chemical mechanism was initially proposed by Allan Turing (1952), and is widely recognized as being applicable to many forms of patterning in development. We also retain some elements of the model by Buettner et al. (1994), in that we explicitly model filopodia within a stochastic model of growth cone dynamics.

Our model of growth cone function is relatively more abstract than our gene model. There are two reasons for this: First, a detailed simulation of growth cones based on what is know about them would be computationally very expensive, since it would involve the explicit representation of intracellular skeletal elements, and their build-up and breakdown. Secondly, the mechanisms which determine growth cone behaviour are not well enough understood for such a model to be complete. While there are good theories for how growth cones transduce a single concentration gradient, we do not understand how multiple chemical signals are detected concurrently, or how the information from each type of receptor is integrated to give a particular behavioural pattern. Since our goal is to model a wide range of growth cone behaviour, including the most complex behaviour observed, a more abstract model is required. We believe that a very wide range of behaviours can be captured through our auction mechanism. Some of the capabilities of the model is demonstrated in a simulation system presented in chapter 5.

Our model is based on the assumption that a simple competitive mechanism underlies growth cone function, and that the complexity of growth cone behaviour is manifested through this competition, particularly in the comparison of the competitive strengths of the filopodia. Our model is supported by observations of how growth cones behave, not by an understanding of why they behave that way. It is therefore likely that it may have to be revised as our understanding of growth cones improve. However, lacking this understanding, the high level of abstraction of our growth cone model gives a good starting point for the simulation of axon guidance.

### 2.6.4 Optimization of collision detection

Measurements showed that the performance of the growth cone mechanism was severely impacted by the search for neighbouring cell components (line 4). This search amounts to finding a set of cell components that fall within a given distance from a particular point in 3D space (*location* in algorithm 3). The search has two components: An appropriate set of WorldNodes must first be selected, and then a linear search is conducted within each of these WorldNodes for cell components that meet the search criterion. Since nothing can be done to improve upon the linear search within each WorldNode, the initial optimization effort is directed at reducing the number of WorldNodes that need to be considered:

#### Maximum range of search

This is a user-definable setting. No cell components which are farther from *location* than this range will be considered. This radius is used to reduce the number of WorldNodes that need to be considered in the search.

#### A maximum number of neighbours

Only the $N$ closest cell components will be included in the set of neighbours. This is used to dynamically reduce the search range described above. When $N$ potential neighbours have been found, the search range can be set to the distance of the most distant of these $N$ potential neighbours, thereby potentially cutting down on the number of WorldNodes that need to be considered.

#### Ordering of WorldNodes

During the search the WorldNodes are visited in approximate order of increasing distance from *location*. This means that the closest neighbouring cell components will be found first, and will allow the search to be aborted as soon as the required number of neighbours or the required search range has been reached.

#### Filtering eligible cells

The filter is a boolean expression which is evaluated in the context of each potential neighbour cell component. Only if the expression evaluates to **true** will the cell component be included in the set of neighbours. This does not reduce the number of WorldNodes that need to be considered for the search — instead it reduces the number of neighbours in the resulting *Neighbours* set. This in turn will speed up the remainder of the growth cone algorithm since the loop in lines 15–23 will run through fewer iterations.

### 2.6.5 Time complexity of the growth cone algorithm

The growth cone algorithm's main iterative loop (line 3–26) is repeated once for each filopodium in the growth cone. This number is user-definable. It does not appear elsewhere in the algorithm, making the algorithm linear with respect to the filopodium count

$f$. The neighbour search in line 4 is, in spite of the optimizations given above, linear in the number of cell components $c$ in the vicinity of the parent axon, i.e., the "density" of cell components. Since the search is carried out once for each iteration of the outer loop, the overall time complexity from these two factors is $O(cf)$,

The growth cone function is specified by the user as part of the genome. Since the precise form of this function is therefore unknown, the complexity of evaluating the growth cone function is also unknown. However, since iteration is not typically involved in executing the growth cone function, it will in most cased be evaluated in constant time. Assuming that this is the case, lines 6–13 run in constant time.

The for-loop on line 15 introduces a linear dependence on the number of neighbouring cell components. As described above, this number may be limited to a user-defined parameter $N$, the maximum number of neighbour cells found. The body of the for-loop executes in constant time, just like lines 6–13. The time complexity of this stage is therefore $O(N)$. The selection of a new filopodium location (line 25) also runs in constant time. The neighbour set is traversed once for every filopodium, giving an overall time complexity of $O((c+N)f)$. If $N$ is large relative to $c$ so that the size of the neighbour set is limited by the cell component density $c$, the complexity becomes $O(cf)$. The execution of the resulting action (lines 27–33) involves adding the new action in the ActionQueue (section 2.10), which is a constant time operation.

## 2.6.6 Actions

A number of actions are used within growth cone functions to define the behaviour of the growth cone, specifically the computation of the bid value and the action of each filopodium. The growth cone related actions are:

| | |
|---|---|
| void ***buildPostSynapse***(number bid, number weight) | void ***migrate***(number bid) |
| void ***buildPreSynapse***(number bid, number weight) | |

corresponding to the MIGRATE, PRE_SYNAPSE and POST_SYNAPSE constants in algorithm 3. These are defined similarly to the gene expression primitives (see section 2.5.3), and they have the same "return"-statement semantics. The actions which try to build synapses have an additional parameter which specifies the initial synaptic weight of the synapse.

### 2.6.7 Queries

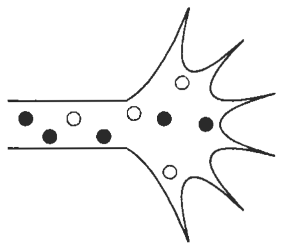
Growth cones' enhanced ability to sense their environment through collision detection is reflected in an extended set of queries that may be used within the context of growth cone functions. These queries are:

```
number growthconeCellDensity()
number neighborRange()
number growthconeConcentrationOf(protein signal)
number neighborConcentrationOf(protein signal)
```

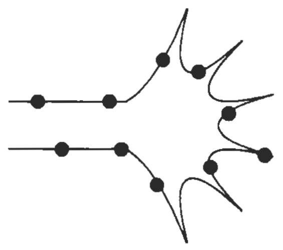
Below are listed the values computed by the queries in a filopodium $F$ whose parent cell component (typically an axon) is $C$, and whose neighbour cell component is $N$. When the protein $s$ is soluble, the queries from the filopodium $F$ compute the following values :

- **internalConcentrationOf** returns the intracellular concentration of the protein in the parent cell component$C$, see figure 2.5A.
- **surfaceConcentrationOf** always returns zero, since soluble proteins do not associate with the cell surface, figure 2.5B.
- **externalConcentrationOf** returns the extracellular concentration (as stored in WorldNodes) of the protein interpolated at the location of $F$ (same as for other cell components), see figure 2.5C — the concentration measured at $\oplus$ is returned.
- **growthconeConcentrationOf** returns the extracellular concentration (as stored in WorldNodes) of the protein interpolated at the location of $C$. In combination with **externalConcentrationOf**, filopodia can compute the concentration difference from their roots to their tips, which is their way of perceiving concentration gradients, see figure 2.5C — the concentration measured at $\otimes$ is returned. The name of this query has been chosen since the query is relative to the location of the centre of the growth cone structure, which is identical to the location of the parent cell component $C$.
- **neighborConcentrationOf** always returns zero, again since soluble proteins do not associate with the cell surface, see figure 2.5E.

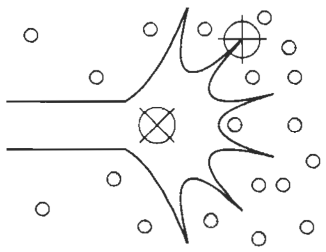
When the protein is membrane bound, the same queries compute the following values:



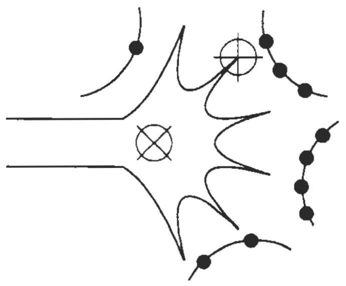
A: The intracellular concentration of protein. The internal concentration of both membrane bound and soluble proteins are identical to those of the parent axon or dendrite.



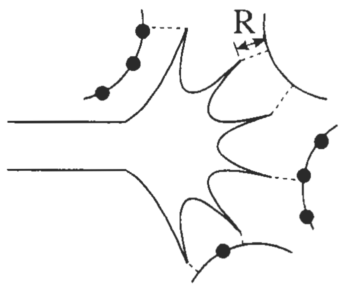
B: The surface concentrations of membrane bound proteins are also identical to those of the parent neurite.



C: The external concentration of soluble protein is computed as shown in figure 2.3A. Here, two different queries exist, one which measures the concentration relative to the tip of the filopdoia (for one filopodium indicated by a ⊖), and another which measures it relative to the centre of the growth cone (at ⊗).



D: The external concentration of membrane bound protein is measured as described in figure 2.3B, with two queries measuring either relative to the location of the tip of the filopodia (at ⊖) or relative to the location of the centre of the growth cone (at ⊗).



E: Unlike other cell components, filopodia perceive the presence and surface concentrations of nearby cell components. Within each filopodium, the query returns the surface concentration of the corresponding neighbouring cell component, as indicated by dashed lines. $R$ is the distance to the neighbour.

Figure 2.5: Definitions of protein concentrations as perceived by growth cones. Soluble proteins are shown as ○, membrane bound proteins as ●.

- **internalConcentrationOf** returns the intracellular concentration of the protein in the cell component$C$, as shown in figure 2.5A.
- **surfaceConcentrationOf** returns the surface concentration of the protein in the cell component $C$, figure 2.5B.
- **externalConcentrationOf** returns the extracellular concentration (i.e. summed over cell components within each WorldNode) of the protein interpolated at the location of $F$ (same as for other cell components), see figure 2.5D, measured at the location $\oplus$.
- **growthconeConcentrationOf** returns the extracellular concentration (as summed within each WorldNode) of the protein interpolated at the location of $C$. See figure 2.5D, location $\otimes$.
- **neighborConcentrationOf** returns the surface concentration of the protein in the cell component $N$. This allows filopodia to distinguish between cell components in their environment, e.g., for the purposes of forming synapses with high selectivity, see figure 2.5E. If the neighbour is not defined, this query returns zero.

The following queries are also specific to growth cone functions.

- **neighbourRange** returns the distance from the location of the filopodium tip to the neighbour, shown as $R$ in figure 2.5E. If the neighbour is not defined, infinity is returned.
- **cellDensity** returns a number representing how many cell components are close to $F$ (same as for other cell components). This query parallels **externalConcentrationOf**, except that is computes the cell density in stead of the protein concentration.
- **growthconeCellDensity** returns a number representing how many cell components are close to $C$. This query parallels **growthconeConcentrationOf**, computing cell density rather than protein concentration.

## 2.7 Neural activity

The variety and flexibility of neuronal activity observed in nature has lead to the proposal of a wide range of computational models of neural activity, varying in precision, temporal resolution, computational cost, etc. NeuroGene is designed in such a way that different cells may use different models of neural activity at the same time. However, the overall execution model of NeuroGene (described in section 2.10) currently limits these models to those that operate in discrete time, excluding e.g. spiking neuron models. Two neural models have been implemented, a linear and a non-linear model.

The different cell component types play the following roles in the implementation of neural activity:

- Soma may receive neural activation from any number of dendrites and postsynapses, and transmit activation via any number of axons and presynapses, all of which are children of the soma (keeping with the *child-parent* nomenclature of the tree data structure).

- Axons receive neural activation from their parent, which is either a soma or another axon segment. They transmit activation via any number of child axons and presynapses. Axons thus transmit neural activity away from the soma.

- Dendrites receive neural activation from any number of child dendrites and postsynapses. They transmit this activation to their parent, which is either a soma or another dendrite segment — thus transmitting neural activity toward the soma.

- Presynapses receive activation from their parent, which is either a soma or an axon. They transmit the activation to their corresponding "partner", which is always a postsynapse. The postsynapse is usually part of a different neuron.

- Postsynapses receive activation from corresponding presynapses. They transmit the activation to their parent, which is either a dendrite or a soma.

### 2.7.1 Bidirectional neuro-genetic coupling

It is thought that activity driven developmental processes may underlie the flexibility of the growing vertebrate nervous system, and the ability to simulate such processes is an important requirement of NeuroGene. The NeuroGene genetic programming language contains actions for altering the neural properties of cells, including synaptic weights, threshold potential, refractive periods, etc. Queries also exist for modeling the effect of neural activity on gene expression. Through these primitives, we achieve bidirectional coupling between genetic and neural activity, again omitting mediating regulatory proteins and other molecular mechanisms. By combining different classes of primitives, NeuroGene can be used to simulate the process by which neural activity causes axon growth, as reported by Carmichael and Chesselet (2002), or affect intracellular protein transport, as seen by Meyer-Franke et al. (1998), or affect gene expression in general, as reviewed by West et al. (2002). Conversely, using neural actions, genes may affect the neural properties of a cell, by e.g. changing firing thresholds or synaptic weights. Genetic and biochemical control of neural properties has been observed in nature, see e.g. the study by Yuan et al. (2002) who found that synaptic plasticity was altered by phosphorylation of ion channels. Phosphorylation is a common mechanism for rapid regulation of intracellular processes.

### 2.7.2 A linear model of neural activity

The linear neural model of some cell $i$ is characterized by the following parameters: The membrane potential $p_i$, the leakage rate $r_i$ and the threshold $T_i$. The membrane potential of cell $i$ at time step $t$ is computed from activity levels of a set of $M$ afferent (or input) cells at time step $t-1$:

$$p_i^t = p_i^{t-1} \exp(-r_i) + \sum_{j=1}^{M} a_j^{t-1} w_{ji} \tag{2.10}$$

where $w_{ji}$ is the synaptic weight from afferent cell $j$ to target cell $i$, and $a_j^t$ is the activity of cell $j$ in time step $t$. For cells using the linear model, the activity $a_j^t$ of cell $j$ in time step $t$ is defined as

$$a_i^t = \frac{p_i^t}{T_i} \tag{2.11}$$

where $T_i$ is the threshold value for cell $i$. With the exception of synaptic weights $w_{ij}$, all these parameters apply to a cell as a whole.

### 2.7.3 A non-linear model of neural activity

The non-linear model extends the linear model to include the following additional parameters: the predicate $F$ which is true if the cell is firing and false otherwise, the firing duration $f$ and the refractive duration $d$. Also, the threshold $T_i$ has a different meaning than above: here it determines whether or not the cell begins to fire in a given time step, i.e., $F$ becomes

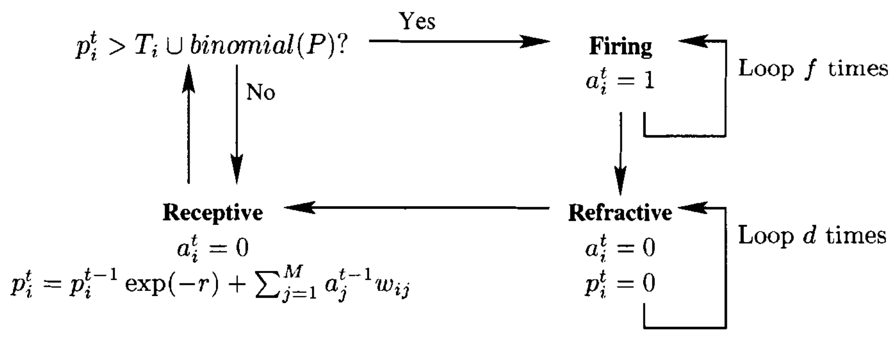


Figure 2.6: Flow-chart defining a non-linear model of neural activity. Expressions of $p^t$ shown how the membrane potential is computed in the receptive and refractive states. Cells are initially in the receptive state. $binomial(P)$ is true with probability $P$ and false with probability $1 - P$. When $P > 0$, receptive cells fire with a probability $P$ even though the membrane potential $p^t$ is below threshold. $f$ is the number of time steps the cell remains in the firing state, and $d$ is the number of times steps it remains in the refractive state.

true in time step $t$ iff $p_i^t > T_i$. The activity of the cell depends on $F$: If the cell $i$ is firing in time step $t$ then $a_i^t = 1$, otherwise $a_i^t = 0$. Once the cell starts to fire, it remains in the firing state for $f$ time steps. After the cell ceases to fire, it remains in a refractive state for $d$ time steps. During the refractive period, the cell in not receptive to neural activity. After the refractive period is over, the membrane potential is reset to zero, and the cell becomes receptive again.

The non-linear neural model also has a probability $P \in [0, 1]$ of spontaneously firing. This is needed to allow us to model certain stochastic behaviours exhibited by some neurons (Feller et al., 1997). The cell can be in one of three states: **receptive**, **firing** or **refractive**, see figure 2.6, where the term $binomial(P)$ represents spontaneous firing. In the receptive state, equation (2.10) defines the evolution of the membrane potential, just like in the linear model.

### 2.7.4 Actions and queries

Actions exist for altering the values of neural model parameters, as well as queries for getting the current values of these parameters implementing neuro-genetic coupling as outlined above. The set of actions and queries consequently depends on which neural model is being used for a particular cell. An error is generated when an inappropriate action or query is used, leading to the interruption of the simulation.

### Linear model

For each parameter of the linear neural model (leakage rate $r_i$, threshold value $T_i$ and synaptic weights $w_{ij}$) there exists queries for retrieving their current values (in **bold**), and actions for changing their values (in ***bold italic***). In addition, there is a query for retrieving the current membrane potential $p_i^t$, however this value is computed from the neural model, consequently the corresponding action for altering this value does not exist.

| | |
|---|---|
| number **leakageRate**() | void ***setLeakageRate***(number rate) |
| number **threshold**() | void ***setThreshold***(number) |
| number **synapticWeight**() | void ***setSynapticWeight***(number) |
| number **membranePotential**() | |

All these parameters apply to the whole cell, except for the action for altering the synaptic weight of any synapse, and the corresponding query for getting the current synaptic weight of the synapse — these apply to the current cell component only, and cause an error if they are executed within a cell component which is not a synapse.

### Additional primitives for the non-linear model

The additional parameters of the non-linear neural model (see figure 2.6) gives rise to additional actions and queries. Most of these are for setting parameter values or for retrieving their current values. The query **cellIsFiring** returns the firing state of the cell, **true** if the cell is firing, **false** if not.

| | |
|---|---|
| number **firingDuration**() | void ***setFiringDuration***(number) |
| number **refractoryDelay**() | void ***setRefractoryDelay***(number) |
| number **firingProbability**() | void ***setFiringProbability***(number) |
| boolean **cellIsFiring**() | |

These queries and actions all relate to parameters of the non-linear neural model, i.e., the firing duration $f$, the refractory delay $d$, the probability of spontaneous firing $P$. Similarly to the linear model, the firing state of the cell is computed by the model. Therefore the

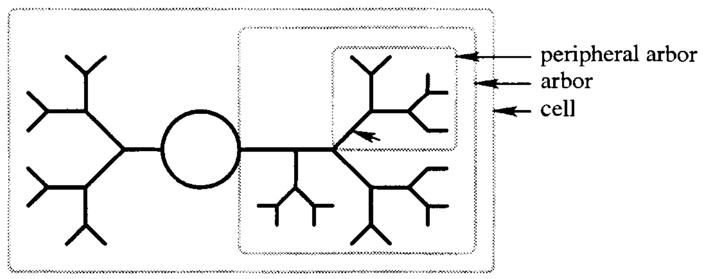


Figure 2.7: Image showing which parts of a neuron are included in various cell wide queries. In black is shown a neuron, consisting of a central soma and two highly branched axons or dendrites. The queries are carried out by the cell component indicated by an arrow. The "peripheral arbor" is the part of the axon that is "distal" to the querying axon segment. This is the subtree of which the querying segment is the root, as shown by the smallest box. The "arbor" as a whole is the whole axon which includes the querying axon, but does not include the soma or any other axons or dendrites (medium box). The "cell" includes the whole cell (largest box).

query **cellIsFiring**() retrieves the firing state $F$ of the cell, but no action exists for altering this state.

### Cell-wide queries

A number of normalization schemes from the connectionist literature assumes that cellular activity may be affected by the combined weight of many synapses. The following queries compute synaptic weights summed over defined sets of synapses, and thereby allow the straightforward implementation of various normalization schemes.

| | |
|---|---|
| number **arborIncomingWeight**() | number **peripheralArborIncomingWeight**() |
| number **arborOutgoingWeight**() | number **peripheralArborOutgoingWeight**() |
| number **cellIncomingWeight**() | number **cellOutgoingWeight**() |

All of these queries compute the sum of the synaptic weights of all synapses within a particular well-defined region of a simulated neuron. Figure 2.7 shows an example of which subset of a neuron's cell components are included in the different types of queries:

| | |
|---|---|
| Cell: | Relative to the cell component $i$ within which the query is executed, it defines the set of all cell components which together make up the cell to which $i$ belongs. |

This is the largest box in figure 2.7.

Peripheral arbor: Again relative to the cell component $i$, this includes the cell component $i$ itself, and all children (direct or indirect) of $i$. This is the smallest box in figure 2.7.

Arbor: This includes the cell component $i$ itself, all parent cell components of $i$ up to but not including the soma, and all children of these cell components. This is the intermediate box in figure 2.7.

For each subset, there is a query which returns the total weight of all postsynapses (**Incoming-Weight**) and presynapses (**OutgoingWeight**) within that subset of the cell. The incoming weight is the sum of the weights of all synapses via which neural activity may reach the current cell (the one of which the querying cell component is part) from other cells. Outgoing weight is the weights of all synapses via which neural activity may reach other cells from the current cells. For example, the query **arborOutgoingWeight**() will compute the sum of all the outgoing synapses that fall within the set of cell components defined as the arbor above. As usual, we have implemented these queries, and leave it to the experimenter to justify their use in any given context.

### 2.7.5 Exogenous neural input

As a compliment to the mechanisms for simulation of neural activity, NeuroGene also contains a system for generating exogenous neural activity. This activity is presumed to originate in some neural system that is not explicitly included in the NeuroGene simulation. This functionality may be used to simulate visual stimulation or other forms of sensory input, or activity originating in other neural systems.

The system is based on the following concepts: Exogenous activity is generated by neural activity *sources*. This activity may pass through *filters* before it is passed to receiving cells. Any number of different sources may exist within the same simulation, and together with the filters a large variety of different activity patterns may be generated. Currently about fifteen different filters and sources have been implemented, these are listed in appendix D.

Any number of cells may receive input from a single source. Each cell receives neural input from the source based on the location of the cell in 3D space. How the location information affects the generated activation depends on the particular sources and filters used, see appendix D. The use of spatial information in generating exogenous activation patterns allows the activity to be spatially coherent, i.e., correlation between inputs received by two different cells decreases with increasing inter-cell distance. This approach also harmonizes well with the emphasis we place on the spatial locations of cells and cell components in our design of the simulation space and cellular data structures. In chapter 5 we show how this system is used to generate simulated binocular visual input with varying inter-ocular correlation.

## 2.8 Simulation architecture

The simulation data structure as a whole represents the current state of the simulated biological system, as well as the genetic information governing how the simulation changes over time. It also manages the computations involved in conducting the simulation itself. However, it is not involved in the display, analysis or other interaction with the user. These functions are handled by components outlined in chapter 3. See figure 2.8 for a UML diagram (Liberty, 1998) showing the interactions between the classes which make up the simulation system.

The simulation system has been designed with computation speed as a primary requirement. As a consequence, the simulation classes are closely integrated and relatively interdependent.

### 2.8.1 Genome

The Genome stores the genetic information governing the activity of all cells. It stores all genes in a way that makes executing the genes for a given cell component efficient. It also stores all defined timers, growth cone functions and ligand-receptor binding relationships. The details of the internal representations of the genome are given in chapter 3.

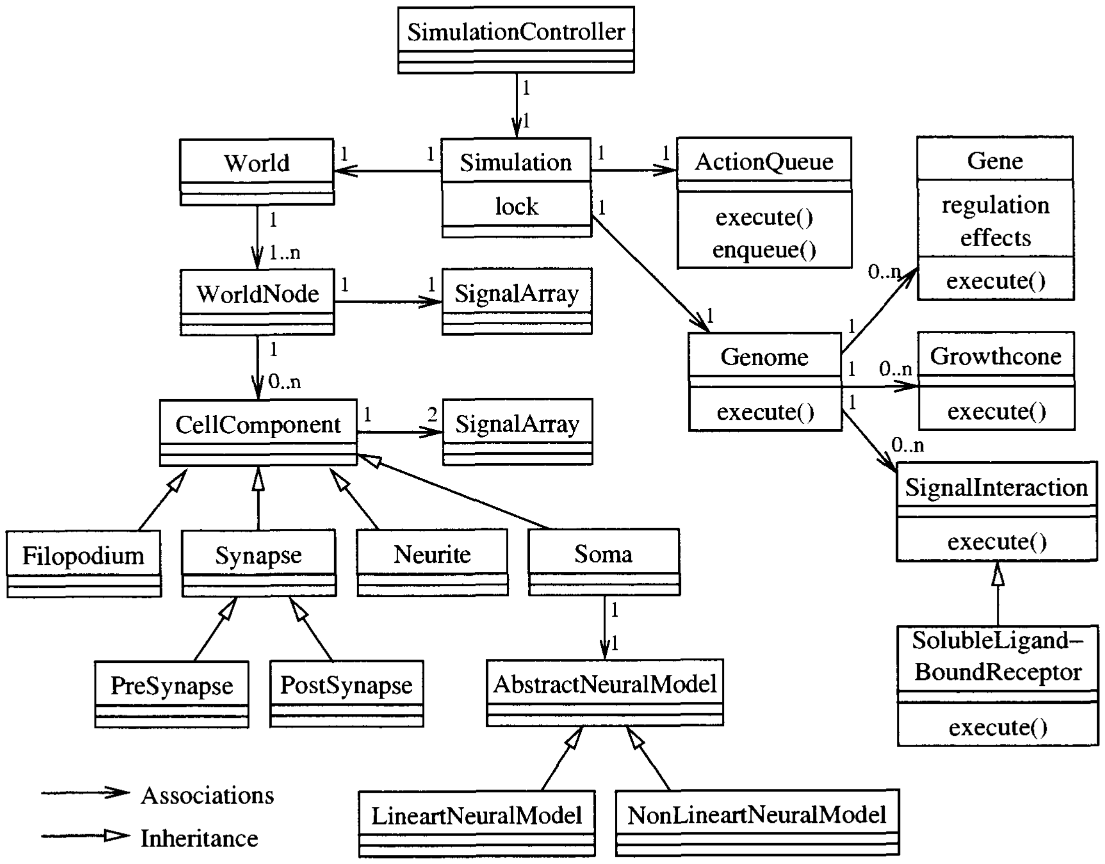


Figure 2.8: UML diagram of the simulation system. Each box represents a class. The top field of each box contains the name of the class, the middle field zero or more properties of instances of the class, and the bottom field zero or more methods of the class. Associations indicate "has-a" relationships, e.g. a Simulation instance has (owns) a World instance, an ActionQueue instance and a Genome instance. Numbers on the association relations indicate how many instances are owned, e.g. a WorldNode has one SignalArray, while a CellComponent has two SignalArray instances (representing internal and surface concentrations respectively). Inheritance shows "is-a" relationships, e.g. a Soma is a (special kind of) CellComponent. In terms of implementation in an object-oriented language like Java, associations are represented by properties, e.g. the class Simulation has a property of type World. Inheritance is represented through class inheritance, e.g. the class Soma extends the class CellComponent. The class SimulationController is part of the simulation control system described in chapter 3, see figure 3.7.

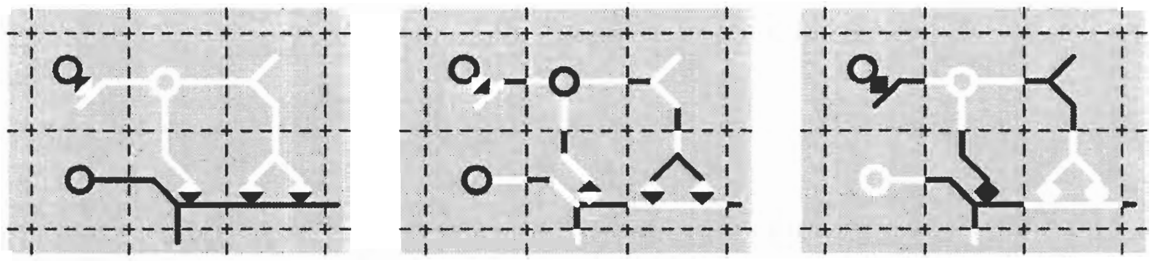


Figure 2.9: Three cells shown within a grid of WorldNodes. Broken lines show the boundaries between WorldNodes. All three panels show the same situation, but with different shading of cell components to highlight different aspects of the data structure. *Left*: Each cell is shown with uniform shading, one cell in white, and two different cells in black. Circles represent the Somas, of which each cell has exactly one. Lines represent the axons and dendrites, which are not distinguished in this figure. Triangles represent the synaptic terminals, two of which make up one synapse. Note that each cell may span multiple WorldNodes, and that the two synaptic terminals making one synapse each belong to a different cell. *Centre*: Individual cell components are shown in contrasting shading. Note that axon and dendrites are segmented such that each segment is contained within a single WorldNode. All axon and dendrite segments are straight, so that e.g. the white "<"-shape in the top right WorldNode represents two segments. Also note that each cell has the topology of a tree, with the soma as the root node. *Right*: Cell components are shaded according to the WorldNode that contains them. Note that a WorldNode may contain cell components belonging to different cells.

## 2.8.2 World and cells

The World represents the simulation space in which the simulated system exists. The physical space is subdivided into a 3D array of small cubic elements (WorldNodes), all of the same size. The length of an edge of the cube is one in the arbitrary unit of length used throughout the simulation system (see section 2.10.8). Each WorldNode may contain any number of cell components (see figure 2.9 right), stored in a singly linked list. This data structure is chosen for rapid modification of the collection of cell components, anticipating that the cell components belonging to a given WorldNode may change often during the course of a simulation. The cells themselves are represented by doubly linked tree structures composed of cell components (figure 2.9 centre). While the cells may span multiple WorldNodes (figure 2.9 left), each cell component is contained within a single WorldNode (figure 2.9 centre). A doubly linked structure is chosen here since cell components representing a single cell interact in many ways, and fast access to both parent and child entities in the tree structure is required.

## 2.9 Simulation initialization

Many neurodevelopmental processes occur in the context of partially or fully formed organs, consisting of large number of cells, with particular spatial arrangements and even particular neural interconnections already in place. NeuroGene needs a powerful mechanism for specifying the starting point for simulations, including the initial arrangements of cells. We chose to use a second scripting language for this purpose. The scripting language is similar to the gene scripting language, and it uses much of the same infrastructure, including the parser. The language also allows the specification of complex inter-cellular connectivity patterns (through axons, dendrites and synapses), initial protein concentrations, etc. The geometry scripting language has some features that are absent in the gene language, including iteration (**for**, **while** and **do-while**). It also has syntax for specifying arbitrary cell initialization blocks, using most of the actions available to the gene scripting language. Using this feature, cell components may be initialized to have arbitrary protein concentrations, neural properties, etc. Variables and multi-dimensional arrays of type **cell** (see table 2.1) can be used to construct complex neural circuits. Exogenous neural filters (see section 2.7.5) are also constructed based on instructions in the geometry script, and cells are initialized to receive neural input from such filters.

## 2.10 Execution model

NeuroGene simulations progress in a series of discrete simulation time steps. Each time step consists of a set number of stages that are carried out in a fixed order. Each of these stages will be described in turn in sections 2.10.1–2.10.6. These stages are executed sequentially in this order during one simulation time step.

An important goal of the NeuroGene execution model is to achieve simulated parallelism among genes and among cell components. Consider a situation with two genes $g_1$and $g_2$ being expressed concurrently within the same cell component. If the changes caused by the simulated expression of the gene $g_1$ were applied to a cell component before some other gene $g_2$ is evaluated within that same cell component, the expression of $g_2$ might be affected by the result of expression of $g_1$. The end result might be different if $g_2$ was evaluated before $g_1$. In nature genes act in parallel. In order to simulate this faithfully, the execution order of genes must not affect the outcome of simulations.

### 2.10.1 Process pending changes

In order to ensure that genes act in parallel, genes cause no changes to cell components until all the genes and all cell components in the simulation have been processed. A queue of pending changes (the ActionQueue) is used to store actions until all genes have been processed. The pending changes on the queue are applied here in stage 1 of the subsequent simulation time step. Each action in the queue applies to a particular cell component, and describes some change in the state of that cell component.

Assuming that making the changes to the simulation as specified by each element in the queue is done in constant time, the time complexity of processing the complete set of pending actions is $O(n)$ where $n$ is the number of pending actions. The dependence of $n$ on the number of genes and the number of cell components will depend greatly on the particulars of the simulation in question, but it will generally vary linearly with the number of cell components, and also approximately linearly with the "size" of the genome, however one chose to measure this.

#### Visualization of growth cones

It may seem counter intuitive that the processing pending changes is the first stage in a time step. This is related to the visualization of growth cones in the GUI: The growth cone model (see algorithm 3) goes through the following steps: I: Create $n$ filopodia. II: Select a winning filopodium. III: Add the action determined from the winning filopodium to the queue of pending actions. IV: Processing the pending changes.

Steps I, II and III occur as part of the execution of genes, more specifically as part of the effects section of one or more genes, see algorithm 2, line 12. In step IV the pending changes resulting from growth cones are processed along with all other pending changes as described above. In order for the user to be able to use the NeuroGene visualization tools to

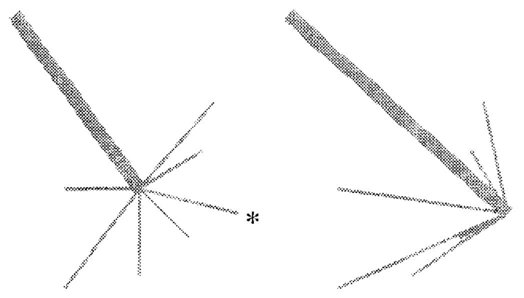


Figure 2.10: Growth cone shown similar to how they are drawn in the NeuroGene GUI. The same growth cone is shown, either as it appears when pending actions are implemented at the beginning of the subsequent time step (left), or at the end of the current time step (right). In the image on the left, the "winning" filopodium (see section 2.6) is marked with a *. Assuming that the growth cone operation is MIGRATION, the resulting action is a migration to the location of the tip of the winning filopodium. On the right is an image of what the same growth cone looks like after this action has been processed. This image is less instructive in showing the result of the growth cone operation, in particular the winning filopodium can not be seen.

understand (and debug) the behaviour of growth cones, growth cones have to be visualized after step II is complete but before step IV is commenced. Figure 2.10 shows the difference between drawing the image showing the same growth cone after II (left) and after IV (right).

A typical use of the NeuroGene GUI is to pause the simulation so that its state no longer changes, and then use the visualization mechanisms of the GUI to investigate the state of the simulation. For the visual display to be interactive while the simulation is paused, the visualization cannot happen during or as a part of the simulation time step, it has to occur between the end of one time step and the beginning of the next. Consequently, IV must be postponed until the subsequent time step.

### 2.10.2 Update neural state

Stage 2 is similar to the stage 1 in that it updates the neural state of the simulation to that computed in the previous time step.

### 2.10.3 Protein diffusion and decay

As outlined in section 2.4.1, extracellular concentrations of all soluble proteins are updated to reflect the diffusion of these proteins through the extracellular space. All protein concentrations in the simulation (including membrane bound and intracellular concentrations) are then reduced to reflect the decay of each protein.

Computing the diffusion of one protein for one WorldNode instance is done in constant time. The same is true for simulating decay. Adjusting the extracellular protein concentrations for the effect of diffusion and decay therefore has time complexity $O(np)$ where $n$ is the size of the simulation space, and $p$ is the number of soluble proteins. Computing the decay of proteins within or on the surfaces of cells has time complexity $O\left(cp\right)$ where $c$ is the number of cell components and $p$ is the number of proteins.

### 2.10.4 Ligand-receptor binding

In this stage of the simulation time step, the ligand-receptor binding model shown in algorithm 1 is applied to each WorldNode in the simulation space. After this step, the concentrations of all ligand, receptor and complex proteins in all WorldNodes will be at equilibrium. As mentioned above, the complexity of the model of ligand binding for one WorldNode is linear in the number of cell components $N_i$ in each WorldNode $i$. For the simulation as a whole, the complexity is therefore linear with respect to $\sum_i N_i$ where the sum is over all the WorldNodes in the simulation space, i.e., the total number of cell components in the simulation as a whole. It is also linear with respect to the number of receptor-ligand relationships in the genome.

### 2.10.5 Execute genes

In this stage of the simulation time step, the genome is "executed". This means that for each gene in the genome, the genetic model as shown in algorithm 2 is applied to each cell component in the simulation.

In this process, gene expression will cause protein to be added to cell components and WorldNodes, according to the expression profile of genes as defined as part of the genome (see lines 4, 7 and 9 in algorithm 2). The actions of adding the appropriate amounts of proteins to cell components and WorldNodes are not executed immediately, instead they are added to the ActionQueue to be executed later.

Also resulting from gene expression are other actions, such as cell division, migration, axon growth, etc. (see line 12 in algorithm 2). These actions are likewise put in the queue of pending actions to maintain parallelism.

Genes which invoke a growth cone action cause the growth cone mechanism to be executed immediately. The growth cone action in turn may add other actions (such as migrate or synapse formation actions) to the ActionQueue for later execution. Since the growth cone structure is not detectable by other cell components, and otherwise does not alter the simulation data structure in any way, this does not violate the requirement of parallelism among cell components.

Assuming that the gene model shown in algorithm 2 runs in constant time, the complexity of this step is $O(cg)$ where $c$ is the number of cell components in the simulation and $g$ is the number of genes in the genome. The assumption regarding constant time performance will usually be valid, but this will depend on the exact form of the regulation and effects portions of the genes in the genome, which is under the user's control when they design the genome.

### 2.10.6 Neural activity

In this stage the neural activity level of all cells are computed. NeuroGene only supports discrete-time models of neural activity, where the activity of each cell in time step $n$ can be computed from the neural activity levels of all cells in time step $n-1$. In this stage of the simulation time step, all cells (i.e. all somas) are visited, and the activation levels for time step $n$ are computed. The activation of cells will not be updated to these new values until stage 2 of the subsequent time step.

### 2.10.7 Ordering of computation stages

The six stages above together constitute one simulation time step. At the end of the time step, the simulation may enter into a paused state in which no further computation occurs, or it may immediately commence the computation of the subsequent simulation time step. While in the paused state, the simulation may be resumed at any time. Pausing and resumption is controlled by user from the NeuroGene GUI, as will be outlined in chapter 3 (section 3.3). The functions by `org.neurogene.simulation.Simulation.doStep()` and `org.neurogene.world.World.execute(Genome)`[7] coordinate the execution of each simulation time step.

Within each of the six stages, a larger or smaller portion of the simulation system as shown in figure 2.8 is traversed. If the system is seen as a tree structure, with Simulation as the root node, these traversals are depth-first from the root, and extends to various depths of the data structure depending on the simulation stage in question. To summarize, the stages of the simulation time step proceed in the order listed below, where $n$ refers to the current and $n-1$ to the previous time step. It is suggested that figure 2.8 be consulted while reading this list, this may help clarify the relationships between the various classes involved. In the function identifiers below, the capitalized class names precede the function names.

1. Process pending changes from previous time step. Before this stage, cell components and protein concentrations are those of time step $n-1$, specifically those of stage 5 of the previous time step. The class `ActionQueueElement` has been omitted from figure 2.8 for clarity. The functions involved in processing this step are:
   (a) `org.neurogene.simulation.Simulation.doStep()`
   (b) `org.neurogene.simulation.ActionQueue.execute()`
   (c) `org.neurogene.simulation.ActionQueueElement.execute()`
2. Update neural state to that computed in stage 6 of time step $n-1$. Note that of all cell components in the simulation, only Soma instances are affected in this stage. The functions involved are:
   (a) `org.neurogene.simulation.Simulation.doStep()`
   (b) `org.neurogene.world.World.execute(Genome)`
   (c) `org.neurogene.world.World.exec_updateActivity()`

[7]Using the standard Java naming convention for software components, the initial part designates the package `org.neurogene.world`, the capitalized name is the class name `World`, and the name followed by `()` is the function name `execute(Genome)`. Following this standard, the source code of this class is to be found in a file called `World.java`, in the directory `org/neurogene/world`. The name of the function parameter is omitted.

   (d) `org.neurogene.world.WorldNode.exec_updateActivity()`
   (e) `org.neurogene.cell.Soma.updateFiringState()`
   (f) `org.neurogene.activity.SomaActivityModel.updateFiringState()[abstract]`
   This module is omitted from figure 2.8 for clarity — it is an abstract base class from which `AbstractNeuralModel` extends. Which is overloaded by the functions
      i. `org.neurogene.activity.AbstractNeuralModel.updateFiringState()`
      ii. `org.neurogene.activity.NonLinearNeuralModel.updateFiringState()`

3. Compute protein concentration changes due to diffusion and decay. After this stage is complete, all protein concentrations are correct for time step $n$ with respect to diffusion and decay. The following functions complete this stage, where `World.exec_-computeDiffusionDecay()` computes the diffusion and decay for extracellular proteins, and `SignalArray.computeSignalDecay()` computes the decay of membrane bound and intracellular protein concentrations:
   (a) `org.neurogene.simulation.Simulation.doStep()`
   (b) `org.neurogene.world.World.execute(Genome)`
   (c) `org.neurogene.world.World.exec_computeDiffusionDecay()`
   (d) `org.neurogene.cell.CellComponent.computeSignalDecay()`
   (e) `org.neurogene.signal.SignalArray.computeSignalDecay()`

4. Compute protein concentration changes due to ligand-receptor binding. Before this stage receptor-ligand-complex concentrations may not be at equilibrium, due to changes in protein concentrations which may have occurred in stages 1 and 3. After this stage is complete, all protein concentrations are at equilibrium with respect to all defined receptor-ligand relationships. The equilibrium concentrations are computed by the following functions:
   (a) `org.neurogene.simulation.Simulation.doStep()`
   (b) `org.neurogene.world.World.execute(Genome)`
   (c) `org.neurogene.world.World.exec_computeLigandBinding(Genome)`
   (d) `org.neurogene.simulation.Genome.exec_computeLigandBinding(WorldNode)`

   (e) `org.neurogene.signal.SignalInteraction.execute(WorldNode) [abstract]`
   Which is overloaded by the function
      i. `org.neurogene.signal.SolubleLigandBoundReceptor.execute(WorldNode)`

5. Execute the genome. No changes to the simulation occur during this step. All actions resulting from gene expression, including protein expression and actions resulting from simulated growth cones, are added to the queue of pending actions by `ActionQueue.-enqueue()`. These will be processed in stage 1 of time step $n + 1$. It is during this stage that growth cones are also simulated.
   (a) `org.neurogene.simulation.Simulation.doStep()`
   (b) `org.neurogene.world.World.execute(Genome)`
   (c) `org.neurogene.world.World.exec_genome(Genome)`
   (d) `org.neurogene.world.WorldNode.exec_genome(Genome)`
   (e) `org.neurogene.simulation.Genome.execute(CellComponent)`
   (f) `org.neurogene.parser.ast.GeneAstNode.execute(CellComponent)`
   (g) `org.neurogene.parser.ast.GrowthConeFunctionAstNode.execute(CellComponent)`
   (h) `org.neurogene.simulation.ActionQueue.enqueue()`

6. Compute neural activity of all cells. Again, no changes are made to the simulation during this step. New activation values (for time step $n + 1$) are computed from the current (time step $n$) values, and stored separately. The values will be updated in stage 2 of time step $n + 1$.
   (a) `org.neurogene.simulation.Simulation.doStep()`
   (b) `org.neurogene.world.World.execute(Genome)`
   (c) `org.neurogene.world.World.exec_computeActivity()`
   (d) `org.neurogene.world.WorldNode.exec_computeActivity()`
   (e) `org.neurogene.cell.Soma.integrateAndFire()`
   (f) `org.neurogene.activity.AbstractNeuralModel.integrateAndFire() [abstract]`
   This function is overloaded in two different classes to implement the linear and non-linear neural models (see section 2.7):

  i. `org.neurogene.activity.LinearNeuralModel.integrateAndFire()`
  ii. `org.neurogene.activity.NonLinearNeuralModel.integrateAndFire()`

Each stage is completed for the entire simulation, including all WorldNodes and cell components, before the next stage begins. A single call to `Simulation.doStep()` causes the computation of exactly one simulation time step, causing each of the six steps to be executed exactly once, in the order they are listed above.

This ordering has been selected with the goal that both the genome and the user should "see" the simulation in the same consistent state. With respect to the genome, this means that throughout the processing of stage 5 above, protein concentrations should reflect the diffusion, decay and receptor-ligand binding relations as defined as part of the genome. This is ensured in stages 3 and 4. All cell components should also be in a state that reflects the genetic changes resulting from the previous time step. This is ensured in stage 1 and 2.

The user will be able to visualize and inspect the simulation in the state it has at the completion of stage 6 — it is at this point in the simulation time step that visual GUI components are updated to reflect the state of the simulation.

### 2.10.8 Units of measurements

NeuroGene has been designed without reference to any universal constants or other parameters that define the units of measurement of the numbers used to represent the state of a simulation. However, many of the parameters of a simulation, as defined as part of a genome, do have units. The values used for these parameters will therefore define the units of time, distance, concentration and electric potential that NeuroGene will use. The unit of time chosen will be equal to the duration of a simulation time step. The unit of distance will likewise become the lengths of the edges of each WorldNode, and the unit of concentration will become the unit of all concentrations computed by NeuroGene. The parameters and their units are summarized in table 2.2 , together with the equation from which the units can be derived. Also listed are examples of commonly used units. The unit $M$ (*molar*) is the concentration unit used in chemistry. One molar is defined as $6.022 \times 10^{23}$ molecules per litre — for a compound with molecular weight of $x$, the concentration of $1M$ corresponds to $x$ grams per litre. Molar (or variants like mM or $\mu$M) must be used as unit of concentration when ligand-receptor binding is simulated.

| Parameter | Units | Example | Definition | (equation) |
|---|---|---|---|---|
| Physical parameters | | | | |
| Decay rate $k$ | $T^{-1}$ | $s^{-1}$ | $dC/dt = -kC$ | (2.1) |
| Diffusion rate $D$ | $L^2T^{-1}$ | $cm^2s^{-1}$ | $\partial C/\partial t = D\partial^2C/\partial x^2$ | (2.2) |
| Ligand-receptor binding constant $K_d$ | $C$ | $M$ | $K_d = [R][L]/[C]$ | (2.6) |
| Biological parameters | | | | |
| Gene expression rates | $CT^{-1}$ | $Ms^{-1}$ | — | |
| Cell migration speed | $LT^{-1}$ | $cm\, s^{-1}$ | — | |
| Membrane potential, etc. | $P$ | $mV$ | — | |

Table 2.2: Units of measurement of various physical and biological parameters used in NeuroGene simulations. $C$ represents concentration, $T$ represents time, $L$ represents a unit of distance and $P$ electric potential. If the parameters are given in the units given as examples, then all concentration values computed by NeuroGene will have units of $M$ ("molar", unit of concentration used in chemistry), the length of one edge of a WorldNode will be one centimetre ($cm$), and the duration of a simulation time step will be one second ($s$). If other dimensions are required, all that needs to be done is to ensure that the values of all physical parameters are adjusted to use the appropriate units.

## 2.11 Summary

NeuroGene simulations are based on data structures which are able to represent a collection of neurons within a 3D environment. The neurons may interact through various chemically and electrically mediated mechanisms, and are controlled by genetic information. In this chapter I have outlined what these data structures look like, and the algorithms that are used to simulate biological processes that occur as part of neurodevelopmental simulations.

Our goal has been to create a simulator which can model a wide range of different developmental processes. Some of these processes, as they occur in nature, are well understood, while others are not. Poorly understood processes, such as growth cone function, can only be simulated using abstractions, since the details of the mechanism are unavailable to us. This has lead to a system where certain processes may be modeled with a high level of detail (such as receptor-ligand binding), while others are simulated using abstractions based on phenomenological descriptions only, such as the growth cone model. Modeling of gene expression falls somewhere in between these extremes.

We have limited the ability to simulate biologically implausible processes only when such limitations are conceptually and implementationally simple. Finally, we have tried to optimize the mechanisms and data structure for fast execution, and secondly, for small memory requirements.

In the next chapter, the software engineering aspects of the design and implementation of NeuroGene will be presented. The chapter centres mainly on the user interface of NeuroGene, consisting of the genetic programming language and the graphical user interface.

# Chapter 3

# Methods II: System architecture

In this chapter I outline the overall architecture of the NeuroGene system. This includes the input and internal representation of genetic information, the graphical user interface (GUI), file I/O and performance optimization — in short, the aspects of the NeuroGene system which are not directly related to the simulation of biological phenomena.

In section 3.1, the implementation of the parser used to read gene and geometry scripts is described. The data structure used for internal representation of such scripts is also outlined. In section 3.2, the overall design of the NeuroGene system, including the simulation engine, GUI, file I/O components, etc., is described. Also in this section, design patterns which play important roles in the architecture are described in some detail. In section 3.3 the implementation of the GUI is described. Section 3.4 is concerned with improvements to the performance of NeuroGene. Finally section 3.5 describes the subsystem for input from and output to files.

## 3.1 The NeuroGene language

The NeuroGene genetic programming language is used to specify the genetic information governing the behaviour of the simulated cells. Following standard compiler implementation techniques, the language is specified in the form of a context-free grammar. An automatic parser-generator is used to create a parser based on this grammar. When presented with an input script, the parser builds an *abstract syntax tree* (AST) data structure. The AST is used as the internal representation of the information contained in the input script. The instructions encoded in the input script are executed by recursively traversing the AST.

### 3.1.1 Abstract syntax tree

Figure 3.1 shows an example of a simple grammar in BNF notation (Louden, 1993), as well as an example of the parsing of a string that confirms to this grammar, including the resulting abstract syntax tree. The grammar consists of six productions ($p_0$–$p_5$). These define a language which consists of numerical expressions involving the operators '+' and '*', as well as parentheses. Each production has exactly one non-terminal symbol on the left, which makes this a "context-free" grammar. Figure 3.1A shows an example of a string in the language. B–F shows the stepwise parsing of this string. In B the string as a whole is parsed by matching production $p_0$, dividing the string at the first occurrence of the "+" token. In C productions $p_3$ and $p_1$ are used on the left and right, respectively. In D, $p_2$ is applied, etc. While the image in H reflects the complete parse of the string, the abstract syntax tree produced by the parsing is the simplified representation shown in I.

### 3.1.2 AST classes

A total of more than 200 different Java classes together implement the AST data structure produced by the NeuroGene parser. Such a large number of classes is needed since each class only implements one simple operation. This has two important advantages: One is that execution of the gene script by traversing an AST data structure is fast, since the execute() function (see figure 3.3) of each node, which implements the traversal of the AST, is very simple. It also makes each AST node class simple and easy to write and maintain.

All AST node classes are derived from the abstract base class AstNode. Figure 3.2 shows part of the inheritance tree of the AST classes. Extending from the base class AstNode are a number of other abstract classes which represent expressions of different types, such as numerical (FloatAstNode), truth values (BoolAstNode), strings (StringAstNode), etc. Extending these type-specific classes are other classes (only a few examples are shown) which implement simple functions such as addition, cosine, the comparison <, if-clauses, etc.

| | | | |
|---|---|---|---|
| $p_0$: | <sum> | $\rightarrow$ | <term> + <sum> |
| $p_1$: | <sum> | $\rightarrow$ | <term> |
| $p_2$: | <term> | $\rightarrow$ | <factor> * <term> |
| $p_3$: | <term> | $\rightarrow$ | <factor> |
| $p_4$: | <factor> | $\rightarrow$ | ( <sum> ) |
| $p_5$: | <factor> | $\rightarrow$ | **number** |

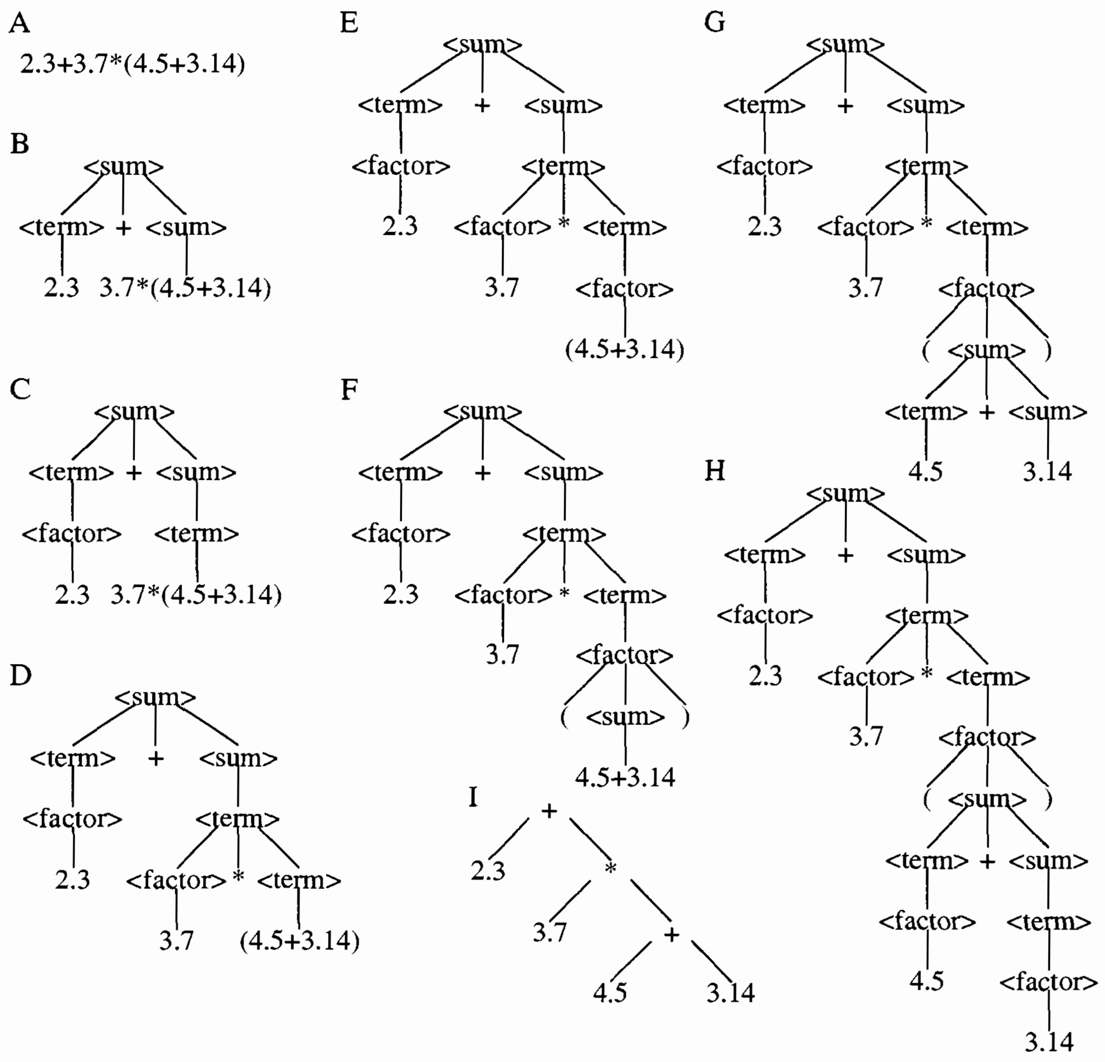


Figure 3.1: Example of a simple context free grammar and the parsing process based on the grammar. Non-terminal symbols are shown like <this>, all other symbols (*, +, left and right parentheses and **number**) are terminal. Terminal symbols match exactly one token, non-terminal symbol may match one or more tokens.

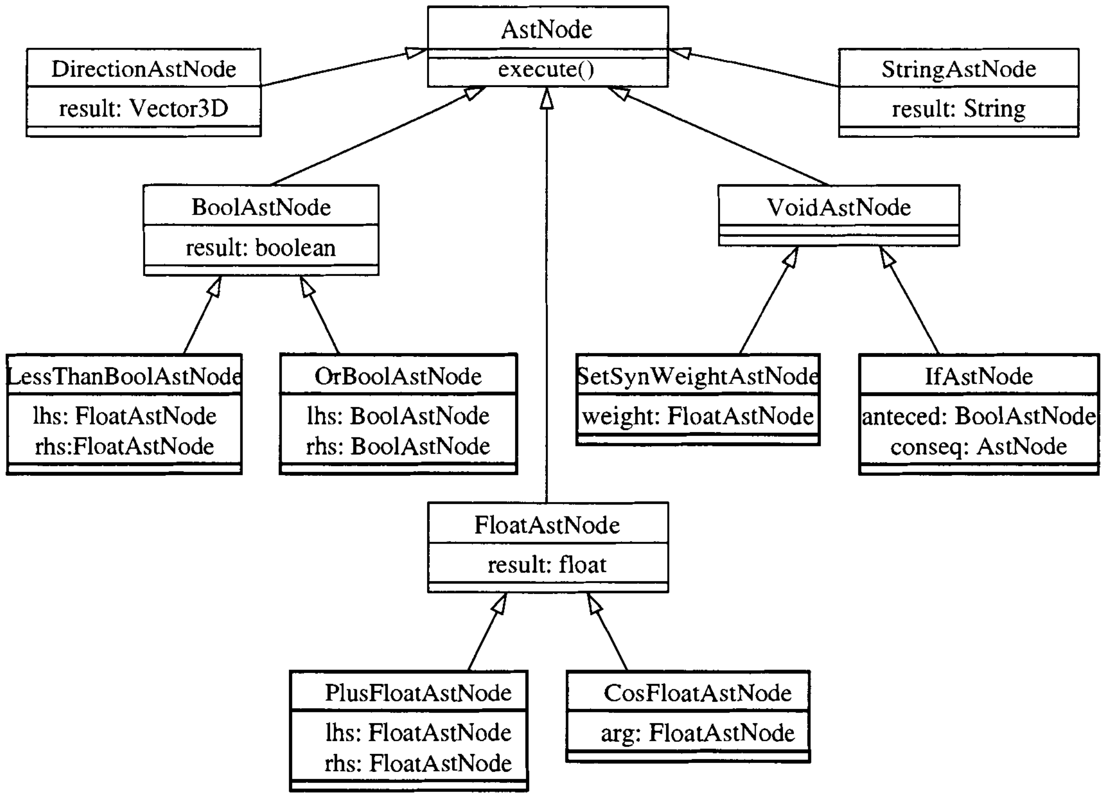
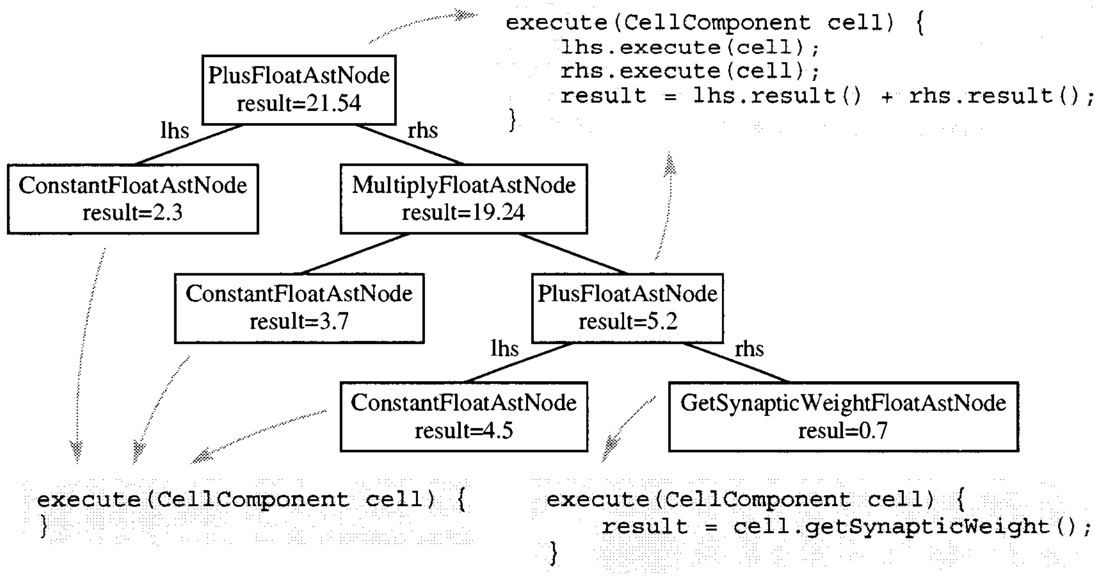


Figure 3.2: Inheritance diagram of AstNode. Included are all classes that directly extend AstNode, and some representative examples of classes which implement specific functionality. Abstract classes shown in light outlines, and concrete classes in heavy outlines.

Figure 3.3 shows an example of how an AST data structure representing a numerical expression ("2.3+3.7*(4.5+synapticWeight())") is used to evaluate that expression. During execution each node in the AST tree structure executes all subtrees below it. It then retrieves the values from the subtrees and computes it's own result value. The result values are thus computed depth first. After the tree representing the entire expression has been traversed in this manner, the resulting value may be retrieved from the root node of the tree. Expressions representing boolean expressions (such as `synapticWeight()` $\leq$ `0.75`) are evaluated in a similar way.

Classes which extend the VoidAstNode class represent operations that don't have result values, but which cause changes to the simulation system, i.e., actions. An example is the



```
execute(CellComponent cell) {
    lhs.execute(cell);
    rhs.execute(cell);
    result = lhs.result() + rhs.result();
}
```

```
execute(CellComponent cell) {
}
```

```
execute(CellComponent cell) {
    result = cell.getSynapticWeight();
}
```

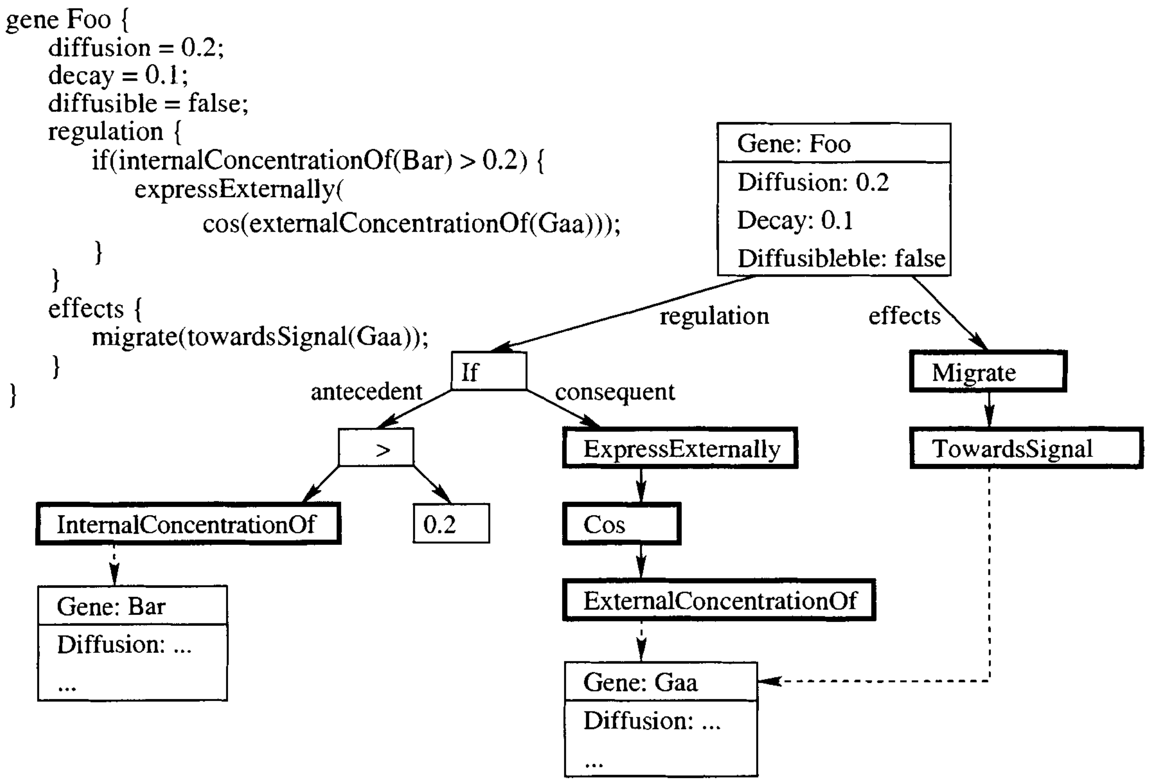
Figure 3.3: Evaluation of an expression by an AST data structure, similar to the one shown in figure 3.1G. The execute() functions for some of the classes are shown, showing the recursive calls which cause the whole AST data structure to be traversed (PlusFloatAstNode), and how information is retrieved from the cell component (GetSynapticWeightFloatAstNode). In the case of ConstantFloatAstNode, the execute() function does nothing.

AstNode class which represents the action to change the synaptic weight of a synapse (see SetSynWeightAstNode, figure 3.2). When such AST nodes are traversed, they enter new actions into the ActionQueue (section 2.10).

### 3.1.3 Language definition

The syntax of NeuroGene's genetic programming language is loosely modeled on the syntax of the Java programming language. It has static typing (the types of all variables are established when the input script is parsed), and uses lexical scoping of variables (Louden, 1993). A number of simple examples written in this language appear in figures in the remainder of this chapter.

Figure 3.4 shows an example NeuroGene gene and the AST data structure which is constructed during parsing and used for internal representation of the gene. Note that the data structure is not strictly a tree, since nodes such as the query "InternalConcentrationOf"



```
gene Foo {
    diffusion = 0.2;
    decay = 0.1;
    diffusible = false;
    regulation {
        if(internalConcentrationOf(Bar) > 0.2) {
            expressExternally(
                    cos(externalConcentrationOf(Gaa)));
        }
    }
    effects {
        migrate(towardsSignal(Gaa));
    }
}
```

Figure 3.4: A short example of a gene together with the abstract syntax tree (AST) data structure representing the gene. Nodes with heavy outlines will have been allocated by the LibraryFunctionTable system, see section 3.1.7.

refers to the "Gene" AST node which represents the protein whose internal concentration is to be queried. However, when the gene is executed, such links are not followed. This is indicated by the link being shown as a dashed line. Such links are instead used to retrieve identifying information from gene nodes needed to execute the query.

### 3.1.4 Language efficiency

The design of a programming language, like the design of other types of user interfaces, is as much an art as a science. There are many measures for judging language design, some are listed in table 3.1. Depending on the intended use of the language and the target audience,

| Property | Description |
|---|---|
| High priority language properties | |
| Readable | The meaning of a program is readily apparent |
| Consise syntax | There are no redundant syntactic elements |
| Expressive | It is easy to encode complex procedures and structures |
| Orthogonal | A given construct has the same meaning in all contexts |
| Medium priority language properties | |
| Uniform | Constructs with the same meaning look the same |
| Conventional | The language follows accepted notations and conventions |
| Low priority language properties | |
| Restrictable | A user with limited knowledge can use the language |
| Simple | Programs are as simple as possible (but not simpler) |
| Translatable | It is easy to make a compiler for the language |
| Extensible | New functionality can easily be added to the language |
| General | The language can be used in a wide range of situations |
| Not applicable (see text) | |
| Precise | The meaning of a program is defined in all cases |

Table 3.1: Measures of language efficiency, taken from Louden (1993, p. 50ff). The measures are ranked according to their relative importance in the design of the NeuroGene gene language.

some measures may be considered more important than others. Our target audience is biologists with limited programming experience. This means that properties such as readability, concise syntax and expressiveness are important. Some requirements which are important for any programming language include orthogonality and uniformity. While we have tried to remain consistent with accepted notations and conventions, the language does also contain novel constructs which closely match the semantics of genes in nature.

The definition of the NeuroGene language is not given formally. Doing so would require a formal description of the underlying execution model of the language, i.e., the NeuroGene simulator as a whole, which also does not exist. The properties of the language is rather given in the form of NeuroGene as an authoritative implementation of a parser and execution model, which define the syntax and the semantics of the language, respectively. From such a definition, the language can be said to be precise, in that the meaning of any genome is defined by the effect of that genome on any cellular system during a NeuroGene simulation.

Restrictability is the language property that a user with a limited knowledge of the language is able to produce useful program code. Experience has shown us that more than a complete overview of the language, a good understanding of the NeuroGene execution model is invaluable in order to be productive in writing genes.

The genetic programming paradigm of the NeuroGene language is to our knowledge novel. As is commonly the case with programming language inventions, they are not fully formed at their inception, but need some time in use before all their implications become apparent. This should then lead to adjustments in the syntax and semantics of the language to make the language constructs more intuitive. We intend to continue the work on NeuroGene in general, and on the genetic programming language in particular. The work reported here has served to map out the required capabilities of the language — future work may involve simplifications of the language, both conceptually and implementationally.

### 3.1.5 Data types, variables and scoping

NeuroGene gene scripts may contain expressions and/or variables representing values of the following types: Real numbers, truth values (boolean), text strings and directions (i.e. vectors in 3D). Variables of these types follow the rules of lexical *scoping*. A variable's scope is the set of statements within the input script from which that variable may be referenced (Louden, 1993). Scopes are delimited by '{' and '}', just like in C, C++, Java and several other languages. Scopes may be nested, i.e. one scope, including the delimiting curly brackets, may be contained within another scope. The former is ***nested*** inside the latter, and the latter is an ***outer*** scope relative to the former. A variable defined within a given scope $S$ is accessible within $S$ from the point of definition onward, and also in any scope nested within $S$ to any depth. The variable is not accessible in scopes that are outer scopes to $S$. Two variables with the same name may not exist within the same scope. A variable in a nested scope may have the same name as a variable defined in an outer scope. The former will ***hide*** the latter within the nested scope, making the outer scope variable inaccessible where the nested scope variable is defined.

Symbols in the scripts may refer to genes or growth cone functions. These all have global scopes, i.e., all symbols defined in the script may be referenced from anywhere within that script. This is similar to the scope rules of functions in C and public functions in C++ and Java.

### 3.1.6 Parser implementation

The parser is implemented using JavaCC (Sun Microsystems, 2003c) which generates $LL(1)$ parsers with the following properties:

- Left-to-right (the first $L$ in $LL(1)$): The input is scanned from left to right, i.e., beginning with the first token in the input file, and proceeding sequentially to the last token in the file.
- Left-derivative (the second $L$ in $LL(1)$): This means that in each production, the terminal and non-terminal symbols in the production are handled in order starting with the left-most symbol and ending with the right-most symbol. Such parsers are also referred to as ***top-down***, since the parser starts with the start symbol of the grammar ($\langle sum \rangle$ in figure 3.1), and builds the parse tree from the root downward.
- Recursive descent: The parsing processes by which the parse tree is assembled is handled by a number of recursive functions, (at least) one for each production in the grammar.
- One token syntactic lookahead (the "1" in $LL(1)$): The parser processes the tokens in the input script in order from the first token in the file to the last. With one token lookahead the parser will only consider the first token not yet parsed when selecting the appropriate production in choice situations, for example, whether $p_0$ or $p_1$ should be used to expand $\langle sum \rangle$ in figure 3.1. In going from A to B, the correct choice is to use $p_0$, while going from B to C, $p_1$ is used.

JavaCC allows individual productions within the grammar to use more than one token lookahead, which means that the generated parser is ***locally*** $LL(k)$ with $k > 1$. This allows for greater freedom in designing the language, since a grammar which is ambiguous with lookahead of one may not be with a greater lookahead.

Semantic lookahead allows the programmer to write an arbitrary condition which must be satisfied for a given production to match the input. This allows the language to be

```
 1 gene Foo {
 2    regulation {
 3       if(internalConcentrationOf(Bar) > 0.1) {
 4          expressInternally(log(internalConcentrationOf(Bar)));
 5       }
 6    }
 7 }
 8 gene Bar {
 9    ...
10 }
```

Figure 3.5: An example gene script in which a gene is encountered for the first time in an embedded scope.

more precisely specified than what is possible with just a context-free grammar, including e.g. disallowing access to variables that are not defined.

Lexical scoping of symbols is implemented using a simple symbol-table consisting of a stack of hash tables. An empty hash table is pushed onto the stack when a '{' is encountered, and popped when '}' is encountered. In order to locate a given symbol, all hash tables currently on the stack are searched from the top downward, i.e. from the current scope outward, until the symbol is found. If all the hash tables are searched without finding the symbol, parsing fails.

Some symbols are always added to the global scope, even though they are encountered for the first time in a nested scope. This is used for symbols which represent genes and growth cone functions. This removes the need for forward declaration of genes[1], and allows the genes and growth cone functions to be included in the genome script in arbitrary ordering. See for example the genome shown in figure 3.5. Here, the expression rate of a gene Foo depends on the concentration of some protein Bar. The protein Bar is referenced by the *regulation* code of gene Foo on line 3. The gene Bar is added to the global scope of the symbol table during the parsing of line 3, even though the symbol was encountered in a nested scope. When the symbol Bar is encountered again on line 4, it exists in the symbol table, which ensures that the same symbol is referenced on lines 3 and 4. The properties of the gene Bar are then

[1]Forward declarations are required in C and C++ but not in Java. In C, if a function Foo calls a function Bar, and Foo for some reason precedes Bar in the source file, then a forward declaration of Bar must be included before the definition of Foo. The forward declaration of Bar includes the function signature only.

filled in during the parsing of the definition of the gene on lines 8–10 (details omitted here for clarity). If no gene called `Bar` is found in the file, a parser error is triggered after the entire gene file has been parsed.

### 3.1.7 Library function table

The actions and queries of the NeuroGene language follow the syntax of function calls, as shown throughout chapter 2. The library function table is used as a common system to verify the form of the function call and also to create the AST node instances which represent the function calls in the AST data structure.

The library function table is populated at application initialization, and does not change during the subsequent operation of the NeuroGene application. Each entry is the table consists of the following fields:

- The name of a function. This is the string which is used in the gene script to invoke the function call.
- A reference to the constructor used to allocate a new instance of the AstNode class implementing the function call.
- The context in which the function may be called (see below).

Reflection (Sun Microsystems, 2004, package java.lang.reflect) is used to generate new instances of AstNode classes representing function calls. The Java reflection system allows the use of references to constructors of Java classes (in this case AstNode classes). These references are used to create new instances of AstNode classes. The super-class of the class implementing a query defines the type of the query, e.g. all functions with a numeric return type extend (directly or indirectly) the abstract class FloatAstNode (see figure 3.2).

The library function table is used for two things: first it is used to verify whether a given function call is valid in a given context. This involves verifying that

- The function with the given name exists.
- The correct number of parameters are passed to the function and these are of the correct types. The required parameter list is determined from the parameters taken by the AstNode constructor instance.

- The return type of the function is appropriate.

This is verified during semantic look-ahead from within the parser. Once the validity of the function call has been established, the library function table is used to allocate the AstNode instance which implements the function call. The AstNode instance is initialized with the parameter values that were passed to the function call. In figure 3.4 all the nodes with heavy outlines are created using the LibraryFunctionTable system, while the remaining nodes are created directly by the parser. The library function table supports function overloading, i.e. multiple functions may share the same name, as long as they are distinguished by their valid contexts and/or parameter lists.

The advantages to this implementation are several. Firstly, it is possible to add new function calls to the NeuroGene language without changing the parser. Secondly, the library function table is a central repository for information about all the library functions of the language, allowing implementation of automatic help systems, etc. In the current implementation, the library function table may be run as a stand-alone application, and may be used to generate lists of defined library functions.

### 3.1.8 Grammatical contexts

As a consequence of the syntax of gene and growth cone function definitions, there are actions and queries which are appropriate only in certain parts of the input text. For example, it is appropriate to put a command for some morphological change (e.g. cell division) in the effects section of a gene, but not in the regulation section. Figure 3.6 shows an example of contexts in a sample gene script. To write a parser that generates parser errors when functions are referenced where they should not be, the concept of grammatical contexts has been introduced. Each function in the gene language is tagged with the grammatical contexts in which it may appear. Semantic lookahead is used to ensure that a function name will only match to a function if that function is valid in the context where the name is encountered. If the function is not allowed in that context, the matching fails, and a parser error results.

### 3.1.9 Mathematical functions

All the actions and queries given in chapter 2 are represented by entries in the library function table. In addition the table contains a number of functions which represent mathematical

```
(context is GENOME)
gene Foo {
   regulation {
      (push: context is REGULATION)
      if(internalConcentrationOf(Bar) > 1e-4){
         expressSurface(7.5e-2);
      }
   }
   (pop: context is GENOME again)
   effects {
      (push: context is EFFECTS)
      migrate(towardsSignal(Bar));
   }
   (pop: context is GENOME again)
}
```

Figure 3.6: Contexts changing through the parsing of a gene definition. The function call **migrate**() is defined as being valid in the EFFECTS context, but not in the REGULATION context. As it appears here, the function call is valid, however, if **migrate()** appeared in the REGULATION context, a parser error would result since the semantic lookahead would fail.

operations. These may be used to encode mathematical expressions in the genome script. Implemented functions are

- Trigonometric functions: Sine, cosine, tangents and their inverse functions, as well as the two-parameter inverse tangents. Trigonometric functions use degrees as unit of angles.
- Stochastic functions: Functions returning numbers with uniform probability distribution, Gaussian distribution, boolean stochastic values and 3D vectors with a predetermined length and stochastic direction. Also implemented a function for seeding the random number generator.
- Vector functions: Scalar ("dot") product and vector ("cross") product, and unit vector for computing a vector of unit length parallel to a given vector.
- Exponential functions: Natural exponent and logarithm, square root, and a function for evaluating $x^y$.

- Rounding: Functions for rounding off, up or down to the nearest whole number.
- Other: Also included are functions for computing the absolute value, and the largest and smallest of two numbers.

Many of these functions are implemented by calling functions in the Java Math class, and have identical properties to the Java implementation of the corresponding functions. The implementation of these functions within the NeuroGene language is therefore trivial, see the class `org.neurogene.parser.ast.MathAstNode`.

### 3.1.10 The geometry scripting language

The geometry scripting language is similar to the gene scripting language, and it uses much of the same infrastructure (including the parser). Compared to the gene language, the geometry scripting language supports additional data types. Variables may refer to cell components, which makes it possible to construct complex neural networks directly at simulation initialization. Activity sources and filters (see section 2.7.5) may also be referenced by variables in order to build signal processing systems for simulated exogenous neural activity.

## 3.2 System architecture

The NeuroGene system consists of two major components, a simulation system, as described in chapter 2, and a GUI front-end controlling the simulation. The GUI front-end allows the user to retrieve information about the current state of the simulation. The system has been designed in modules that interact through simple interfaces. See figure 3.7 for a UML diagram of the classes that make up the NeuroGene graphical user interface and control infrastructure. All the data structures and functionality described in chapter 2 are implemented within the module labeled "Simulation" in figure 3.7. The front end described here controls the simulation by calling `Simulation.doStep()`, which in turn triggers the six stages of a simulation time step as described in section 2.10.7.

The NeuroGene system architecture went through a major re-design during the winter of 2003/2004. Eric DeWitt of Gary Marcus' lab made significant contributions to this process, through discussions and suggestions. The large majority of the work implementing these changes was done by the author. The re-design involved mainly the components shown

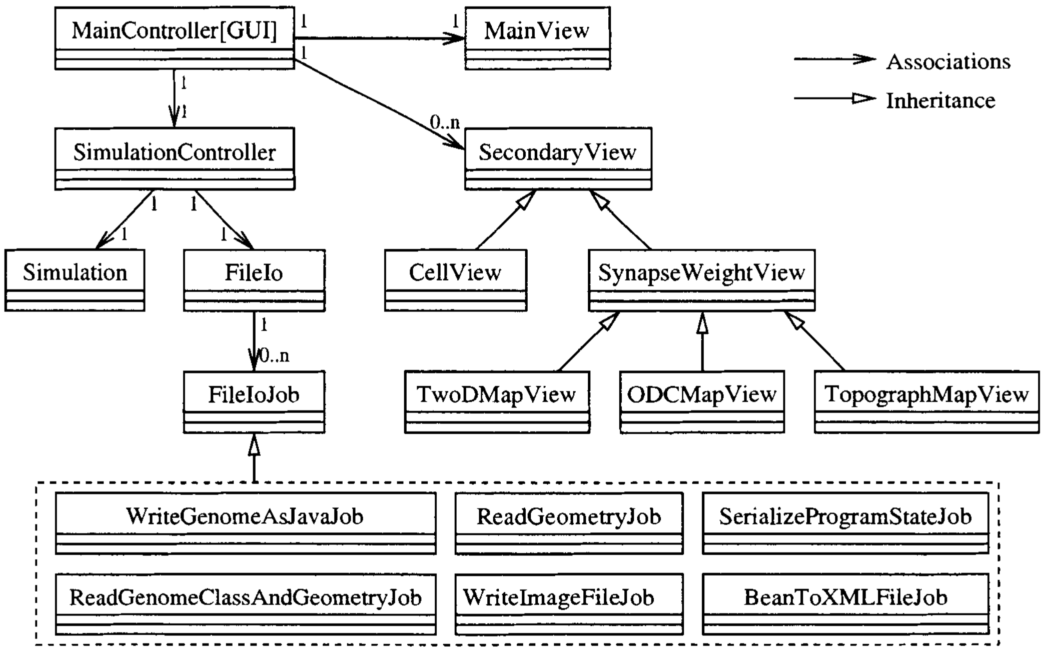


Figure 3.7: UML diagram of the NeuroGene system. The simulation system (described in detail in chapter 2) is here shown as a simple entity labeled "Simulation". This entity represents the entire system shown in figure 2.8.

in figure 3.7, including their interactions through various design patterns outlined below. It established the layered system architecture and the unified approach to communication between these layers. The simulation system itself was not significantly altered.

### 3.2.1 The Observable pattern

The architecture is arranged in layers, where components in upper layers depend on specifics in lower layers, but where the lower level components are independent of those at in upper layer. The top layer consist of the MainController, below which is the SimulationController, then the Simulation below that. They are independent in the sense that the upper layer component can be radically altered or even replaced without needing to make any changes to the lower level component.

When an upper layer needs to communicate to a lower layer (e.g. MainController tells SimulationController to change the simulation from being "running" to being "paused"), this is straightforwardly implemented by having the MainController call a function in the SimulationController instance. This means that the MainController implementation does depend on the details of SimulationController. In order to simplify the design of the system, this form of dependence is avoided in the opposite direction. This is achieved through the user of the "Observable" pattern. In this pattern an upper layer component (the *observer*) registers itself with the lower level component (the *observable*). The observable sends notification of *events* to all registered observers. The observable maintains a list of all the registered observers. Registration normally occurs at application initialization, but can also happen e.g. when a new view is created.

For example, the SimulationController maintains a list of TimeStepListeners — these are objects that have registered themselves with the SimulationController in order to receive notification about when simulation time steps are completed. In order to be registered, these objects must implement a Java interface called TimeStepEventListener (*listener* being a common synonym for *observer*). This interface defines a single function

```
public abstract void timeStepCompleted(TimeStepEvent event)
```

The list contains among other objects the MainController, however, the SimulationController does not need to be "aware" of this. When a time step is completed, the SimulationController generates a TimeStepEvent, and calls timeStepCompleted() on each TimeStepListener in the list, passing the event object as argument. The TimeStepEvent contains details about the event (e.g. how many time steps have been completed so far in the simulation, etc.) which can be retrieved by the various listeners according to their needs.

For each type of event, an event class and a listener interface is defined. See table 3.2 for a list of all event types generated by the simulation. This pattern is used for all communication from Simulation → SimulationController, FileIo → SimulationController and SimulationController → MainController. Several additional types of event/listener pairs are used within the GUI system, such as from SecondaryView instances → MainController.

To demonstrate the versatility of this pattern, the following mechanisms are all implemented using the TimeStepListener observer pattern:

- Update of the GUI (see figure 3.8) to show the current time step counter and duration of last time step.

| Event | Cause |
|---|---|
| ExceptionEvent | An error has occurred in the simulation |
| FileIoEvent | File I/O is complete or has been interrupted |
| FileIoProgressEvent | File I/O in progress, used to generate user feedback |
| RunStateChangedEvent | Simulation has gone from e.g. running to paused |
| SimulationEvent | Change in some parameter of the simulation |
| TimeStepEvent | Simulation time step has completed |

Table 3.2: The types of events generated by the simulation. For each of these events there is a corresponding listener (or observer) interface implemented by all classes that need to receive the events. Each event object contains information describing the event. E.g. the TimeStepEvent contains the number of time steps computed so far, the ExceptionEvent contains the cause of the error, etc. FileIoProgressEvents are used to generate messages of the type "10% complete", "20% complete", etc.

- Update GUI views of various kinds to reflect changes in the state of the simulation system as simulations progress.
- Automatic file saving every N time steps, including saving the simulation state and saving GUI views to image files.
- Time dependent exogenous neural input filters, e.g. caching filter which clears the cache at the end of each time step, and strobe filter which changes from bright to dark, etc. (see appendix D).

All these mechanisms are implemented without altering the simulation system itself in any way.

### 3.2.2 Visitor pattern

This pattern is used for display and analysis of the simulation state. The visitor pattern allows a complex data structure, like the complete simulation system, to be traversed without the entity that does the traversal (the *visitor*) needing to know how that data structure is constructed. The visitor implements an interface with functions like e.g.

```
public void visit(WorldNode node)
public void visit(CellComponent cell)
```

The classes which make up the simulation data structure ensures that these functions are called for each world node and each cell component in some order. The visitor may retrieve and store information from some or all of these objects in order to do some analysis of the simulation data structure, or it may paint an image of the simulation on the screen, etc.

This pattern is used with the same purpose as the Observable pattern above, which is to make the simulation data structure independent of the details of the display and analysis code which it interacts with. In this case either the simulation system or the visitors may each be radically altered without needing to change the other. New visitor classes may also be created and incorporated into the NeuroGene system, thereby extending the visualization or other capabilities of the system, without having to alter the simulation system.

## 3.3 Graphical user interface

The NeuroGene graphical user interface is constructed using the Model-View-Controller (MVC) pattern. This pattern is used primarily in user interfaces. The software system is divided into components which each plays one of the following three roles:

### Model

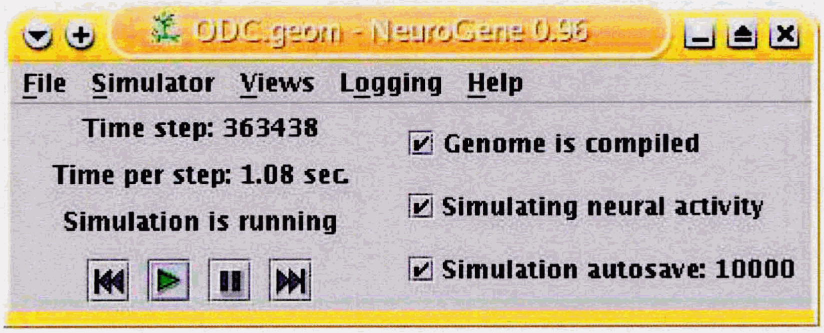
Maintains the internal state of whatever the GUI allows the user to interact with. In the context of NeuroGene, the main model will be the simulation itself, but other components may also play the model role.

### View

Displays data from the model. Also allows the user to interact with the model, and contains visible components (buttons, drop-down menus, etc.) allowing the user to issue commands to modify the state of the model.

### Controller

Coordinates the interactions between the model and the view, e.g. triggering updates of the view in response to changes in the state of the model, and issues commands to the model in response to user interactions.



Time step: 363438

Time per step: 1.08 sec.

Simulation is running

☑ Genome is compiled

☑ Simulating neural activity

☑ Simulation autosave: 10000

Figure 3.8: The NeuroGene main GUI view.

### 3.3.1 GUI main controller

The main controller coordinates the users interaction with one main view (a control panel) and any number of secondary views (showing cells, diagrams representing synaptic connectivities, etc., see the various views in figure 3.7) and the main model, represented by the simulation controller.

### 3.3.2 GUI main view

The main view (see figure 3.8 ) is a control panel which shows the most important information regarding the state of the simulation, and allows the user to control the simulation. Drop-down menus allow the user to open secondary views showing more specific information about the simulation.

### 3.3.3 Secondary views

Secondary views are used to display the current state of the simulation. The most important type of view is the cell view, which displays the cells currently being simulated (see numerous examples in chapter 5). This view allows for the visualization of protein concentrations, both extracellular, intracellular and surface concentrations. It also allows zooming, panning and rotation of the cellular assembly using mouse gestures ("click-drag").

Other views visualize synaptic connectivities among cells. The views are written according to the JavaBean coding standard as outlined in appendix E. This allows us to use a general property sheet (Taylor, 2001) to show and modify the properties of each view, which makes it simple to add new views to the system as the need arises.

## 3.4 Performance

NeuroGene has been optimized for execution speed. This optimization has been carried out using Java profiling tools (Shirazi, 2000), which measure the time spent running different parts of the NeuroGene code. A profiler is well suited to find performance bottlenecks, allowing the user to focus on performance bottlenecks where improvements to computation speed will have the greatest impact on the overall performance.

We have also built-in profiling functionality into NeuroGene itself. NeuroGene has the capacity to measure the time it spends on the various stages of a simulation time step. It can also measure the time which is spent on each of the different genes in the genome. These features give the user information that may help in improving the speed of simulations by altering the genome.

### 3.4.1 Java code generator

The most powerful performance enhancement of the NeuroGene simulation is the ability to translate a genome into Java code, which can be compiled to a class file. This java class can then replace the AST data structure as the mechanism for executing the genome. The speed advantage to doing this is significant. Depending on the nature of the simulation the speed may be improved by as much as an estimated 25-30%. This gain is speed arises from the removal of the overhead involved in traversing the AST structure. Using the java package InstantJ (Meier, 2002), the NeuroGene application can itself compile the generated Java code, giving on-the-fly Java code generation, compilation, and installation of the compiled class into the NeuroGene system.

## 3.5 File I/O

The file input/output system in NeuroGene is accessed through a single class called FileIo (see figure 3.7). This class mediates access to a number of other classes which implement the actual file operations, according to the "Façade" pattern (Grand, 1998). The various file formats used for input and/or output are listed below:

### Gene and geometry files

These files are written in the NeuroGene scripting languages for specification of genomes and initial simulation geometries. Files of both types are read using the NeuroGene parser, see section 3.1.

### Compiled genome files

As outlined above, NeuroGene can save the current genome to a file in the form of Java source code. NeuroGene can also load a compiled Java .class file containing the definition of the genome. The generation of the java file and compilation can also be done on the fly, in which case file I/O is not involved.

### Program state files

The current state of the NeuroGene simulation can be saved to file. To accomplish this, the Java mechanism of serialization (Sun Microsystems, 2004) is used. To save hard-drive space, these large files are compressed using the *gzip* algorithm (Sun Microsystems, 2003a,b) as they are written to disk, typically achieving about 75% compression.

### Image files

Graphical images created by NeuroGene can be saved to file using the file formats PNG (Portable Network Graphics, PNG development group, 2004), a raster image file format with lossless compression similar to GIF, or EPS (Encapsulated Postscript), a vector image file format, using EpsGraphics2D (Mutton, 2004). PNG is appropriate for images that are going to be used on the screen, and may be used to create animations. EPS files are better suited for printing. All images of simulations used in this thesis were generated directly from NeuroGene and saved in EPS format. Graphical images of NeuroGene simulations may be automatically saved to file at set intervals. This is used for creating animations showing the progress of the simulations, and also allows the user to easily monitor the progress of the simulation.

### Image autosave files

It can often be cumbersome to set up the various views to be autosaved to file at regular intervals. To simplify simulation setup, a set of such image autosave jobs may themselves be written to, or read from, a file. This is done using the java.beans.XMLEncoder class which produces a description of a Java class using the XML file format (DuCharme, 1999). Using this mechanism requires that the views are written according to the JavaBean coding standard, see appendix E. The resulting files are human readable and may be altered using a simple text editor.

## 3.6 Summary

The NeuroGene language used to encode genomes have been designed to follow widely accepted conventions while also introducing novel constructs which represent aspects of simulated biological mechanisms. These constructs are not available in other programming languages, which necessitates the implementation of the genetic language in the first place. The language is implemented using modern techniques, including automatic parser-generator, which takes as input a context-free grammar describing the system, and generates as output a Java-class capable of parsing strings in that language and generating AST representations of those strings.

While the NeuroGene simulation system is designed for speed, the GUI and simulation control system has been designed to be highly modular, anticipating that alternative systems may be developed in the future. Through the use of the visitor and observer patterns and JavaBean based tools, the different components of the GUI are decoupled and less interdependent. This makes each component simpler, and also makes it easier to modify and enhance the system in the future. Some such enhancements, such as the addition of new view components, are anticipated in the current design. Both the cellular view and those for visualizing neural interconnections are highly customizable. By virtue of having been written according to the JavaBean standard, additional functionality is accessible, such as saving view configurations to file, etc.

This completes the description of the NeuroGene implementation. In the following two chapters, experimental results obtained from NeuroGene will be presented. The next chapter presents results which validate the correctness of the implementation of simulations of various physical processes.

# Chapter 4

# Results I: Verification of algorithms and implementation

NeuroGene simulates a number of physical processes, including diffusion, decay and ligand-receptor binding. In order to have confidence in the results of NeuroGene simulations, it is important that these simulated physical processes give results that are in close agreement with what would be observed in nature. In this chapter I will outline the physical laws governing these processes, report the numerical results as computed by NeuroGene, and discuss the level of agreement between the two in an effort to establish the overall accuracy of NeuroGene simulations with respect to physical processes.

The physical environment as simulated in NeuroGene is discrete (see section 2.8.2), while 3D space in nature is continuous[1]. Several measures have been taken within NeuroGene to simulate a continuous environment, notably relating to how protein concentrations are perceived by cell components. In this chapter I will also discuss these measures and report their performance.

In this chapter, the results of simple NeuroGene simulations will be presented. These

[1]The physical/philosophical discussions as to whether space and time are discrete or continuous need not concern us here. According to these discussions, the smallest indivisible unit of time and space, if they exist, are much smaller than the scale of time and space at which biological development can reasonably be described.

simulations generate numerical data which can be compared with the physical processes simulated by NeuroGene. This allows us to estimate the precision and accuracy of the NeuroGene implementation, particularly with respect to the simulation of physical and chemical processes. Sections 4.1 and 4.2 deal with how cell component perceive extracellular concentrations and concentration gradients. Sections 4.3 and 4.4 deal with the chemical processes of first-order decay of proteins and of equilibrium receptor-ligand binding. Section 4.5 presents data generated by NeuroGene's simulated diffusion computations under various conditions, and compares it to the solutions to the diffusion equation as derived in appendix F. Finally, section 4.6 outlines the techniques used to verify the implementations of the parser, AST data structure, gene and growth cone simulations and neural models.

## 4.1 Measuring extracellular concentrations

In nature, the concentration of a protein at a particular point $P$ in 3D space within a developing organism is best defined as the number of molecules of the protein in a small volume centred at $P$, divided by the size of that volume in the limit where the size of the volume approaches zero. In NeuroGene, 3D space is discretized into a 3D array of equally sized cubic elements known as world nodes. The concentration of each protein is stored explicitly within each world node, and signifies the number of molecules within the world node divided by the size of the world node.

The concentration at an arbitrary point $P$ within that array might be defined as the concentration stored in the world node which contains $P$. However, this leads to ***aliasing effects*** in the common case where two adjacent world nodes contain different concentrations of the protein: Consider a cell $C$ which moves slowly through the 3D array of world nodes, measuring the concentration of the protein as it goes. As long as $C$ stays within one world node, it will always measure the same concentration value — however, as it moves into the adjacent world node, the measured concentration will suddenly change to a new value. In NeuroGene, such jumps in measured concentration values are avoided using linear interpolation in 3D. Concentration values at general points $(x, y, z)$ are computed from the values in the world nodes at points $(X, Y, Z)$[2] close to the point $(x, y, z)$ using linear interpolation.

[2]In the following lower case symbols $x$, $y$, ... represent real values and upper case symbols $X$, $Y$, ... represent integer values.

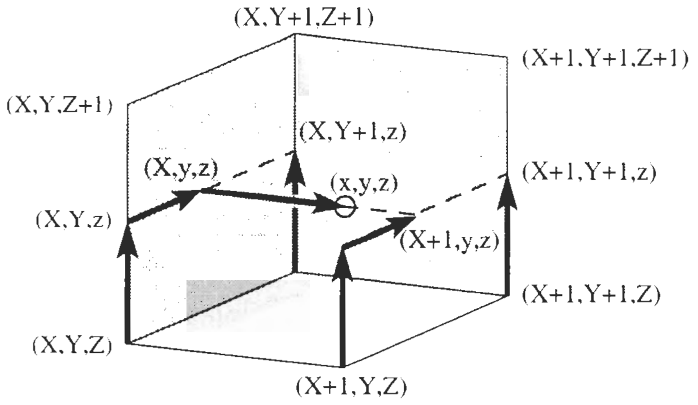


Figure 4.1: Computing the concentration at a point $(x, y, z)$ (circle) by successive interpolation in 3D. The vertices of the cube shown coincide with the centres of the eight WorldNodes surrounding the point $(x, y, z)$. First four interpolations in the vertical (Z) direction, then two in the Y direction and lastly one in the X direction.

Higher degree polynomial interpolation was also tried, but they were found to be poorly suited for this purpose. In particular, concentration gradients established through diffusion tend to have exponential shapes, which are poorly matched by polynomial functions.

With known concentration values $f(X)$ and $f(X+1)$, with $X$ representing locations at the centre of world nodes, the concentration value $f(x)$ at some non-integer location $x$ with $X \leq x < X+1$ (i.e. $X = \lfloor x \rfloor$) is computed using simple linear interpolation

$$f(x) \quad = \quad f(X) + (x - X)\left(f(X+1) - f(X)\right) \tag{4.1}$$

In 3D, three such computations are carried out in sequence (Press et al., 1986) to give the concentration at $f(x, y, z)$ from the values at eight points surrounding $(x, y, z)$, i.e. $f(X, Y, Z)$, $f(X+1, Y, Z), \ldots, f(X+1, Y+1, Z+1)$. Interpolation between $f(X, Y, Z)$ and $f(X, Y, Z+1)$ gives the value of $f(X, Y, z)$ with $Z \leq z < Z+1$, see figure 4.1. Four such interpolations are done to compute $f(X, Y, z)$, $f(X+1, Y, z)$, $f(X, Y+1, z)$ and $f(X+1, Y+1, z)$. Interpolation between $f(X, Y, z)$ and $f(X, Y+1, z)$ gives the value of $f(X, y, z)$ with $Y \leq y < Y+1$, and similarly, interpolation between $f(X+1, Y, z)$ and $f(X+1, Y+1, z)$ gives the value of $f(X+1, y, z)$. In the last interpolation these give the value of $f(x, y, z)$.

Near the edges of the simulations space, some of the data points involved in the interpolation will be missing, since the corresponding WorldNodes fall outside the simulated

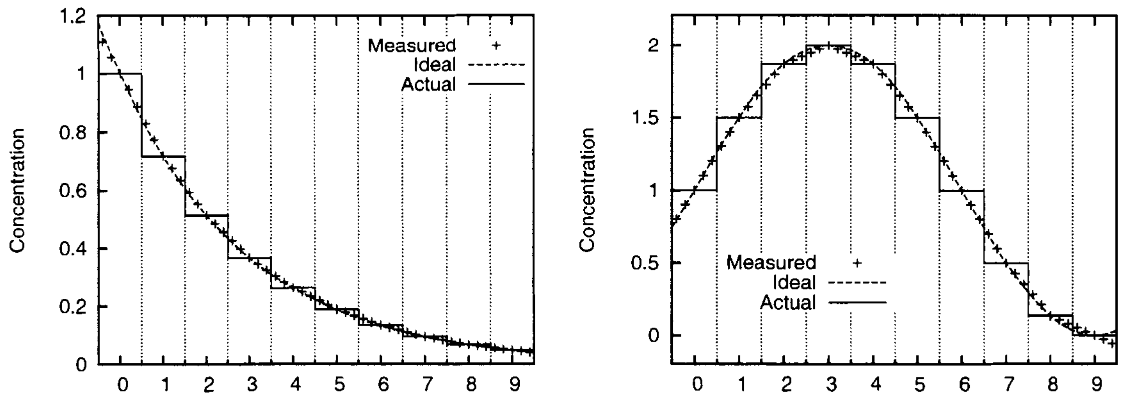


Figure 4.2: Protein concentration values computed by NeuroGene from two different concentration gradients, $u(x) = \exp\left(-\frac{1}{3}x\right)$ (left) and $u(x) = 1+\sin\left(\frac{1}{6}\pi x\right)$ (right). Vertical grid lines indicate boundaries between world nodes.

space. In these cases extrapolation is used, based on available data points from within the simulated space. If the simulation space has the length $L$ (always an integer), then the leftmost concentration value (i.e. the leftmost world node) is at $x = 0$ and the rightmost at $x = L - 1$. However, cell components can have positions in the range from $x = -\frac{1}{2}$ to $x = L - \frac{1}{2}$, meaning that concentration values may need to be computed at such locations too. In these cases extrapolation is used in stead, so that concentration values in the range from $x = -\frac{1}{2}$ to $x = 0$ are computed by linear extrapolation from the values at $X = 0$ and $X = 1$.

Figure 4.2 shows the extracellular concentration of a protein, as measured by NeuroGene the method outlined above (crosses), compared with the mathematical function used to create the concentration gradient (broken line) and the actual values stored in the world nodes (solid line). As can be seen in the figure, the computed protein concentration reproduces the mathematical function used to compute the concentrations with high accuracy. However, an artifact at the right edge of the right hand figure shows how the extrapolation fails to follow the function. This is because the last data-point used in the extrapolation is at $x = 9$. The gene and geometry scripts used to initialize this experiment are listed in appendix B.1.1.

In conclusion, 3D linear interpolation is effective in overcoming the aliasing problems introduced by the discrete representation of protein concentrations in the simulation space,

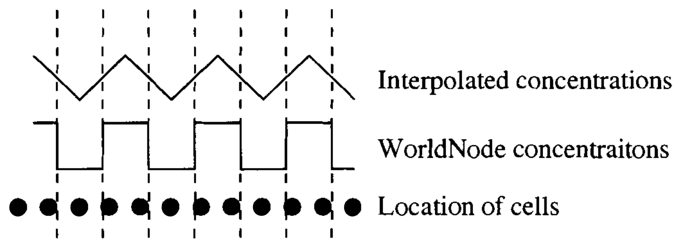


Figure 4.3: Aliasing problem deriving from the production of extracellular protein. A 1D array of equidistant cells each express a signal at the same rate. Some world nodes contain two cells and receive twice as much protein as other world nodes containing only one cell. Interpolation of these world node concentrations does not recover the ideal uniform protein concentration.

and represents a significant improvement over using the values as stored in the world nodes directly. Note that interpolation does not solve the aliasing problem deriving from the production of protein, as demonstrated in figure 4.3. This problem arises from the fact that all extracellular protein produced by a cell component is added to the world node containing that cell component, regardless of where within the world node the cell component in located.

### 4.1.1 Extracellular concentration of membrane bound signals

The methods outlined above can be used to compute the concentrations of membrane bound proteins as well as soluble ones. The extracellular concentration of some membrane bound protein $P$, as measured by some cell component $C$, is related to the amount of the protein $P$ which is bound to the surface of cell components in the vicinity of $C$, not counting that which is bound to $C$ itself.

There are two possible definitions of this concentration: Either the concentration is in amount of protein per unit volume of extracellular space, or it is the amount per unit surface area of the surrounding cell components. I will refer to these alternatives as the *volume* and *surface* models, respectively. In the following, these two alternatives are compared for suitability.

Consider the case where a number of axons invade a region of space initially occupied by a single cell component carrying a high concentration of some protein $P$. Under the surface model, the invasion of axons, which are low in $P$, will cause the measured concentration of

$P$ in the vicinity to drop. Such a drop will not occur in the volume model.

Consider the same situation, except that the invading axons express another protein $Q$ on their surface, all axons carrying the same concentration of $Q$. Under the volume model, as increasing numbers of axons arrive, the measured concentration of $Q$ will increase. Under the surface model it will stay the same as more axons arrive, since the total surface area and total amount of protein increase by the same amount.

While either model may seem reasonable, the volume model was finally chosen. This decision was based on observations in a simulation of topographic projections, where invading axons needed to accurately read a concentration gradient of membrane bound proteins carried on tectal cells (see chapter 5). We found that the surface model interfered with this reading, since the concentration values would depend on the number of invading axons in the vicinity of the one reading the concentration value — this greatly hampered the simulation. The volume model does not have this problem.

A third possibility is to let the choice of model be up to the user. This is possible, but since we have not yet encountered a situation in which the surface model is superior to the volume model, only the volume model is currently available.

## 4.2 Measuring extracellular concentration gradients

Many developmental mechanisms rely on protein concentration gradients. There is evidence that growth cones (Baier and Bonhoeffer, 1992; Goodhill, 1998) can directly detect the direction and magnitude of such concentration gradients. The NeuroGene gene language contains queries for directly measuring concentration gradients. The results of these computations is a vector in 3D space, representative of $\nabla f(x, y, z)$ but normalized for overall concentration. In order to compute the $x$ component of this gradient vector, a function $g_x(x, y, z)$ is defined as follows

$$g_x(x,) \quad = \quad \frac{f(x+\frac{1}{2}, y, z) - f(x-\frac{1}{2}, y, z)}{f(x+\frac{1}{2}, y, z) + f(x-\frac{1}{2}, y, z)} \tag{4.2}$$

The denominator here ensures that the resulting vector represents the gradient relative to the overall magnitude of the concentration. Using the relative rather than absolute difference in concentration means that a magnitude of the computed gradient will be constant along a gradient with an exponential shape (see figure 4.4 left). This is a desirable property given that gradients established by diffusion tend to have exponential shapes.

Divide-by-zero problems are avoided by defining

$$g_x(x, y, z) = 0 \quad if\ f(x + \frac{1}{2}, y, z) = f(x - \frac{1}{2}, y, z) = 0 \tag{4.3}$$

$g_x(x, y, z)$ represents the change in concentration between adjacent WorldNodes along the $x$-dimension. Similar function $g_y(x, y, z)$ and $g_z(x, y, z)$ are used to compute the other two components of the vector:

$$g_y(x, y, z) = \frac{f(x, y + \frac{1}{2}, z) - f(x, y - \frac{1}{2}, z)}{f(x, y + \frac{1}{2}, z) + f(x, y - \frac{1}{2}, z)} \tag{4.4}$$

$$g_z(x, y, z) = \frac{f(x, y, z + \frac{1}{2}) - f(x, y, z - \frac{1}{2})}{f(x, y, z + \frac{1}{2}) + f(x, y, z - \frac{1}{2})} \tag{4.5}$$

Since $f(x, y, z)$ is only known at integer values of $x$, $y$ and $z$, it follows that $g_x(x, y, z)$ is only known at integer values of $y$ and $z$ and half-integer values of $x$. In order to compute $g_x(x, Y, Z)$ where $x$ is any real value and $Y$ and $Z$ are both integers, an interpolation must be done between the values $g_x(X - \frac{1}{2}, Y, Z)$ and $g_x(X + \frac{1}{2}, Y, Z)$. Here $X$ is chosen so that $X - \frac{1}{2} \leq x < X + \frac{1}{2}$, i.e. $X$ is $x$ rounded to the nearest whole number. Interpolation along the $y$ and $z$ dimensions proceed as described for measuring extracellular concentrations above. This means that $g_x(x, y, Z)$ is computed by linear interpolation of the values $g_x(x, Y, Z)$ and $g_x(x, Y + 1, Z)$, with $Y = \lfloor y \rfloor$, and $g_x(x, y, z)$ by interpolation of $g_x(x, y, Z)$ and $g_x(x, y, Z + 1)$ with $Z = \lfloor z \rfloor$. The $y$ and $z$ components of the gradient are similarly computed by interpolations of the functions $g_y(x, y, z)$ and $g_z(x, y, z)$ respectively.

Figure 4.4 shows two different concentration gradients, an exponential gradient on the left and a sine on the right. See appendix B.1.1 for the scripts used to set up this experiment. In each figure, the gradient shape is shown using solid lines. The gradient slopes solved symbolically using

$$U(x) = \frac{c}{u(x)} \frac{du(x)}{dx} \tag{4.6}$$

are shown in broken lines. The factor $c$ is a scaling value chosen to match the measured data in magnitude. Finally, the gradient slope as measured by NeuroGene using the method outlined above is shown with crosses. The measured data shows good agreement with the ideal values. The adjustment for overall magnitude gives the result that the gradient of an

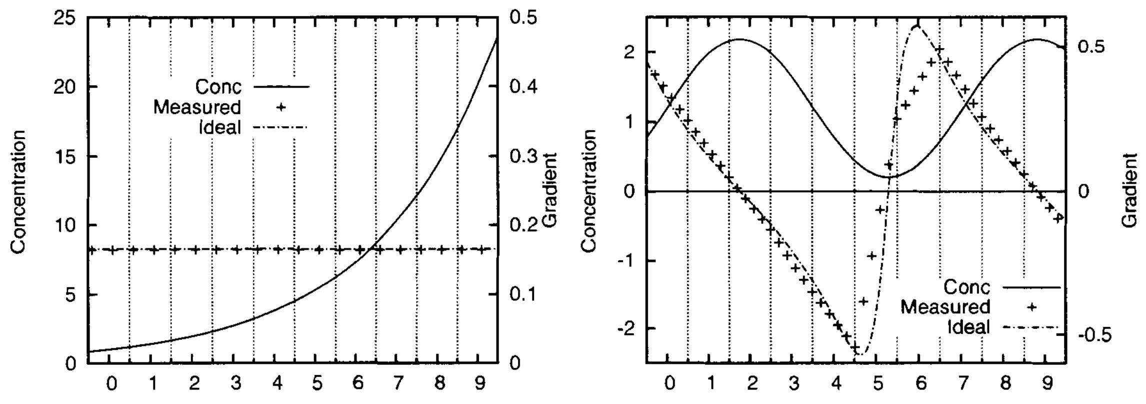


Figure 4.4: Computed protein concentration gradients from shown concentration profiles. Concentrations (labeled "Conc") are governed by the functions $u(x) = \exp(\frac{1}{3}x)$ (left) and $u(x) = 1.2 + \sin(\frac{17}{60}\pi x)$ (right). Vertical grid lines indicate the boundaries between world nodes.

exponential function is constant (left). The interpolation involved in computing the vector components can be spotted near $x = 6$ in the right hand image, where a sharp maximum in the symbolically solved gradient function is missed by the interpolation routine. Note also that the interpolation is done between points that fall on half-integer values of $x$, since the gradient values that form basis for the integration are known at the boundaries between adjacent world nodes.

Figure 4.5 shows the direction and magnitude of the protein concentration gradient vectors computed from a concentration function of the form

$$u(x,y) = \sqrt{R^2 - x^2 - y^2} \tag{4.7}$$

This function has circular symmetry about the origin, meaning that concentration gradient vectors should always be pointing toward the origin. From figure 4.5 it can be verified using a ruler that all the vectors are indeed pointing toward the origin, accurately reflecting the variation of this function. The gene and geometry scripts used for this experiment are listed in appendix B.1.2.

The symbolically computed data in figure 4.5 (marked "ideal") was generated from the

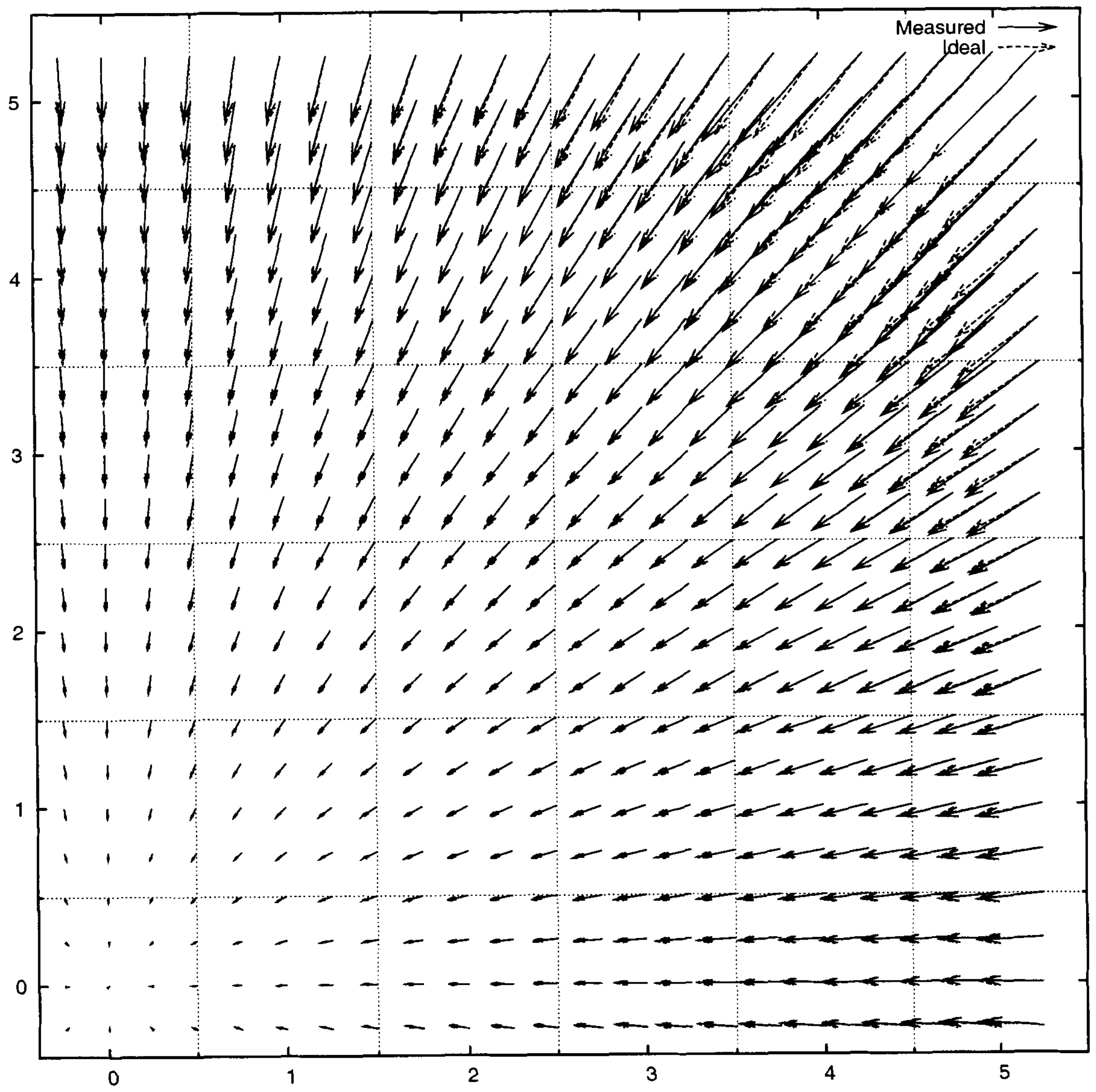


Figure 4.5: Vector field representing the computed concentration gradient vectors across a simulation space with a simple spherically symmetrical concentration distribution. Values as measured with NeuroGene (solid arrows) and computed symbolically (broken lines) are shown, although in most places these coincide too closely to be separated. Grid lines show the boundaries between world nodes. Note that a scaling factor is used, similar to the one in equation 4.6.

equation

$$\vec{U}(x) = c\left[\frac{\partial u(x,y)}{\partial x}, \frac{\partial u(x,y)}{\partial y}\right] \tag{4.8}$$

While data measured with NeuroGene was adjusted for overall concentration (see equation 4.6), the symbolically computed data was not. It can be seen that this adjustment affects the magnitude of the vectors (solid and broken arrows differ slightly in length), but that the direction of the vectors are only affected to a small degree (arrows are always very close to parallel).

In summary, the concentration gradient components computed by NeuroGene closely match the first derivative of the concentration with respect to spatial dimensions, subject to some artifacts arising from the interpolation. No systematic error is observed in the computed gradient vectors close to the edges of the simulation space or anywhere else.

## 4.3 Simulated decay

Proteins in NeuroGene are subject to simulated first-order decay. The decay rate of a protein is quantified by the rate constant $k$, and the evolution of the concentration $C$ of some protein is defined by

$$\frac{dC}{dt} = -kC \tag{4.9}$$

with $k \geq 0$. It can be seen that $k$ has the unit of $sec^{-1}$ or *per second.* Within NeuroGene, the unit for this quantity should in stead be *per simulation time step.* This means that the concentration $C^n$ in time step $n$ is

$$C^n = C^0 \exp(-kn) \tag{4.10}$$

where $C^0$ is the initial concentration at $n = 0$. In the implementation, this is simplified to give

$$C^{n+1} = C^n \exp(-k) + C^{n+1}_{add} \tag{4.11}$$

where $C^{n+1}_{add}$ is the amount of the protein added during time step $n+1$. Figure 4.6 shows the concentration as function of time as computed by NeuroGene. There is close agreement between the values are computed by NeuroGene (squares) and the symbolic solution (solid line). See appendix B.1.3 for the relevant scripts.

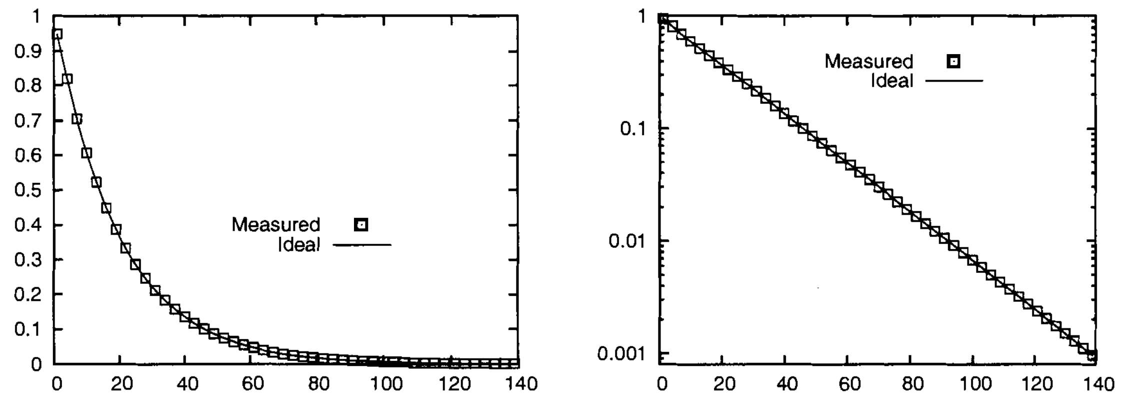


Figure 4.6: Decay of protein with decay rate 0.02 and initial concentration 1.0. Both figures represent the same data and differ only in the scale used for the $y$-axis, linear on the left and logarithmic on the right.

## 4.4 Receptor ligand binding

In order to verify the receptor-ligand binding simulation (see section 2.4.4), the following simple simulation was set up: Three proteins are defined, called Ligand (which is soluble), Receptor (which is membrane bound) and Complex (also membrane bound). These are defined to be related through a receptor-ligand binding relationship, with a association constant $K_d = 10^{-2}$M. The simulation space consists of a single world node, which initially contains no protein, i.e. $L = 0$ at $t = 0$. In this world node there are two somas $A$ and $B$. These are initially given a certain amount of receptor (0.02 for soma $A$ and 0.1 for soma $B$) on their surface. As the simulation progresses, an increasing amount of Ligand is added to the world node. The total level of receptor (i.e. both ligand-bound and free) in both cells is never altered. The gene and geometry scripts used to initialize this experiment are listed in appendix B.1.4.

The concentration of Ligand in the world node ($L$) and Receptor and Complex at both cells ($R_A$and $C_A$ for soma $A$, $R_B$ and $C_B$ for soma $B$) were then recorded as more Ligand was added to the world node. The results are shown for selected time steps in table 4.1. The gene and geometry scripts used are listed in appendix B.1.4. From this table it may be verified that the following properties of receptor-ligand binding are accurately simulated by

| | Soma A | | Soma B | |
|---|---|---|---|---|
| Ligand ($L$) | Receptor ($R_A$) | Complex ($C_A$) | Receptor ($R_B$) | Complex ($C_B$) |
| 0 | 0.0200 | 0 | 0.100 | 0 |
| $1.71 \times 10^{-12}$ | 0.0200 | $3.41 \times 10^{-12}$ | 0.100 | $1.71 \times 10^{-11}$ |
| $1.38 \times 10^{-11}$ | 0.0200 | $2.76 \times 10^{-11}$ | 0.100 | $1.38 \times 10^{-10}$ |
| $1.11 \times 10^{-10}$ | 0.0200 | $2.21 \times 10^{-10}$ | 0.100 | $1.11 \times 10^{-9}$ |
| $1.77 \times 10^{-9}$ | 0.0200 | $3.54 \times 10^{-9}$ | 0.100 | $1.77 \times 10^{-8}$ |
| $1.14 \times 10^{-8}$ | 0.0200 | $2.83 \times 10^{-8}$ | 0.100 | $1.41 \times 10^{-7}$ |
| $1.13 \times 10^{-7}$ | 0.0200 | $2.26 \times 10^{-7}$ | 0.100 | $1.13 \times 10^{-6}$ |
| $1.81 \times 10^{-6}$ | 0.0200 | $3.62 \times 10^{-6}$ | 0.100 | $1.81 \times 10^{-5}$ |
| $1.45 \times 10^{-5}$ | 0.0200 | $2.90 \times 10^{-5}$ | 0.0999 | $1.45 \times 10^{-4}$ |
| $1.17 \times 10^{-4}$ | 0.0198 | $2.32 \times 10^{-4}$ | 0.0988 | $1.16 \times 10^{-3}$ |
| $4.85 \times 10^{-4}$ | 0.0191 | $9.23 \times 10^{-4}$ | 0.0954 | $4.62 \times 10^{-3}$ |
| $1.01 \times 10^{-3}$ | 0.0182 | $1.84 \times 10^{-3}$ | 0.0908 | $9.19 \times 10^{-3}$ |
| $5.47 \times 10^{-3}$ | 0.0129 | $7.07 \times 10^{-3}$ | 0.0646 | 0.0354 |
| 0.0179 | $7.16 \times 10^{-3}$ | 0.0128 | 0.0358 | 0.0642 |
| 0.223 | $8.60 \times 10^{-4}$ | 0.0191 | $4.30 \times 10^{-3}$ | 0.0957 |
| 0.986 | $2.01 \times 10^{-4}$ | 0.0198 | $1.00 \times 10^{-3}$ | 0.0990 |

Table 4.1: Simulation of ligand-receptor binding by two somas (A and B) competing for the same reservoir of ligand.

NeuroGene:

### Mass balance

No protein is unaccountably added to or removed from the simulation. The total amount of receptor (bound to ligand and free) is maintained during these computations. For the receptor, this holds for each soma individually ($R_A + C_A = 0.0200$ and $R_B + C_B = 0.100$ within each line), and consequently also for the system as a whole ($R_A+R_B+C_A+C_B = 0.120$ within each line).

### Chemical equilibrium

Chemical equilibrium is maintained for each cell and for the system as a whole: The system is at chemical equilibrium if the relation

$$K_d = \frac{[R][L]}{[C]} \tag{4.12}$$

holds (see section 2.4.4). For each line in table 4.1 it can be verified that the two cells are individually at equilibrium

$$\frac{R_A L}{C_A} = \frac{R_B L}{C_B} = K_d \tag{4.13}$$

and the system as a whole is also at equilibrium

$$\frac{(R_A + R_B)\,L}{C_A + C_B} = K_d \tag{4.14}$$

### Competition

Receptor-ligand binding is a simple mechanism by which competition may be implemented in nature. The total amount of receptor carried by the two cells directly determines the amount of ligand that will bind to each of them. This means that the ratio $C_A/C_B = 0.200$ in each line, which is also equal to the initial value (when $L = 0$) of the ratio $R_A/R_B$. Simulated ligand-receptor binding is therefore suitable for implementing competitive processes, and the receptor concentration of each competing cell component represents directly the competitive strength of that cell component.

As shown, NeuroGene's computation of equilibrium concentrations of ligand, receptor and ligand-receptor complex on two dissimilar cell components is consistent with all the properties of chemical equilibria, and represent a valid approach to implementing competitive mechanisms between cell components.

## 4.5 Simulated diffusion

In order to verify the simulated diffusion computation (see section 2.4.1), a couple of simple systems will be solved symbolically and also computed numerically using NeuroGene. The symbolic solutions to the diffusion equation is given in appendix F.

The solutions differ depending on whether the NeuroGene simulation space is *open* or *closed*. Open means that the edges of the space is permeable to diffusing protein, and the concentration of all proteins outside the simulation space is zero. Closed means that the edges of the simulation space is not permeable. NeuroGene allows for the six faces of the simulation space to be set to open or closed individually. In the following, symbolic solutions will be compared with the numerical results from the NeuroGene simulator.

### 4.5.1 Open simulation space

As shown in appendix F, the solutions to the diffusion equation with open simulation space with diffusion coefficient $D$ have the form

$$\begin{aligned} u(x,y,z,t) &= \sum_{n=1}^{\infty} B_n \sin\left(\frac{n\pi x}{L_x}\right) \sin\left(\frac{n\pi y}{L_y}\right) \sin\left(\frac{n\pi z}{L_z}\right) \\ &\qquad \exp\left(-Dn^2\pi^2\left(\frac{1}{L_x^2}+\frac{1}{L_y^2}+\frac{1}{L_z^2}\right)t\right) \end{aligned}$$

where $L_x$, $L_y$ and $L_z$ are the dimensions of the simulation space. For the purposes of testing the NeuroGene simulation, we may choose a simple function of the form $u(x,y,z,t)$ and subsequently compute the initial concentration function $f(x,y,z) = u(x,y,z,0)$ which will be used to initialize NeuroGene simulations. Initially use the function with $B_1 = 1$ and $B_n = 0$ for all $n \neq 1$.

$$\begin{aligned} u(x,y,z,t) &= \sin\left(\frac{\pi x}{L_x}\right) \sin\left(\frac{\pi y}{L_y}\right) \sin\left(\frac{\pi z}{L_z}\right) \\ &\qquad \exp\left(-D\pi^2\left(\frac{1}{L_x^2}+\frac{1}{L_y^2}+\frac{1}{L_z^2}\right)t\right) \end{aligned}$$

which gives the initial condition

$$u(x,y,z,0) = \sin\left(\frac{\pi x}{L_x}\right) \sin\left(\frac{\pi y}{L_y}\right) \sin\left(\frac{\pi z}{L_z}\right)$$

in 3D. 1D and 2D forms are easily derived in the same way. Figures 4.7–4.9 show the evolution of a system with this initial concentration function in 1D, 2D and 3D respectively. In each figure, the same data is presented using a linear and logarithmic scale for the vertical axis. The three figures are less alike than they appear. The timespan of each is quite different, the diffusion occurring faster in the 3D case than the 1D case (see captions). Figure 4.10 shows the 1D case (i.e. same geometry as figure 4.7) but with a threefold faster diffusion rate. All these functions show relatively good agreement between the simulated diffusion data from NeuroGene (squares) and the symbolic solutions to the diffusion equations (solid

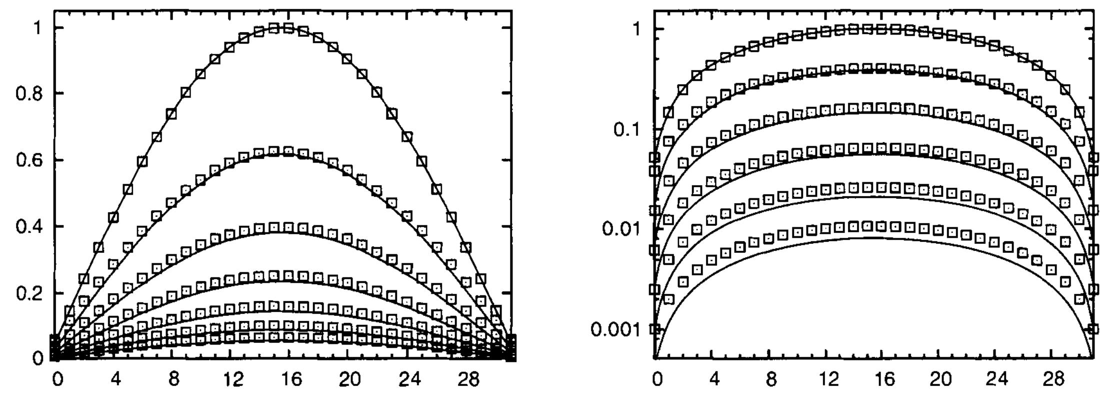


Figure 4.7: Diffusion from a simple initial concentration function in 1D, simulation space dimensions $L_x = 32$, $L_y = L_z = 1$ with $u(x,0) = \sin\left(\frac{\pi x}{L_x}\right)$. The vertical axis shows concentrations (linear scale on the left, logarithmic scale on the right), and the horizontal axis represents the spatial coordinate. There are 1000 simulation time steps between curves on the left, 2000 steps between curves on the right.

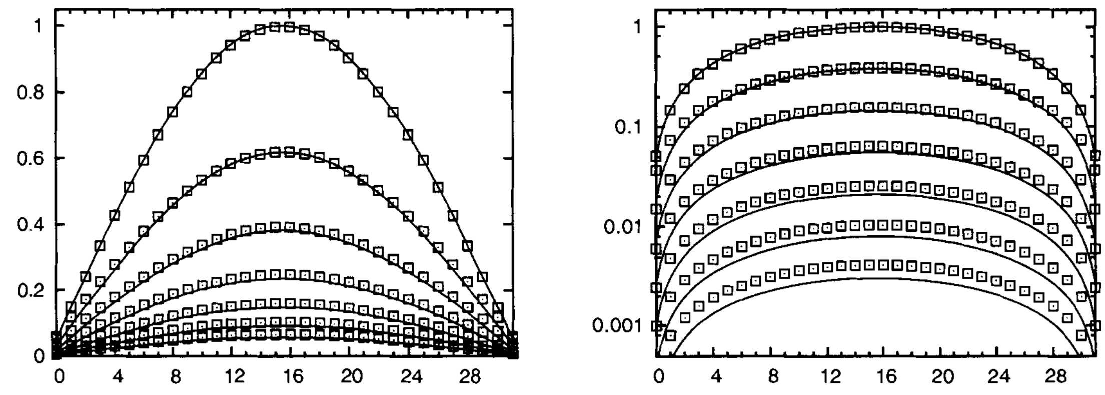


Figure 4.8: Diffusion from a simple initial concentration function in 2D, simulation space dimensions $L_x = L_y = 32$, $L_z = 1$ with $u(x,y,0) = \sin\left(\frac{\pi x}{L_x}\right)\sin\left(\frac{\pi y}{L_y}\right)$ measured at $y = \frac{1}{2}L_y$. There are 500 simulation time steps between curves on the left, 1000 steps between curves on the right.

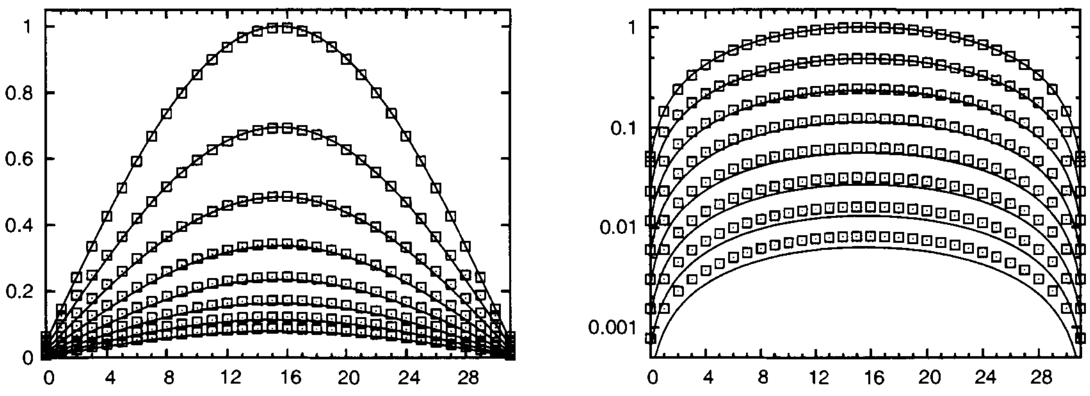


Figure 4.9: Diffusion from a simple initial concentration function in 3D, simulation space dimensions $L_x = L_y = L_z = 32$ with $u(x, y, z, 0) = \sin\left(\frac{\pi x}{L_x}\right)\sin\left(\frac{\pi y}{L_y}\right)\sin\left(\frac{\pi z}{L_z}\right)$ measured at $y = \frac{1}{2}L_y$, $z = \frac{1}{2}L_z$. There are 250 simulation time steps between curves on the left, and every 500 steps between curves on the right.

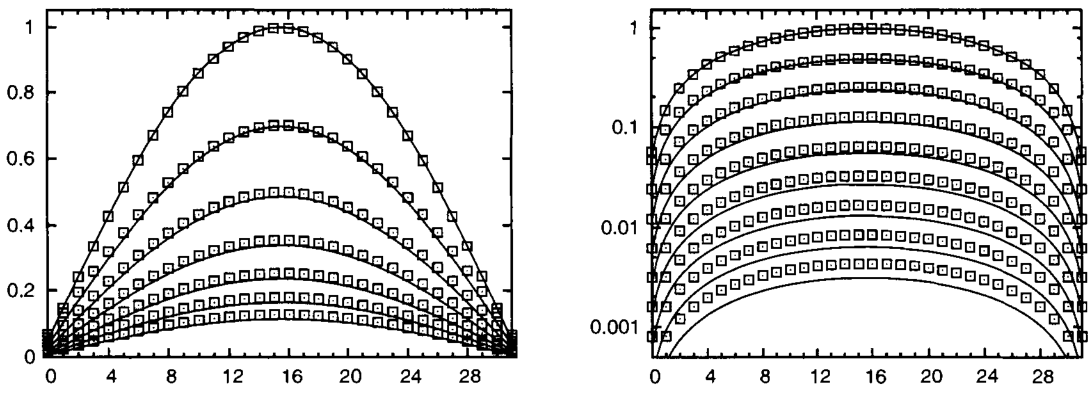


Figure 4.10: Same simulation as that shown in figure 4.7, except that the rate of diffusion is 3 times higher (0.15), with 250 simulation time steps between curves on the left, 500 steps between curves on the right.

lines). The gene and geometry scripts used to set up these experiments are listed in appendix B.1.5.

### 4.5.2 Closed simulation space

Again as shown in appendix F, the concentration of a diffusing protein in a closed simulation space is governed by functions of the general form

$$\begin{aligned} u(x,y,z,t) &= A_0 + \sum_{n=1}^{\infty} A_n \cos\left(\frac{n\pi x}{L_x}\right)\cos\left(\frac{n\pi y}{L_y}\right)\cos\left(\frac{n\pi z}{L_z}\right) \\ &\quad \exp\left(-Dn^2\pi^2\left(\frac{1}{L_x^2}+\frac{1}{L_y^2}+\frac{1}{L_z^2}\right)t\right) \end{aligned}$$

A simple choice of function to use for testing NeuroGene diffusion simulations is the one given by $A_0 = A_1 = 1$ and $A_n = 0$ for all $n > 1$. The function with $A_0 = 1$ is chosed to ensure that concentrations are never negative.

$$u(x,y,z,t) = 1+\cos\left(\frac{\pi x}{L_x}\right)\cos\left(\frac{\pi y}{L_y}\right)\cos\left(\frac{\pi z}{L_z}\right)\exp\left(-D\pi^2\left(\frac{1}{L_x^2}+\frac{1}{L_y^2}+\frac{1}{L_z^2}\right)t\right)$$

which gives the initial condition

$$u(x,y,z,0) = 1+\cos\left(\frac{\pi x}{L_x}\right)\cos\left(\frac{\pi y}{L_y}\right)\cos\left(\frac{\pi z}{L_z}\right)$$

Figure 4.11 shows the evolution of this initial concentration profile under diffusion. Again there is a difference in speed between the 1D, 2D and 3D case, and again there is good agreement between the symbolic solution and the numerical calculation by NeuroGene. The scripts used for these experiments are also listed in appendix B.1.5.

In conclusion, the simulation of diffusion as computed by NeuroGene closely matches the analytical solutions to the diffusion equation. The match is notably better in the closed than in the open case. The reason for this difference is not currently understood. However, in both cases the match is good enough for the simulations reported in this thesis, as well as other simulations that may be envisaged. A basic intuition here is that any developmental program which is so sensitive to the value of a protein concentration as to be disrupted by a deviation as seen in figures 4.7–4.10 is going to be so fragile as to be biologically implausible.

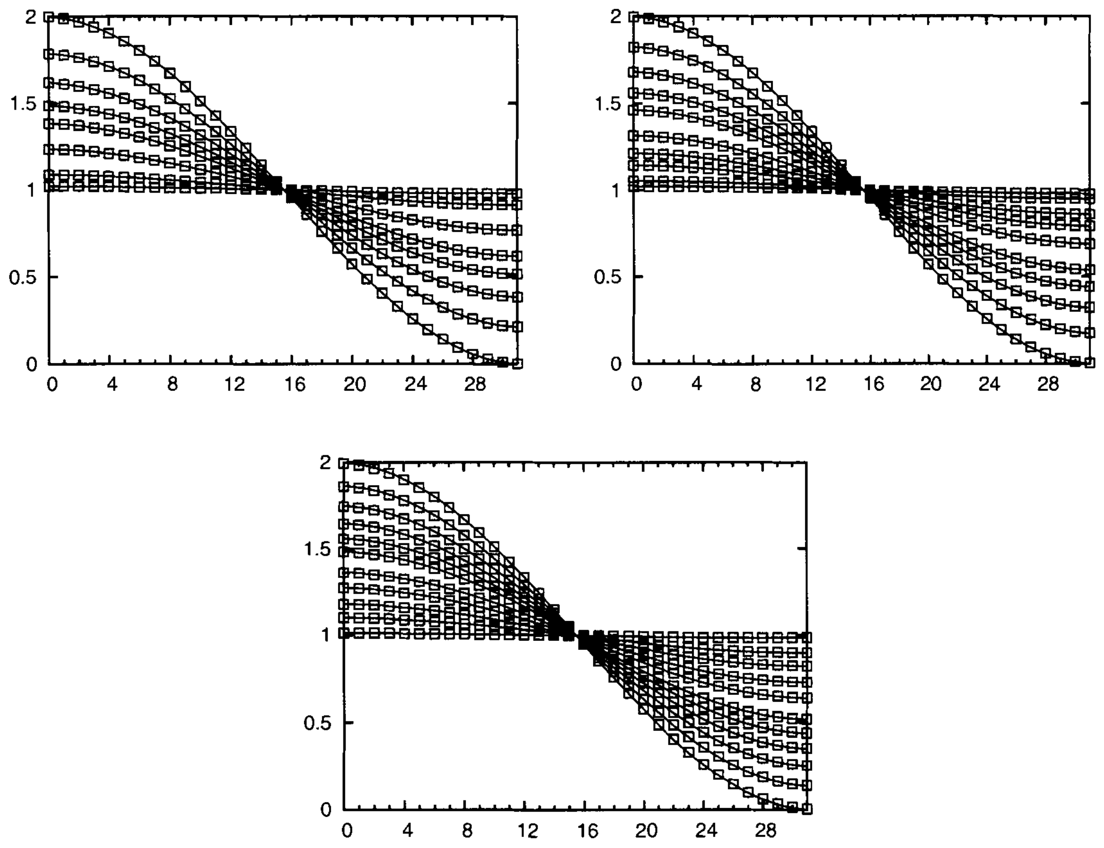


Figure 4.11: Diffusion in a closed simulation space, shown with a linear scale on the vertical axis only. Top left: in 1D ($L_x = 32$, $L_y = L_z = 1$) with $u(x,0) = 1 + \cos\left(\frac{\pi x}{L_x}\right)$, 0, 500,..., 8000 time steps. Top right: in 2D ($L_x = L_y = 32$, $L_z = 1$) with $u(x,y,0) = 1 + \cos\left(\frac{\pi x}{L_x}\right)\cos\left(\frac{\pi y}{L_y}\right)$, at $y = 0$ and at 0, 200, ..., 4000 time steps. Bottom: 3D ($L_x = L_y = L_z = 32$) with $u(x,y,z,0) = 1 + \cos\left(\frac{\pi x}{L_x}\right)\cos\left(\frac{\pi y}{L_y}\right)\cos\left(\frac{\pi z}{L_z}\right)$, at $y = 0$, $z = 0$ and at 0, 100, ..., 3000 time steps.

## 4.6 Verifying other aspects of the implementation

Standard debugging techniques were employed to verify the correctness of the implementation of algorithms, such as the simulated gene and growth cone mechanisms, as well as the neural models of activity. This includes using interactive debugging tools for single-step execution of the algorithms under a range of different circumstances that would cover all possible paths through the code in question. This was possible since these algorithms are relatively simple and limited in scope. In particular, the implementation of the gene and growth cone algorithms can be verified independently of the AST data structures which determine their behaviour.

As noted earlier (section 3.1.2), the classes implementing the AST data structure are numerous, but in most cases very simple. The verification of the language semantics, as implemented by the execute() functions of each AST class, is therefore relatively straightforward. Single-step execution within interactive debugging tools was also used to verify the implementation of the language semantics in the context of the simulation system as a whole.

A parser written using a tool like JavaCC (Sun Microsystems, 2003c) consists of two elements: The grammar specification and the actions associated with each production within the grammar. The implementation of the grammar specification is verified by parsing example input scripts, since an error in the grammar specification causes parser errors on correct input. The parser has been tested against a number of different input scripts (see appendix B).

While the grammar contains a large number of productions, the actions associated with each are in most cases very simple, and often follow a repetitive pattern. Bugs in this part of the parser are therefore relatively easy to discover and correct. During development of the parser and the AST classes, the following technique was used to verify the correctness of both: The AST classes were equipped with recursive-decent `toString()` functions. When `toString()` is called on the root node of some AST data structure, a text representation of the AST data structure *as a whole* is returned. Furthermore, the text representation was designed to be identical (disregarding insignificant white-space) to the input script which was used to create the AST data structure in the first place. Any deviations between the two, indicative of an error either in the parser or in the AST data structure, is easily spotted by eye, using automatic text-comparison tools, or even by resubmitting the produced text to the parser.

## 4.7 Conclusion

The goal of this chapter has been to demonstrate the accuracy of the simulation of physical processes by NeuroGene. Given the constraints imposed by e.g. the discretization of the simulation space, the computations underlying the NeuroGene simulations perform satisfactorily with respect to the requirements of the developmental systems under investigation.

# Chapter 5

# Results II: Developmental simulations

NeuroGene has been designed to be able to simulate a wide range of developmental phenomena. The simulations presented in this chapter involve three developmental mechanisms that all play central roles in the development of the nervous system. These are patterning, axon guidance and activity-driven development. In each case, we have chosen to simulate systems which have been the subject of scientific study for many years, and which are relatively well understood. Using NeuroGene as a tool, we have been able to develop a novel model for the axon guidance mechanism underlying the formation of topographic maps. The new model explains a wider range of observed biological phenomena than other previously proposed models, and brings new insights into the mechanism of axon guidance.

Patterning is the process by which an initially uniform mass of cells become subdivided into contiguous domains of cells which differ in properties, which may ultimately develop into organs with different roles in the life of the organism. While patterning plays an important role in the development of the vertebrate central nervous system (see e.g. Scholpp et al., 2003, Bishop et al., 2002 and Fukuchi-Shimogori and Grove, 2001), we have chosen to simulate one of the the most studied and best understood examples of patterning, the initial establishment of segments in the embryo of the fruit-fly.

Axon guidance is the process by which axons extend and grow toward a specific target area, forming specific connections to particular target cells. Axon guidance has been studied in many contexts in insects and vertebrates — however, the best understood system is the topographic projection from the eye to the mid-brain in vertebrates. This connection transmits the visual image from the retina to the brain. The mid-brain target area, known as the tectum or superior colliculus (depending on the species) receives the neural activity from the eye in such a way that adjacent sites in the target area receive input from adjacent sites in the retina. Effectively, the image projected on the retina is replicated in the form of neural activity in the tectum. The topographic projection is an important structural motif in the brain — they are found throughout the somatosensory systems which process information from all our senses, and many speculate that they may also play a role in higher cognitive function.

In activity dependent development and learning, patterns of neural activity cause changes in the synaptic weights of neural connections. Again the best studied examples are from the vertebrate visual system. We simulate the concurrent processes of visual field refinement and the formation of ocular dominance columns. The former is the process by which initial, diffuse and imprecise connections of a topographic projection, formed through axon guidance, are sharpened through synaptic modifications. In the latter process, target cells which initially receive equal activation from each eye become dominated by one or the other eye.

In this chapter, I present simulations involving these three important developmental mechanisms. In the first (section 5.1), phenomena of tissue patterning through diffusible chemical signals are simulated. Here we use a well understood system from fruit fly development, but similar processes may be involved in the development of the vertebrate brain. such as in the formation of the segmented vertebrate hind brain (Barrow et al., 2000).

The second set of experiments (section 5.2) simulate the initial formation of topographic connections from the retina to the main target area in the mid brain. This process depends on axon guidance by chemical signals, and is not activity dependent. We present a novel model for the formation of such maps which was developed using NeuroGene. There is a wealth of experimental data on the changes in this connection resulting from changes in the geometry or biochemistry of the retina or the mid brain target. The test of any model is the degree to which it can explain this results. Our model, which relies on the explicit modeling of growth cones, is able to explain a very wide range of experimental observations.

Finally, simulations of the activity-dependent refinement process of the same topographic connection is presented in section 5.3. These simulations use a learning mechanism which is based on chemical signaling across the synaptic gap, where all molecular interactions are explicitly modeled. The last two simulations both involve the development of the same axonal projection from the retina to the tectum. Not included here is the simulation of the formation of the initial synaptic connections between the RGC axons and the tectum. NeuroGene does not at present have the capability to model this complex arborization process. The addition of this functionality to NeuroGene is possible, and this would make NeuroGene much more of a complete tool for the study of neurodevelopment. While this and other mechanisms are also involved in the formation of the visual system and the nervous system as a whole, these three simulations presented here clearly demonstrate the wide range of processes which NeuroGene can currently simulate.

## 5.1 Patterning: Drosophila segmentation

The fruit-fly (*Drosophila melongaster*) is widely used in research aimed to better understand developmental processes. One of the first events in the development of the fruit-fly is the emergence of patterns of gene expression which determine the future locations of the segments of the fruit-fly body. Starting from a uniform population of cells, a pattern of cell differentiation is established, where initially identical cells start to show differences in the genes that are expressed within them. The cells subsequently go through different developmental programs which are determined by these early gene regulation events. Through non-linear gene regulation mechanisms, slight differences in the concentrations of regulatory proteins early in development lead to cells going through completely different developmental programs, and ultimately fulfilling different roles in the fully formed organism. Variations on this basic mechanism causes the formation of organs and other structures in all complex animals.

An important gene in this context is *Even-skipped*, so named because mutants which lack this gene are missing every other segment (the even-numbered ones) in the body. In the early stages of development of the fruit-fly, cell division and migration establish a situation where a single layer of cells line the inside surface of the egg. Gene expression in these cells is initially controlled by a number of diffusible regulatory proteins. These factors establish a pattern of bands perpendicular to the long axis of the egg. This pattern delineates the cells which will eventually become the segments of the fully developed fruit-fly larva. The first band to appear is band number two counted from the head. We here simulate the establishment of segment number two of the fruit-fly embryo through the regulation of *Even-skipped* and related genes.

Figure 5.1 shows the gene network underlying the simulation (Muller, 1997; Held, 2002). Two broad concentration gradients (*Bicoid* and *Nanos*) together give rise to a sharply defined expression pattern of *Even-skipped.* This happens because of numerous enhancing and inhibiting interactions between a total of eight different compounds (seven proteins and one mRNA). This gene network includes many examples of gene-gene interactions which go beyond simple boolean logic, but which are straightforwardly represented in the NeuroGene language. See for example the interaction from *Hunchback* to *Kruppel*, which is enhancing at low levels of *Hunchback*, and inhibiting at high levels, as well as the *Nanos* inhibition of the translation of *Hunchback* mRNA to protein. The latter requires that the transcription and translation of *Hunchback* be modeled separately. *Hunchback* is therefore represented by two simulated genes, one which expresses the *Hunchback* mRNA (representing transcription), and another which expresses the *Hunchback* protein (translation). The latter simulated gene requires the *Hunchback* mRNA in order to be expressed, thereby simulating the role of the mRNA in translation Brook (1998).

### 5.1.1 Results

Figure 5.2 shows the NeuroGene simulation of this system. Soma instances represent the cells of the fly embryo, as shown on the top left. The arrangement of the different genes are the same as in figure 5.1. On the left is shown the concentration distributions of proteins and *Hunchback* mRNA at the outset of the simulation. All protein concentrations are initially zero (blue) with the exception of *Bicoid* and *Nanos*, which are initialized to have broad concentration gradients extending across the extent of the organism, and *HunchBack* mRNA which is present at a constant level throughout the embryo.

The situation after the simulation has reached a steady-state is shown on the right. All genes except *Bicoid* and *Nanos* are expressed by the somas, and therefore will only have high concentrations close to where the somas are. The concentration profiles of all the proteins and the *Hunchback* mRNA shown in figure 5.2 match the experimental data summarized in figure 5.1. The gene and geometry scripts for these simulations are included in appendix B.

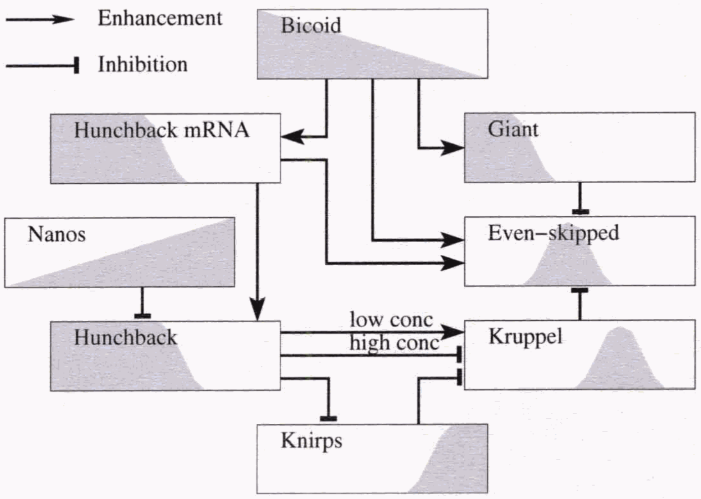


Figure 5.1: Gene network responsible for establishment of the first segment of *Drosophila melongaster*. Boxes show the concentration distribution of each protein or mRNA, with the fly's head (anterior) to the left and tail (posterior) to the right.

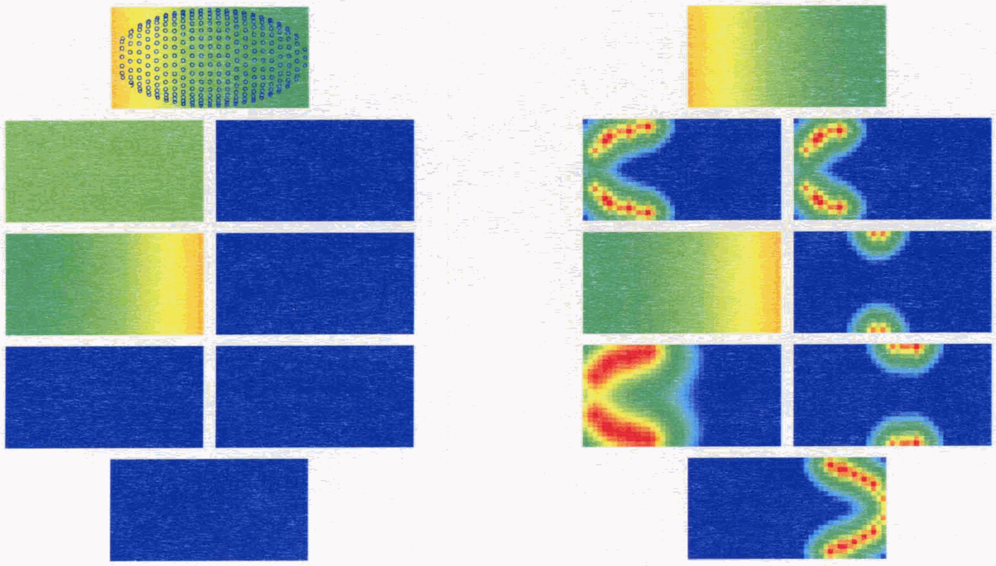


Figure 5.2: Start point (left) and steady state situation (right) of the NeuroGene simulation of the Evenstripe system of *Drosophila melongaster*. The arrangement of the different genes are the same as in figure 5.1. Blue small circles on the top left show the location of the cells which are hidden in the other images. Colour scale: Blue represents concentrations $< 1 \times 10^{-10}$ and red concentrations $> 2 \times 10^{-4}$ moles/L.

### 5.1.2 Discussion

Patterning processes often rely on the establishment of protein concentration gradients through diffusion. Such gradients may then directly or indirectly establish well defined patterns of differential gene expression. This in turn may give rise to additional concentration gradients of other proteins, thereby establishing increasingly complex patterns. The simulation of *Even-skipped* is an example of such increasing levels of pattern detail, as the broad gradients of *Nanos* and *Bicoid* regulate *Hunchback*, *Knirps* and *Giant*, which in turn define the highly restricted expression domains of *Kruppel* and *Even-skipped*. As figure 5.2 shows, NeuroGene is capable of encoding the genetic regulatory network controlling *Even-skipped* expression. Based on the gene network and on the simulation of physical processes including diffusion, NeuroGene simulates the formation of expression domains similar to those which are observed in nature.

## 5.2 Axon guidance: Topographic map formation

The ability of the vertebrate central nervous system to act as a sophisticated information processing device depends on precise neural interconnections between its various parts. These connections are formed through axon guidance, by which an extending axon, tipped by a dynamic growth cone, navigates to its proper target site where it forms synapses with appropriate target neurons. Topographic maps is a type of neural structure which is formed through sophisticated axon guidance. While a set of parallel connections from one tissue to another might seem like a simple structure to build, the resilience of this mechanism to experimental tampering (Goodhill and Richards, 1999) points to a complex developmental process.

As mentioned in section 2.3, topographic projections are observed in many parts of the brain involved in early processing of sensory information, including visual, auditory, olfactory and somatosensory input (Kandel et al., 2000) with some systems containing several such projections. The same mechanism and even the same proteins may be involved in the formation of these topographic projections, at least within the visual pathway (Marín et al., 2001).

While no definite proof exists, it is possible that topographic projections also play a role in cognitive functions. For example, Rodriguez et al. (2004) report a computational model of the thalamus and sensory cortex capable of simple storage/recall and hierarchical classification. Topographical connections are central in the model's hierarchal classification task, while the non-topographical connections implement the storage/recall mechanism. In an experimental study, the role of topographic projections in learning in rodents and humans was investigated (Diamond et al., 2003), looking at the sense of touch. In the human trials, subjects learned to distinguish two different frequencies of vibration, using one particular finger tip. The researchers found that when testing the subjects using another finger tip, the learning did not transfer, indicating that the site of learning was associated with the trained fingertip only.

A more detailed study, reported in the same paper, was carried out on the sense of touch conveyed by the whiskers of rats. All but one whisker was removed on each rat during the training session. In the test phase, the trained whisker was removed, and a "prosthetic" whisker was attached with glue to the base of another, previously removed and therefore untrained whisker. The researchers found that the rats' ability to repeat the learned behaviour decreased with increasing distance between the trained and tested whisker, indicating that the learned ability is transferred to nearby but not to distant whiskers. This is strong evidence that the topographic relationship between the whiskers is maintained at the site where the learning of the task takes place.

Topographical projections are also interesting from a theoretical viewpoint. Artificial neural networks with a topographic topology may be trained, using common connectionist learning rules, to represent *universally qualified one-to-one mappings* (UQOTOMs, Marcus, 2001, chapter 3). Such mappings are by definition able to generalize outside the training space, something which is otherwise an elusive goal in connectionism.

The topographic projection from the retina to the mid-brain has been extensively studied in many species under a wide range of experimentally manipulated conditions. As predicted by Roger Sperry in 1963, the development of the well formed projection depends on proteins expressed in concentration gradients both in the RGCs (including their axons and growth cones), and on proteins expressed in the tectum. The tectum expresses a class of proteins known as ephrins. Ephrins are divided into two classes A and B, each of which has a number of members (ephrinA1 – ephrinA5 and ephrinB1 – ephrinB3) (Eph nomenclature committee, 1997). The RGCs in turn express a class of proteins known as Eph-receptors. Eph receptors

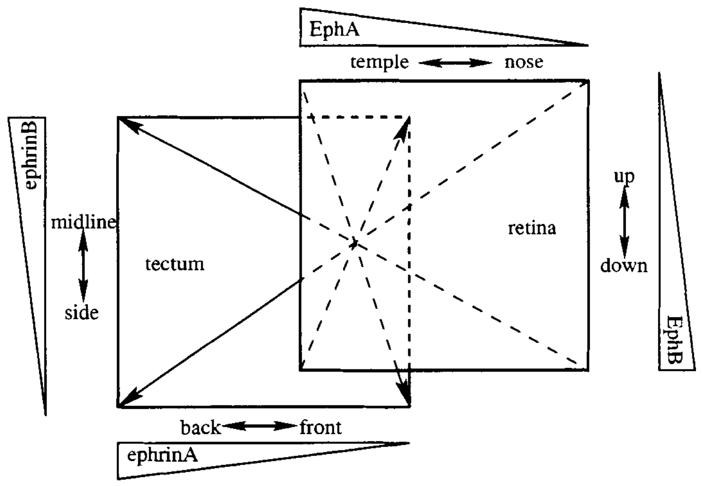


Figure 5.3: Expression pattern of ephrinA/B in the tectum and EphA/B in the retina. Horizontal and vertical ramps show the directions of concentration gradients of ephrins in the tectum and of Eph receptors in the retina. The projection from the eye to the tectum is crossed both vertically and horizontally, as indicated by diagonal lines.

also fall into two classes A and B (with members EphA1 - EphA8 and EphB1 - EphB6 respectively, *ibid*) . EphA receptors generally bind to and detect ephrinA proteins only, and EphB receptors detect ephrinB only. However, within each class A and B the receptors are generally "promiscuous", i.e. most EphA receptor types detect many of the ephrinA proteins with varying sensitivities, and the same goes for EphB and ephrinB (*ibid*). For the sake of simplicity, I will in the following only refer to these proteins by classes, and not go into the distinguishing properties of the individual proteins and receptors.

All these proteins are expressed in gradients from low concentration at one edge of the topographic map to high concentration at the other. The ordering of the topographic projection along the horizontal dimension depends on the interaction between tectal ephrinA and retinal EphA, while ordering along the vertical dimension depends on interaction between ephrinB and EphB, see figure 5.3.

Note that in the horizontal dimension, high EphA axons map to low ephrinA tectum, while in the vertical, high EphB maps to high ephrinB tectum. From this arrangement of gradients it is evident that the ephrinA/EphA interaction is repellent, leading the high EphA axons (which are most sensitive to ephrinA) to regions with low ephrinA levels. The ephrinB/EphB interaction is attractive, since the high EphB axons, which are most sensitive to ephrinB, wind up in the part of the tectum where the ephrinB levels are high.

A number of models have been proposed to account for the map formation process, many of which have been submitted to computer-based simulations (Hope et al., 1976; Whitelaw and Cowan, 1981; Gierer, 1983; Overton and Arbib, 1982; Fraser and Perkel, 1990; Weber et al., 1997; Yates et al., 2004). None have yet been able to account for all the experimental observations, particularly relating to neural structures resulting from experimental modifications of the eye and/or mid-brain during early stages of development before the map normally forms, see appendix G. Here we report a new model which accounts for many, but still not all observed phenomena.

### 5.2.1 Implementation

The NeuroGene simulation of topographic maps rely on the model of growth cones described earlier (see section 2.6). The gene script is given in appendix B.2.2. In these simulations, the growth cones are involved in guiding the RGC axons to the tectum, as well as in the sorting mechanisms giving rise to a fully formed topographic projection.

The genes which together implement the NeuroGene simulation of topographic map formation fall into three classes: First are simulated genes which are analogous to genes which are known to exist in nature. All the genes in the fruit fly segmentation experiment above belong to this category. In the following, ephrin and Eph receptor genes fall into this category. Secondly, some simulated genes are not known to have directly analogous genes in nature, but their function is typical for a wide range of genes known in nature. The simulated cell identity genes are such genes. Thirdly, there are simulated genes for which the existence of analogous genes in nature can be inferred from circumstantial evidence. This includes the simulated genes defining cardinal axes: We know that Eph receptors and ephrins are expressed in concentration gradients, and we can infer the existence of some genetic mechanism by which this is accomplished. In some cases, the genetic mechanism of the regulation of these genes are known (e.g. EphA3, see Schulte and Cepko, 2000), however such details are not relevant to the simulation at hand, and a simple mechanism of regulation of EphA and EphB are used in stead. Similarly we can infer the existence of genes which cause the formation and which control the activity of growth cones.

```
1 growthconeFunction findTectum {
2    filopodiumCount  = 4;
3    filopodiumLength = 0.2;
4    searchRange      = 0.0;
5    neighbourCount   = 0;
6    migrate(externalConcentrationOf(retinoTectalGradient));
7 }
```

Figure 5.4: Growth cone function for climbing a concentration gradient of the protein *retinoTectalGradient*. Lines 2–5 define the shape of the growth cone, while line 6 defines its simple behaviour. The growth cone shape used here is not based on experimental observations. However, the simulation is not sensitive to the growth cone shape in this case.

The growth cone program for finding the tectum uses a concentration gradient established by a diffusible protein produced by the tectum. The gene *retinoTectalGradient* expresses this protein, which is released by tectal cells and diffuse through the extracellular space. A corresponding gene is not known to exist in nature — the simulated gene represents the mechanism by which retinal axons navigate to their target tissue, which in nature may involve multiple genes. Through diffusion this protein establishes the concentration gradient which RGC axons use to navigate toward the tectum. The growth cone program which causes an axon to navigate this gradient toward its source is simple (see figure 5.4): The bid value for each filopodium is simply the perceived external concentration of the signal forming the gradient. This means that whichever filopodium perceives the higher concentration, i.e. the filopodium which is closer to the target area, will win the competitive auction of the growth cone algorithm.

When the RGC axons reach the tectum, they detect a tectum-specific protein, *tectalCellMarker*. This causes the RGC growth cones to switch from a "find the tectum"-mode to a "sort-map" mode upon axon arrival at the tectum. The various simulated genes which are used to implement this system within NeuroGene are listed below.

### Cell identity genes

As outlined in section 1.5.2, cell of a particular cell type are identified by the genes which are expressed within them, and the functional and structural consequences of such gene expression. Genes which uniquely identify a cell type are cell identity genes. The simulated

```
1 gene retinalCellMarker {
2    diffusible = false;
3    regulation {
4       if (surfaceConcentrationOf(RetinalCellMarker) > 0.1) {
5          expressExternally(0.25);
6       }
7    }
8 }
```

Figure 5.5: Implementation of a self-enhancing cell identity gene. Once this gene is activated, leading the surface concentration to exceed 0.1, the test on line 4 will be satisfied, and the gene will continue to be expressed forever.

cell identity genes are *auto-enhancing*, meaning that the presence of the protein produced by each such gene causes the gene to be expressed. This gives such genes the property that once they are expressed, they continue to be expressed. We have implemented such genes for tectal cells (*tectalCellMarker*) and retinal cells (*retinalCellMarker*, see figure 5.5). When the simulation is set up, tectal cells are initialized (see section 3.1.10) with a non-zero concentration of the protein *tectalCellMarker*, ensuring that the gene is expressed in these cells. All other cells are initialized with zero concentration of *tectalCellMarker*. The proteins are membrane bound and expressed on the surface of cells, allowing other cell components to detect the protein and thereby identify the cell type. For example, the retinal growth cones "know" that they have reached the tectum when they detect the protein *tectalCellMarker* in their environment.

### Genes defining cardinal axes

These simulated genes are expressed in linear gradients, and are used to establish ephrin and Eph receptor gradients. While corresponding genes are not known to exist in nature, clearly there must be some genetic mechanisms which establish the observed concentration gradients of ephrins and Eph receptors as well as those of many other proteins. These simulated genes are hypothetical, however, they represent the mechanism which causes the formation of concentration gradients which are known to exist in nature. There are four such genes, used to establish the two cardinal axes in the retina and in the tectum.

### Ephrin genes

The simulated ephrin genes represent genes which have been identified and studied in nature. These genes are expressed in gradients across the tectum where they act as ligands to the Eph receptors carried by incoming RGC axons. While precise measurements of absolute protein concentrations are difficult to make in tissues, the concentration gradient of ephrinA2 in the retina of goldfish has been observed to have an approximately exponential shape, with about 2-fold difference in concentration between the two extremes of the gradient[1]. In our experiments, ephrinA is expressed in an exponential gradient decreasing from the back to the front edge of the tectum (see figure 5.3), the exact shape of the gradient given by $A\exp(-Bx)$ with $A = 0.5$, $B = \ln(5)$ and $0 \leq x < 1$. The expression rate of ephrinA is thus 0.5 at the back edge of the tectum (where $x = 0$) and 0.1 at the front edge (where $x = 1$). As will be shown, the system is tolerant to variations in $A$ and $B$.

While the sorting of RGCs axons in the midline–side dimension has many similarities to the mechanisms driving the sorting in the front–back dimension, it has been studied less, and we have less experimental data to draw on in designing the simulation. For this reason, we use a slightly modified form of the back–front mechanism to implement the sorting in the other direction, and modify the growth cone definition to integrate the sorting cues in both dimensions. This gives rise to the simulated genes ephrinB2 and ephrinB5. The only modification we make is that we change the ephrinB – EphB interaction to be attractive unlike the repellent ephrinA – EphA interaction, compare lines 189–193 and 198–202 in the gene script in appendix B.2.2. The simulated ephrinB gene is expressed in an exponential gradient increasing from the side to the midline edge of the tectum (see figure 5.3).

### Eph receptor genes

There are two Eph receptors carried by RGC axons, one for the midline–side (or nasal–temporal) dimension (EphA3) and one for the top–down dimension (EphB3), see figure 5.3. These are both expressed in exponential gradients on the surface of the RGCs, including their axons. The function used is $A\exp(-Bx) + C$, with $A = 0.176$, $B = 5$ and $C = 0.074$,

[1]Dr. Sara Dunlop, University of Western Australia, personal communication during a presentation hosted by ICORD, University of British Columbia, April 2 – 2002, with the title "Optic nerve regeneration: maps, molecules and making connections".

where $x$ is the internal concentration of the proteins which define the horizontal coordinates of the retina. This gives expression rates of EphA of 0.25 at the temporal end of the retina to 0.075 at the nasal end. The map formation model is tolerant to changes in these parameters, as will be shown below. A corresponding gradient of EphB is established, increasing from the upper to the lower edge of the retina (see figure 5.3).

### 5.2.2 The topographic mechanism

We introduce a novel model for the formation of topographic maps. This model is based on the concept of a *pecking order* between RGC axons. The behaviour of each RGC axon is determined by the presence or absence of other RGC axons with relatively higher pecking order in the vicinity. The pecking order is defined by the axonal concentration of EphA and EphB — the higher the concentration, the higher the pecking order. The model can be summarized as follows:

> For a given axon $A$, if there are no axons in the vicinity of $A$ which have higher pecking order than $A$, then $A$ moves toward the more attractive end of the target gradient, otherwise it moves toward the less attractive end.

In the horizontal dimension, the ephrinA gradient in the target area is repellent, so that the more attractive end of the gradient is the anterior, low-ephrinA end. In the vertical dimension, ephrinB is attractive, meaning that the attractive end of the gradient is the high-ephrinB at the mid-line. The roles of the ephrinA/B and EphA/B gradients are therefore well defined: the ephrinA/B gradients give directional information to the invading axons, as has already been suggested (Hope et al., 1976). The EphA/B gradients determine the behaviour of the axons in such a way that low EphA/B axons cede their place to relatively higher EphA/B axons, and high EphA/B axons ignore low EphA/B axons.

This algorithm relies on RGC axons being able to switch from being attracted to being repelled by the same ephrin gradients, depending on the Eph receptor levels of surrounding RGC axons. This is similar to the cAMP and $Ca^{2+}$-dependent switching in growth cone behaviour observed in several systems, see section 1.3.2. In support of this model of map formation, switching between attractive and repellent growth cone behaviour toward EphA/B signaling has also been observed (McLaughlin et al., 2003a, p. 63).

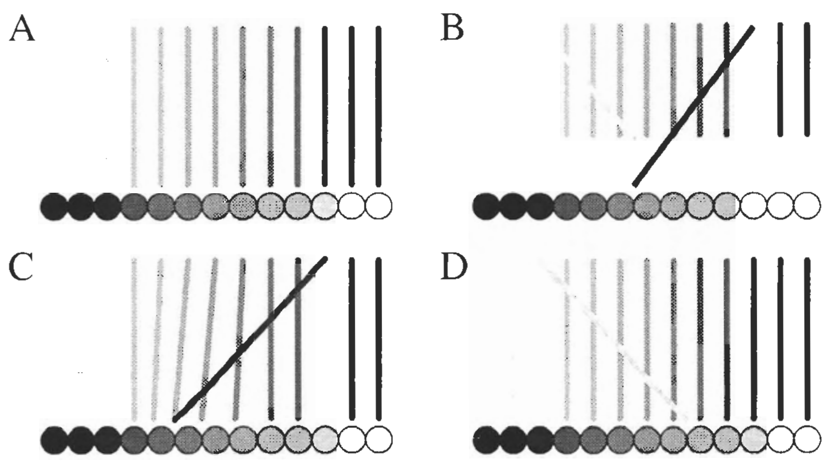


Figure 5.6: This figure shows how the "pecking-order" mechanisms gives rise to a sorted topographical projection. A: a properly sorted map. B: two dissimilar axons meet in an immature map. The high-EphA axon (dark) is not affected by the low-EphA axon, and moves toward the low-ephrinA end of the map. The low-EphA axon is induced by the high-EphA axon to shift toward the high-ephrinA end of the map. C: A high-EphA axon with an inappropriate location causes nearby low-EphA axons to shift to the left, giving room for the high-EphA axon to move to the right. D: A low-EphA axon in an inappropriate location surrounded by high-EphA axons shifts toward the higher ephrinA. The surrounding high-EphA axons are not affected by the wayward low-EphA axon.

Figure 5.6 shows how this algorithm leads to a sorted topographic projection. Figure 5.6A shows a fully sorted map, where high-EphA axons (dark lines) project to low-ephrinA tectum (light circles), and low-EphA axons (light lines) project to high-ephrinA tectum (dark circles). In 5.6B, two RGC axons with different EphA levels meet, initially in an inverted order with respect to proper sorting. The high-EphA axon is not affected by the low-EphA axon, and moves toward the low-ephrinA part of the tectum, i.e., toward the right. The low-EphA axon detects the presence of the high-EphA axon. Consequently, the low-EphA axon moves toward the higher-ephrinA part of the target area, i.e., toward the left. This leads to the proper ordering of the two axons being restored. Figure 5.6C-D shows the situation in which a single high-EphA axon (5.6C) or low-EphA axon (5.6D) is misdirected in an otherwise properly sorted projection. In 5.6C, the aberrant axon is higher pecking-order than all the surrounding axons, meaning that it will migrate toward lower ephrinA concentrations. Its presence causes properly sorted axons to migrate up the ephrinA gradient, but this effect is transient, and the proper sorting will be reestablished once the high-EphA axon has left. In 5.6D a low-EphA axon is in an aberrant location. The presence of high-EphA axons in its vicinity will cause this axon to migrate toward the higher-ephrinA part of the target area.

The results of our simulations show that this model can create topographic order in both horizontal and vertical dimensions simultaneously. We also show that this model replicates a wide range of results from *in-vivo* experiments, beyond what any other published model is able to do.

NeuroGene has proved an invaluable tool in the development and verification of this model. We use the NeuroGene simulated growth cones to implement the model, which is encoded in the form of a growth cone function. The bid values computed by this function are such that the winning filopodium causes the axon to migrate according to the model as stated above. This encoding requires the explicit representation of the growth cone's interaction with its environment in terms of the protein concentrations it detects and the internal signal processing it performs. This makes it easy to relate the model to what is known about RGC growth cones, their receptors and their behaviour.

### 5.2.3 Results

The retinotectal system has been subject to a wide range of experiments in which the retina or the tectum were modified through surgery at an early stage in development. More recently, genetic manipulation has also been used. The resulting maps are then studied to determine whether and to what extent the normal process of map formation has been disrupted by the experimental modifications. See Goodhill and Richards (1999) for a review of early experiments, and Brown et al. (2000) and Hornberger et al. (1999) for important studies using genetics to experimentally modify the projection.

In the following, we report the results of a series of NeuroGene simulations which seek to replicate the findings from *in vivo* experiments in which either surgical modifications or genetic manipulations were used to alter the context in which topographical projections form. In each experiment we use the same simple geometry to represent the retina and the tectum, two parallel 2D sheets of cells, separated by a certain distance. The concentration gradients within each cell sheet are arranged so that axons forming a normal projection will cross in the space between the two layers. This was done so that the initial projection formed when the RGC axons first reach the tectum is highly dissimilar from the final projection. The precision of the resulting projection can also be estimated in this geometric arrangement, since an ideal projection is one in which all axons cross through a single point half-way between the two cellular sheets.

### Wild type

The term "wild-type" is common biological jargon referring to the naturally occurring processes as found "in the wild", and not subjected to experimental modifications. It represents the standard situation to which experimentally modified systems are compared. The results of the wild-type experiment is shown in figure 5.7.

In the initial situation (figure 5.7A) axons extend approximately parallel from the retina (top) to the tectal target area (bottom). As the simulation progresses (figure 5.7A–I) axons shift across the tectal area, until most axons cross to the opposite side of the tectum, with axons high in EphA (shown in red) project to tectal areas low in ephrinA (shown in blue) and low EphA axons (blue) project to high ephrinA tectal areas (red). The projection is near ideal, with nearly all axons crossing each other in a small area between the two cellular sheets. A handful of axons remain in inappropriate locations. There is evidence that in nature, activity dependent processes cause neurons with such errant axons to die, thereby improving the quality of the topographic projection.

Figure 5.7J–K shows the topographic ordering in the vertical direction. In 5.7K, high EphB axons are shown in yellow and low EphB axons in blue, while high ephrinB tectum is shown in red and low ephrinB tectum in blue. The topographic model is able to sort the projection along both dimensions simultaneously with high precision.

### Changes in gradient shapes

Figure 5.8 shows a set of eight experiments carried out to verify that the topographic map forms regardless of the particular slope and overall magnitude of the retinal EphA and tectal ephrinA gradients. These experiments are all slight modifications of the "wild-type" experiment — the only changes being in the expression rate of EphA or ephrinA as function of position along the retinal or tectal axis. If the EphA or ephrinA gradient in the wild

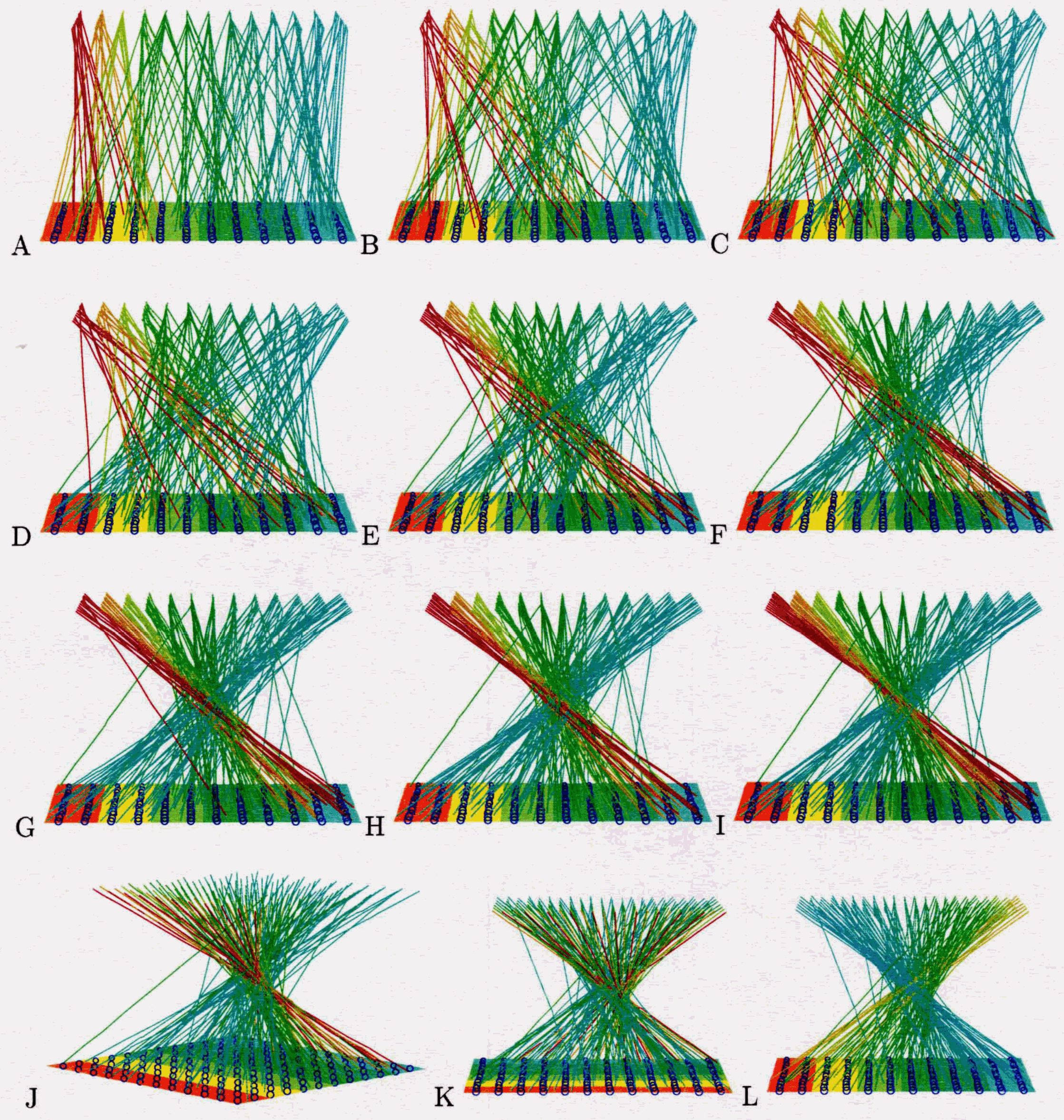


Figure 5.7: Simulation of map formation in wild-type. A-I are images of a wild type experiment shown at 1,000, 4,000, 7,000, 10,000, 13,000, 16,000, 19,000, 22,000, and 25,000 time steps. J and K the same situation as I, but seen from different angles. L is the same situation and view angle as K, but here the ephrinB and EphB proteins are visualized in the tectum and the RGCs, respectively.

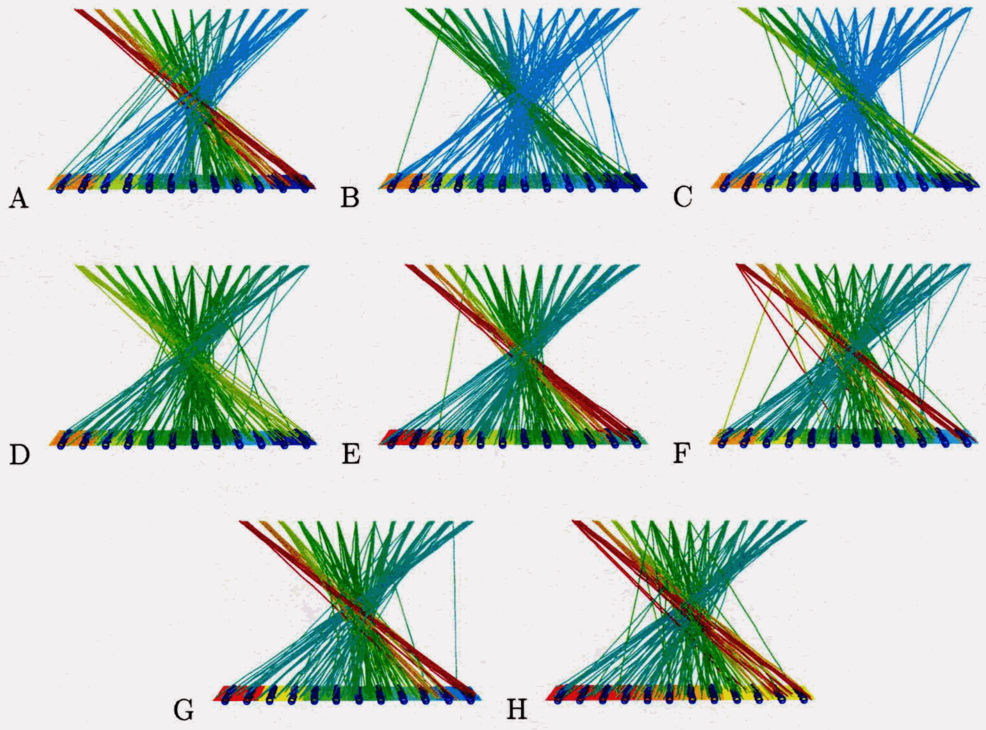


Figure 5.8: Experiments showing the resistance of the map formation model to variations in gradient shapes. A: high magnitude of retinal EphA; B: low magnitude of retinal EphA; C: high slope of retinal EphA; D: low slope of retinal EphA; E: high magnitude of the tectal ephrinA; F: low magnitude of the tectal ephrinA; G: high slope of tectal ephrinA; H: low slope of tectal ephrinA. See text for details of these gradient shapes.

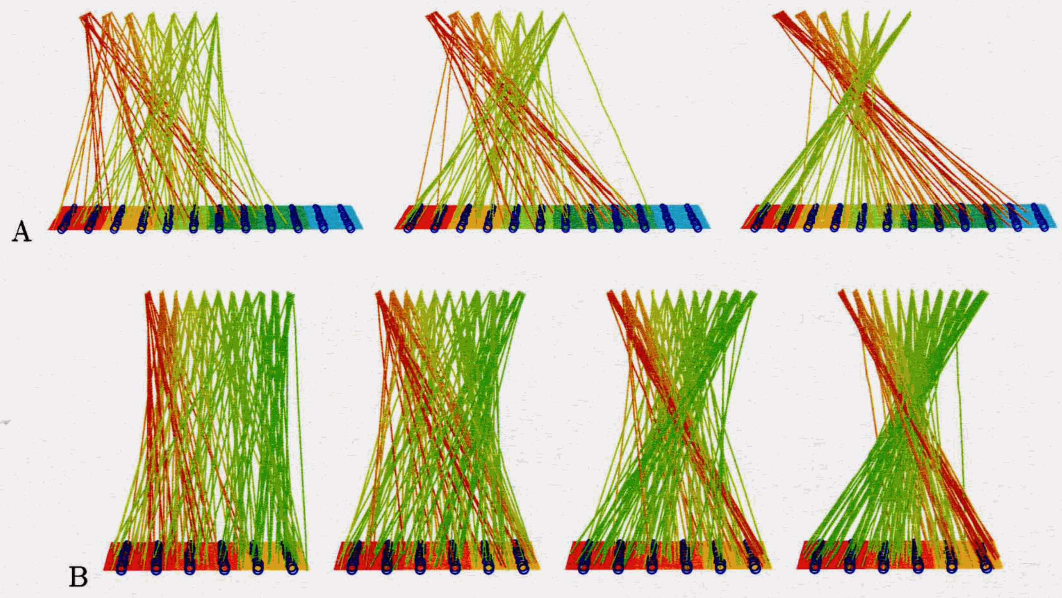


Figure 5.9: A: Expansion of the topographic projection caused by removal of half the retina: Half the normal number of axons form a map covering the entire tectum. The axons also represents half the normal gradient in EphA concentrations. B: Compression of the topographic projection caused by removal of half the tectum: The normal number of axons, representing the normal range of EphA concentration gradient, form a map covering the reduced tectum. The tectum represents half the normal gradient in ephrinA concentrations.

type experiment is described by a function $A\exp(-Bx)$ with $A > 0$, $B > 0$ and $x$ varying from zero to one across the retina or tectum, the modified gradients where given by the following functions: High magnitude: $2A\exp(-Bx)$; low magnitude: $\frac{1}{2}A\exp(-Bx)$; high slope: $A\exp(-2Bx)$; low slope: $A\exp(-\frac{1}{2}Bx)$. Note that since $B > 0$, all these gradients will have maxima at $x = 0$, all with maximal concentration equal to $A$ except the "low magnitude", which has the maximal concentration of $\frac{1}{2}A$. These experiments show that the map formation does not depend on the particular shape of the concentration gradients either in the retina or the tectum.

## Mismatch

The kind of arbitrary changes in the shape of concentration gradients, as outlined above, are not generally possible to make *in vivo*. However, in one experiment, a similar result was obtained though surgical means: The temporal half of the retina was removed from a goldfish, as well as the posterior half of the tectum. The remaining nasal RGCs projected in topographic manner to the anterior tectum, which is normally innervated by temporal axons (Goodhill and Richards, 1999; Horder, 1971). In this experiment, low-EphA nasal axons were able to form a topographic projection in an area of the tectum where the ephrinA concentration was lower than normally encountered by nasal axons. This is one manifestation *in vivo* of the tolerance of the map formation mechanism to variations in overall ephrinA concentrations. We show in figure 5.8E and F that our model of map formation displays the same form of tolerance to gradient variability.

### Expansion and compression

Early experiments involved removing part of the retina (Schmidt et al., 1978) or part of the tectum (Finlay et al., 1979; Cook, 1979), and observe the resulting map. In the case of removing part of the retina, it was observed that the remaining RGC axons innervated the entire tectum, maintaining topography, giving rise to an "expanded" map. It the reverse case, where part of the tectum was removed, it was found that the RGC axons all ended up targeting the remaining tectal region, also this time maintaining topography, leading to a "compressed" map.

The expansion experiment we simulate by reducing the number of RGCs by half, and also reducing the difference in EphA levels across the horizontal axis of the retina by half. The compression experiment is simulated by halving the size of the experiment in the horizontal dimension while doubling the density of RGCs, thereby reducing the size of the target area and the gradients within it while maintaining the number of RGCs and their gradient. See figure 5.9 for the results of the expansion and compression experiments.

### Compound eye

In experiments by Gaze et al. (1963) and Hunt and Jacobson (1973), animals (frogs in both cases) were surgically altered so that their retina consisted of identical half-retinas mounted back-to-back, creating a retina with symmetric Eph receptor gradients. In different individuals, two nasal, temporal or dorsal half-retinas were used, giving retinas in which EphA increased or decreased toward the vertical midline (double nasal and double temporal

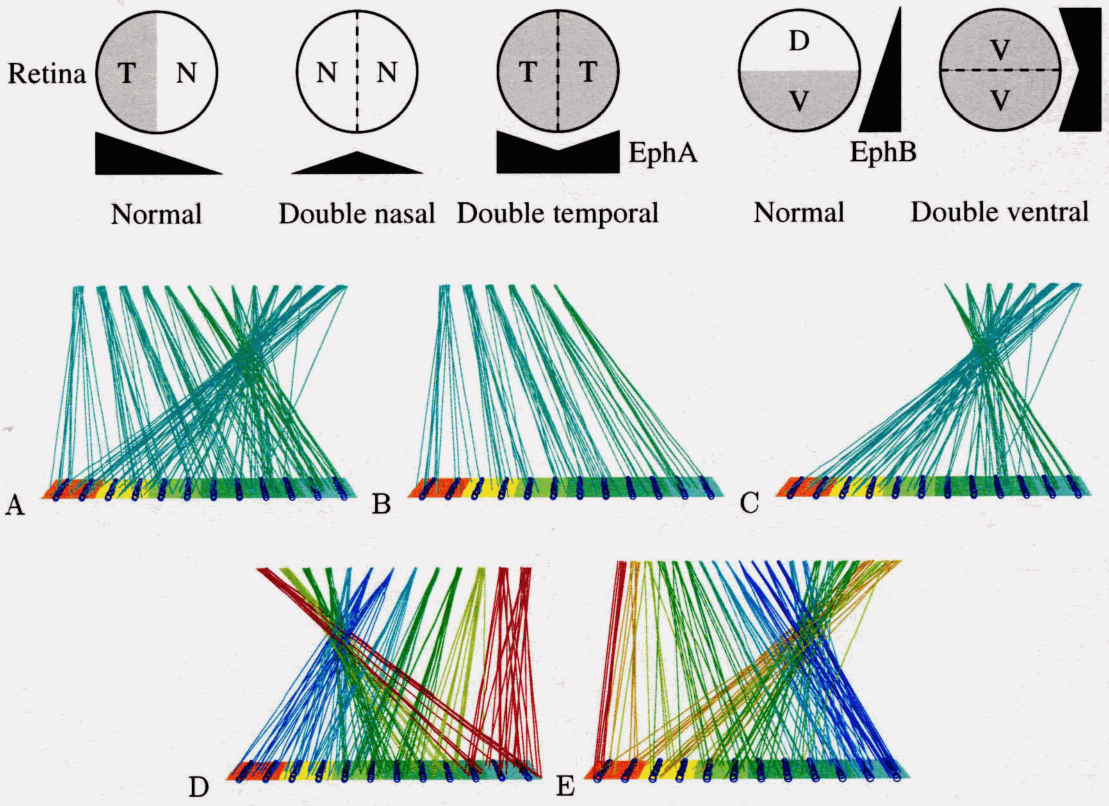


Figure 5.10: Experiment in which the retina consists of two nasal half-retinas, so that opposite gradients exists in the left and right half of the retina. A–C: Simulation of the double nasal case. A: All axons are shown; B: Only the cells in the left hand half of the retina are shown, they form a non-crossed projection; C: Only the cells in the right hand half of the retina are shown, forming a crossed projection. D: Simulation of the double temporal case. E: Simulation of the double ventral case. Axons are coloured according to their EphA concentrations (A–D) and EphB (E). The tectum is coloured according to the ephrinA concentration (A–D) and ephrinB (E).

respectively), or in which EphB decreased toward the horizontal midline (double ventral), see figure 5.10, top. The rest of figure 5.10 shows results of NeuroGene simulations with retinas altered in corresponding ways. Figure 5.10A–C shows the projection from a double nasal retina. Figure 5.10A shows axons originating from both half-eyes, while figures B and C show each of the two half-eye projections separately. Each half-eye forms a complete projection of uniform density covering the whole of the tectal target, causing half the eye

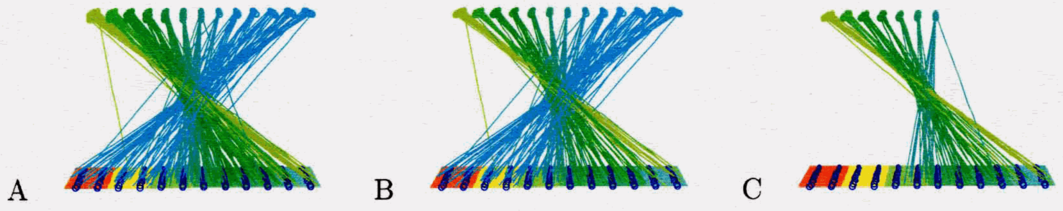


Figure 5.11: Experiment in which two eyes innervate the same tectum, one of which is reduced in size. All three images show the same situation; A: all RGC axons are shown; B: Only RGC axons from the intact eye are shown; C: Only axons from the partly ablated eye are shown. Note that the map in B is not expanded in the area where axons from the other eye innervate.

to form a crossed projection and the other half to form a non-crossed projection. Figure 5.10D–E shows the projections formed by two temporal half-retinas, and by two ventral half retinas, respectively. As is observed in the *in vivo* experiments, the axonal level of EphA and EphB determine the termination point of RGC axons.

### One-and-a-half eye

The following experiment, carried out in goldfish, offers an instructive complement to the expansion experiment described above: Half the retina was removed from one of the eyes, and the nerve from this eye was diverted so that it innervated the tectum which was already innervated by the nerve from the intact eye (Schmidt et al., 1978). It was found that the axons from the reduced eye made connections covering only half the tectal target area, specifically the tectal area which was already innervated by axons from the corresponding half of the intact eye. This shows that the axons from the intact eye prevent the axons from the half-eye from innervating the entire tectum. In figure 5.11 we show the results of simulations of this system using our model of map formation. In figure 5.11A is shown the projections from both eyes, in 5.11B only from the intact eye and in 5.11C only from the manipulated eye.

**Eph knock-in**

In an elegant genetic experiment Arthur Brown and coworkers (2000) caused a subset of retinal cells to express a higher than normal amount of EphA receptors. This subset was dispersed randomly throughout the retina, and the increase in EphA receptor concentration above their "wild-type"-levels was the same for all these cells. This gives rise to a population $P_w$ of RGCs which express their wild-type level $C_w^x$ of EphA corresponding to their location $x$, and a population $P_{ki}$ which express a level of EphA which is higher by an amount $C_{ki}$, i.e., $C_w^x + C_{ki}$, where $C_{ki}$ is constant for all RGCs in $P_w$. Brown et al. found that all the RGCs in both populations projected their axons to locations within the normal extent of the tectal target area. However, the two populations formed two distinct, non-overlapping maps. One map was formed by the $P_w$ population, which lay to the posterior (i.e. at higher ephrinA levels) than the map formed by the $P_{ki}$ population. The important conclusion from this experiment is that it is the relative, not the absolute, level of EphA receptor carried by the RGCs which determine their ultimate termination zone in the tectum. See figure 5.12 for the results of our simulation of the EphA knock-in experiments. The colours of RGC axons in figures 5.12A–J reflect their EphA concentrations, while 5.12K shows the RGCs in $P_w$ as blue and those in $P_{ki}$ as red. Figure 5.12L shows that the sorting in the vertical dimension is not affected by the higher levels of EphA.

The experimental system devised by Brown et al. allowed them to vary the amount $C_{ki}$ by which the EphA concentration was increased by inserting one or two copies of an altered gene. They got qualitatively different results in the two kinds of experiments. In the low-EphA knock-in experiment, they found that the two maps overlapped to a much greater extent than in the high-EphA knock-in experiment described above. They also found that the two maps, while clearly distinct in the posterior tectum, fused into one in the anterior tectum (Brown et al., 2000, figure 5). To investigate this effect, we have run a 1D version of the experiment shown in figure 5.12, with either high (100% of maximal wild-type EphA levels) of low (30% of max) levels of EphA knock-in. We chose to use 1D experiments here, since they tend to converge better. We also altered the orientation of the tectal ephrinA gradient so that an ideal projection does not require axons to cross over — this makes the resulting images easier to interpret. The results of these experiments are shown in figure 5.13. Figure 5.13A shows the wild-type experiment for reference. The projection forms a

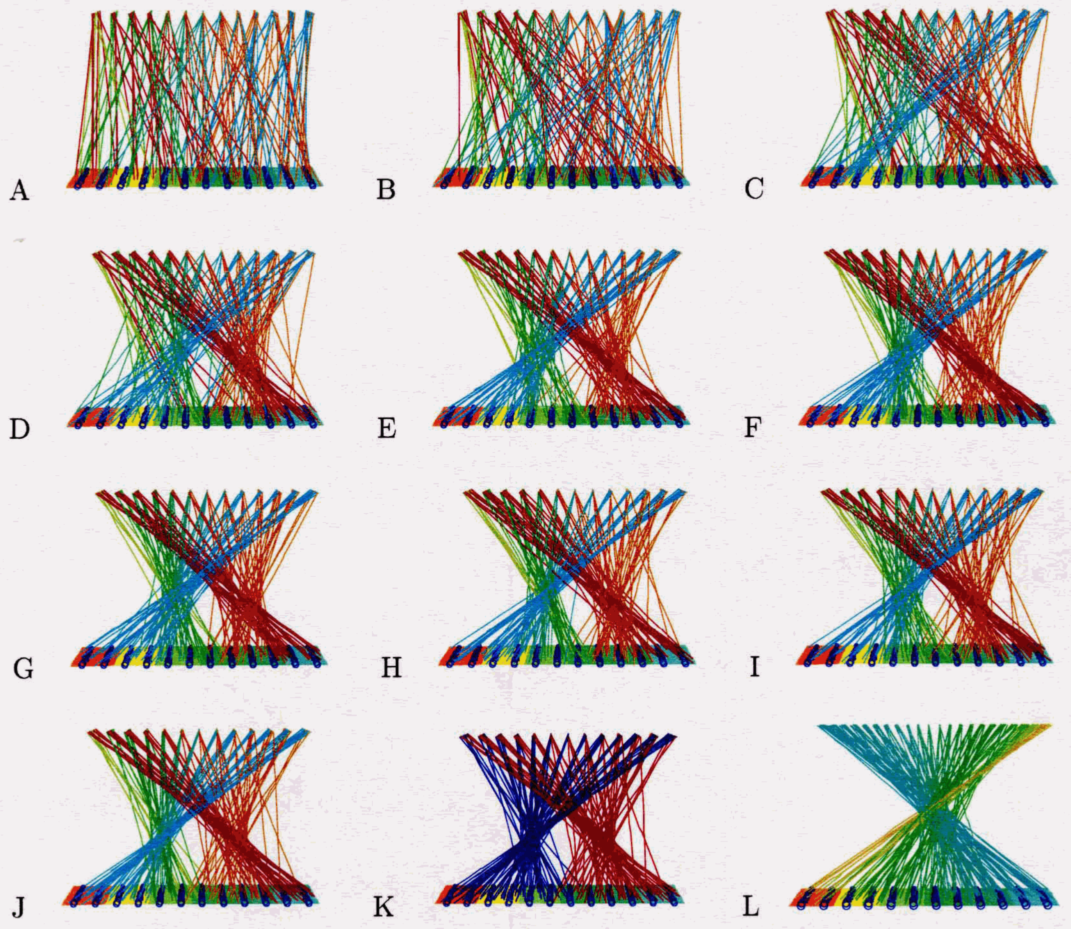


Figure 5.12: Simulation of the topographic map formation with the experimental modifications after Brown et al. (2000). EphA knock-in experiment. A-J are images shown for 2,000, 6,000, 12,000, 20,000, 50,000, 70,000 and 80,000, 90,000, 100,000, 130,000 time steps. K is the same image as J, except it is co-loured for the level of knock-in EphA. L shows the same situation as J and K, but rotated 90 degrees and coloured for EphB and ephrinB.

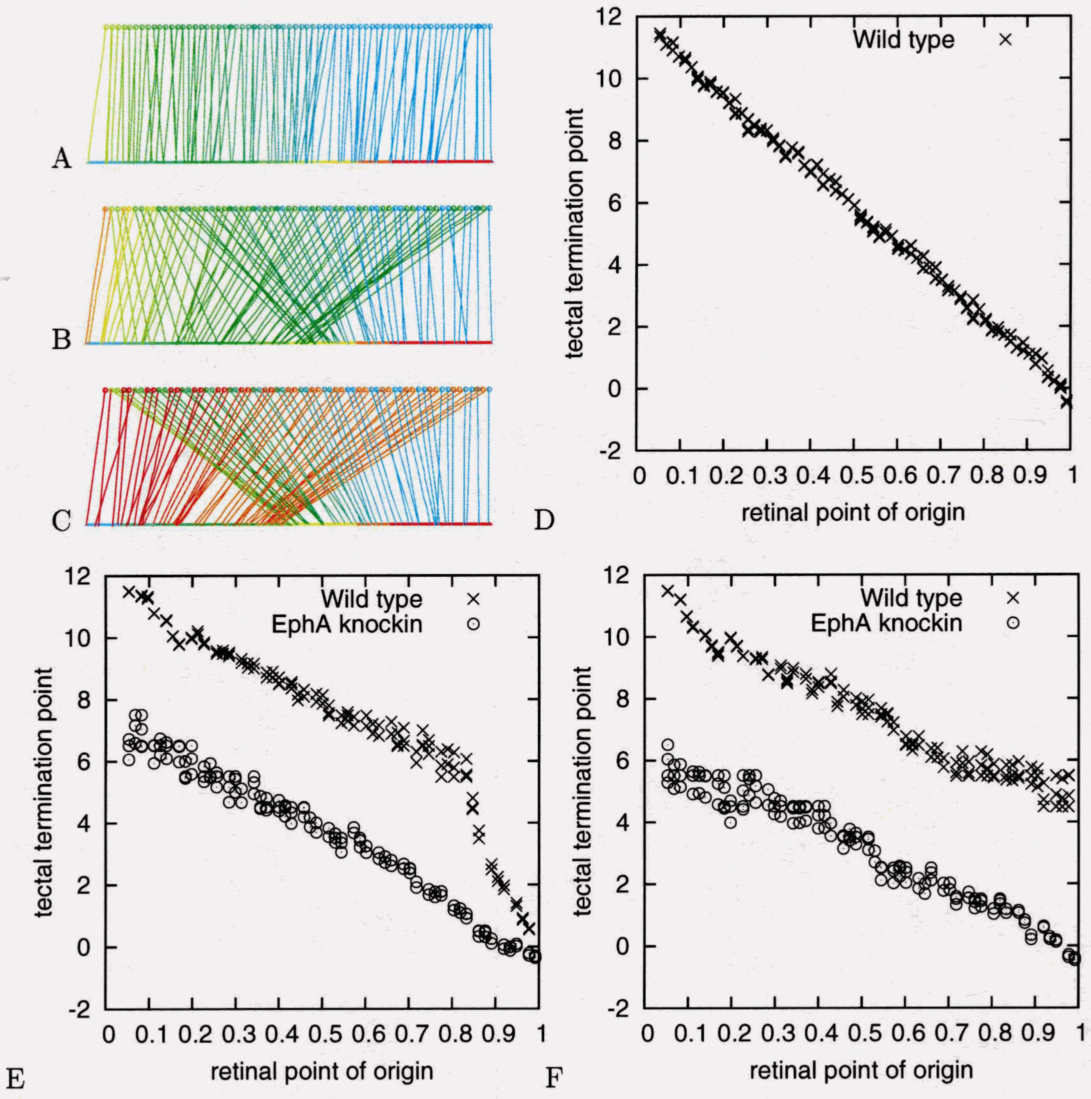


Figure 5.13: Results from simulations with varying levels of EphA knockin. These figures can be compared directly to experimental results shown in Brown et al. (2000), figure 5.

single map on the tectum. This is also shown in figure 5.13D, where the termination point of each axon is shown as function of the location of the RGC along the retinal axis. Figure 5.13C and F shows the projection with 100% EphA knock-in. The two distinct populations of RGCs form two non-overlapping maps on the tectum. The experiment of the intermediary level of EphA knock-in (30%) is shown in figure 5.13B and E. Here the two populations of RGCs form clearly distinct maps only in the posterior (high ephrinA) portion of the tectal target area (to the right in figure 5.13B and in the upper portion of 5.13E. In the anterior tectum the maps converge. This is very similar to what was found by Brown et al. It is likely that activity-driven refinement could cause the maps to fuse into one in the anterior tectum, as was seen by Brown et al., while maintaining the distinct maps in the posterior tectum.

To our knowledge, these findings have not yet been matched to this level of precision by any simulated map formation model. All models of map formation which we are aware of are summarized in appendix G. Of those models that match a range of experimental observations comparable to that matched by our model (Whitelaw and Cowan, 1981; Fraser and Perkel, 1990; Weber et al., 1997), they match only poorly the biological realities of the retinotectal system: Whitelaw and Cowan's system is based on synaptic learning and ignores axon guidance, and relies on parallel rather than anti-parallel chemical gradients. Fraser and Perkel use simulated annealing to form their maps. Weber et al.'s system is formulated as a set of equations solved using Euler's method. Other systems which incorporate more biological detail (Hope et al., 1976; Overton and Arbib, 1982; Gierer, 1983) fail to reproduce important experimental observations: Neither Hope et al. nor Gierer attempt the compound eye, one-and-a-half eye or mismatch experiments. The arrow model (Hope et al. 1976, extended by Overton and Arbib, 1982), which is closely related to our model, generally fares better: They successfully simulate both the compound eye and mismatch experiments, albeit in 1D only, at the cost of introducing several additional parameters without clear biological interpretations. The recent model by Yates et al. (2004) incorporates more biological detail than our model, including axon branch formation. They are able to match both EphA knockin and ephrinA knockout experiments. However, their model, which is implemented in a 1D version only, does include a large number of parameters without obvious biological interpretations.

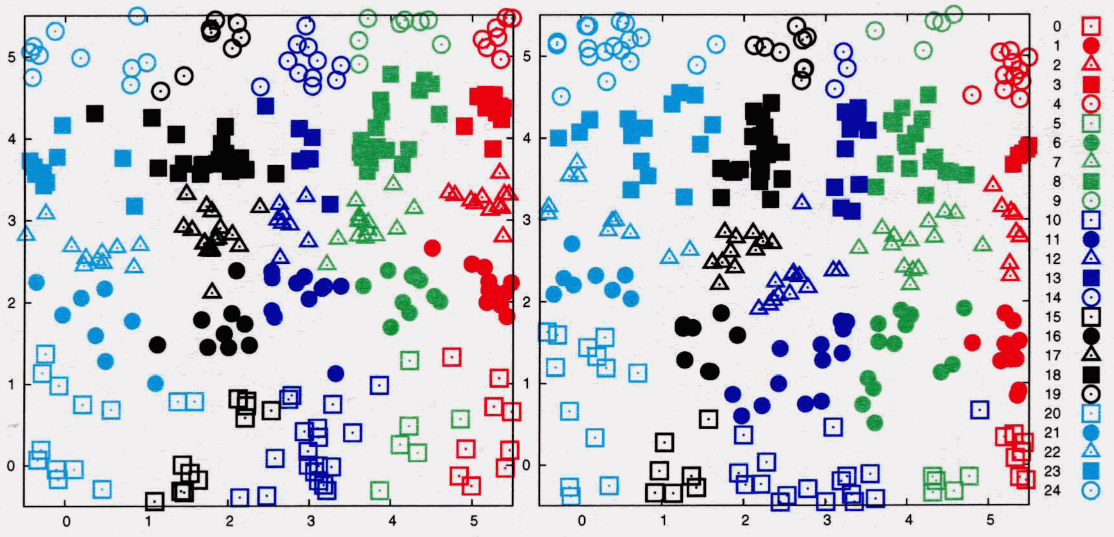


Figure 5.14: Projection of 25 distinct populations of nasal receptor cells onto a target area representing the nasal bulb. EphrinA is increasing from left to right and ephrinB is increasing from bottom to top.

### The nasal projection

The Eph/ephrin signaling system is also involved in the formation of topographic maps in the olfactory system (Knöll et al., 2001). However, this system differs in several ways from projections in the visual system. This makes it interesting to investigate whether we are able to simulate topographic map formation in the olfactory system using the same model of map formation.

The olfactory system consists of approximately 1,000 different primary receptor cell types. Each cell type is sensitive to one particular odorant molecule (St. John et al., 2002). The receptor cells are distributed randomly across the nasal surfaces which are exposed to the air stream through the nose. Each receptor cell projects an axon to the target area in the olfactory bulb in the brain. This projection is such that all cells belonging to one of the $\sim 1,000$ cell populations project to one or usually two distinct and invariant (between individuals) locations in the target area. The resulting projections are thus not parallel as in the visual system, but in stead convergent on small target areas specific for each cell type.

Figure 5.14 shows the results of simulating the nasal projection from 25 distinct populations of nasal neurons arranged in a single sheet just like the retino-tectal simulations.

All neurons within each population have identical concentrations of EphA and EphB. These concentrations, $C_{EphA}$ and $C_{EphB}$, are related to the number $N_p$ identifying each population (see legend, figure 5.14) as follows:

$$\begin{aligned} C_{EphA} &= A\exp(Bx) \quad \textit{where } x = 0.1 + 0.2\mathrm{div}(N_p,\, 5) \\ C_{EphB} &= A\exp(By) \quad \textit{where } y = 0.1 + 0.2\mathrm{mod}(N_p,\, 5), \end{aligned}$$

and $A$ and $B$ have the same values as in the "wild-type" retinotectal simulation, $\mathrm{div}(x, y)$ is the largest integer $\leq \frac{x}{y}$, and $\mathrm{mod}(x, y)$ is the modulus given by $x - y\mathrm{div}(x, y)$. This means that for example all neurons belonging to the population with $N_p = 16$ will have the same EphA and EphB concentrations as a retinal neuron in the retinotectal simulation with fractional coordinates $x$ = 0.1+0.2div(16, 5)= 0.7 and $y$ = 0.1+0.2mod(16, 5)= 0.3. The level of EphA and EphB carried by an axon here depends not on the location of the neuron within the nasal surface, but rather on which population the neuron belongs to, which in turn is determined by which odorant receptor is carried by the neuron.

Nasal neurons were randomly assigned to the different populations. As is shown in figure 5.14, the map formation model causes axons from the same population to cluster in the target area, in spite of the fact that they originate in disparate locations on the nasal surface. On the left is shown a simulation where all populations are of approximately the same size. On the right populations 3, 5, 9, 14 are reduced in size, and population 18 is larger. Increased variability in the sizes of populations does not disrupt map formation, although it is notable that large populations do not appear to lay claim to larger areas of the target than smaller populations.

This experiment shows that the map formation model performs well under conditions that are markedly different from the retinotectal projection, but which are similar to projections which are known to involve the same signaling molecules in nature.

### 5.2.4 Discussion

The experiments above show that the model of map formation based on EphA "pecking-order" is able to replicate a wide array of phenomena observed in topographic projection *in vivo*. It also shows NeuroGene as a powerful tool in the investigation of axon guidance as a developmental process. Part of the strength of NeuroGene is that it allows the same genetic system to be tested in the context of numerous different geometric contexts, differing both in the spatial arrangement of cells and their initial genetic state as defined by their protein concentrations.

We make the assumption that growth cone detect concentration gradients by comparing the concentrations of signal molecule at the tips of filopodia and at their base. This assumption is backed by theoretical arguments (Goodhill and Urbach, 1999) but no direct observations. We also assume that filopodia individually integrate the various chemical signals depending on the nature of those signals, such that the presence of different neighbouring RGC axons cause filopodia to respond differently to the signals they receive. This is supported by the observation that changes in growth cone behaviour may be induced by contact with a single filopodium (O'Connor et al., 1990; Palka et al., 1992). Finally, we assume that the area of the tectum covered by each RGC axon remains constant throughout the simulation. RGC axon branching, which is known to occur in nature, may cause some RGC arbors to grow at the expense of others. The model by Yates et al. (2004) takes such branching into account, and constant RGC arbor size is an emergent property of the model.

### Role of axon-axon interactions

The importance of axon-axon interactions in the formation of topographic maps was revealed by the experiments of Brown et al. (2000). Without such axon-axon interactions, it is very difficult to understand how, in the context of this Eph knock-in experiment, an axon with the normal level of EphA should project to a location much more posterior than normal. Only by being able to detect the larger than normal number of axons with higher Eph level than its own, can it make such an adjustment. In principle such a detection might be mediated through some change in the tectal target induced by the invading axons — however, a simpler mechanism, involving direct axon-axon interactions, must also be considered. In our model, axon-axon interactions have two effects: They lead to the proper sorting of axons based on EphA levels through the mechanism of a pecking order, and they cause the approximately even spacing of axons across the target area.

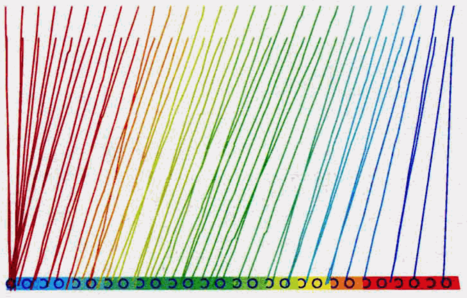


Figure 5.15: A: Neurotrophic mechanism of axon-axon repulsion at 9000 time steps (steady state).

### Role of axon-axon repulsion

It might seem like the requirement of creating a map with uniform density might be met simply by making axons repel one another, as suggested by Flanagan and Vanderhaegen (1998, p. 325). In order to investigate the effect of axon-repulsion, 1D map formation experiments were run in which repulsion was implemented as follows: a neurotrophic[2] protein, which we named *BDNF* after an actual gene (*brain-derived neurotrophic factor*, which perhaps could play a similar role, *ibid.*), was expressed uniformly by all tectal cells. Invading RGC axons absorb this protein at a uniform rate. Consequently the protein becomes depleted where there are numerous axons. By making this protein attractive to the RGC axons, these are attracted to areas where there are fewer RGC axons, which amounts to a general repellent interaction among RGC axons.

The result of this simulation in 1D is shown in figure 5.15. The resulting map is generally of uniform density, showing that the neurotrophic model is effective in maintaining axon-axon repulsion. However, there is a clear tendency in the density of the map, decreasing steadily from the attractive low-ephrinA end of the map (to the left) to the high-ephrinA end of the map (to the right). The image shows the steady-state situation, meaning that each axon is at a point of equilibrium: for each axon, the inherent tendency to move down the gradient, away from the repellent ephrinA, is balanced by an equal tendency to move up

[2]"Trophic" is derived from the Latin *to eat*. However, neurotrophic signals do not literally support the life function of neurons, they are signals which tell the neuron not to die but to grow.

this gradient. This latter tendency can only come from axon-axon repulsion. For axon-axon repulsion to create a net tendency to move in one direction, the distribution of axons must necessarily be non-uniform. More specifically, the axon density must be higher on the left in order to create a net tendency for axons to move to the right. Such an axon distribution causes a faster consumption of the neurotrophic factor on the left, generating a concentration gradient increasing to the right. This in turn generates the tendency for axons to shift to the right, a tendency that at steady state is exactly balanced by the tendency to move to low ephrinA.

The problem revealed by the experiment shown in figure 5.15 is not specific to the neurotrophic BDNF-based implementation of axon-axon repulsion, but inherent in axon-axon repulsion as the force opposing wholesale axon migration to the left. We also discovered other problems arising from repellent interactions among axons, such as the difficulty with which mis-sorted axons pass by each other in order to form a proper topography — in order for two mis-sorted axons to swap places, they must come into close proximity with one another, which is difficult when there is a strong repellent force acting on them.

Our model presented earlier establishes how a more sophisticated axon-axon interaction leads to a sorted topography, rather than hindering its formation. While under a repellent scheme axon-axon interactions would cause an axon to move away from other axons, in our model, axon-axon interactions cause certain axons to move in a posterior direction, regardless of the relative locations of the interacting axons. As shown by our simulations, this form of axon-axon interaction can form basis of a topographic sorting mechanism, without the drawbacks manifested by a simple repellent interaction.

To our surprise, we found that while our map formation model enforces complete and uniform innervation of the entire available tectal target area, it does not do so by equal spacing of axons. Rather it causes equal spacing of EphA levels in these axons. This can be seen in the one-and-a-half eye experiment (see figure 5.11) where the additional axons from the manipulated eye does not cause local expansion of the map from the intact eye, instead causing local variability in the density of axons within the target area. Similarly, the number of axons within each of the populations in the nasal projection does not determine the area occupied by these populations within the target area. In both cases, this can be explained by the fact that RGC axons with equal EphA concentrations do not interact with each other under our model.

### Complete coverage of the tectum

This same process also ensures that the map covers the entire target area, as is seen in all of the experiments reported here: High EphA axons will move the anterior as far as they can go, precluding any gaps in the map at the anterior edge. Conversely, low EphA axons will be pushed to the posterior as far as they can go. All other axons wind up in positions where their inherent tendency to move in the anterior direction is opposed by the presence of higher EphA axons. Our experiments show that when the majority of axons have found such a point of balance, a properly sorted map of approximately uniform density results (see e.g. figure 5.7). In this context it is interesting to compare to the experimental finding that a minimum of 10-20% of the normal number of RGC axons are needed for a topographic map to form[3]. In our model, if too few axons reach the tectum, their spacing would be too large for axon-axon interactions to be effective, and the map formation mechanism fails.

### Roles of Eph receptors and ephrin in retina and tectum

In the early days of the investigation into the phenomenon of topographic map formation, a relatively simple picture emerged: The tectum expresses ephrinA, RGC axons express EphA which are receptors for ephrinA, and through the interactions between these molecules, the map is formed. Several new discoveries have revealed new complexities in this picture:

- EphrinA (Davy et al., 1999) and ephrinB (Hindges et al., 2002; Mann et al., 2002) may act as receptors of EphA and EphB, respectively, giving rise to *bidirectional* signaling between cells carrying ephrins and Eph receptors. When the EphA of cell $A$ binds ephrinA on cell $B$, bidirectional signaling causes events (possibly leading to changes in gene expression) within *both* cell $A$ and cell $B$.
- RGC axons express ephrinA and B in gradients which go in the opposite direction to the EphA and EphB gradients (Hornberger et al., 1999), i.e. ephrinA is highly expressed in the nasal retina where the EphA levels are low, while ephrinA levels in the temporal retina are low.
- Tectal cells similarly express EphA and EphB in gradients that are opposite to the tectal gradients of ephrinA and ephrinB (Connor et al., 1998).

[3]Dr. Sara Dunlop, personal communication, see footnote 1 on p. 158.

- Eph receptors and ephrin can interact "cis" as well as "trans" (Feldheim et al., 2000; Hornberger et al., 1999), meaning that Eph receptors and ephrin which are bound to the surface of *the same* cell may interact with each other. Such interactions are biologically significant, at least in RGC axons (Hornberger et al., 1999).

Our model is consistent with many of these observations. The ephrin expressed by RGC axons may be the receptor which is used to detect the Eph receptors carried by other retinal axons. The cis-interactions between ephrin and Eph within the same cell membrane may be involved in the comparison of the Eph levels between self and other RGC axons, allowing the cell to directly compare the amount of interaction through concurrent cis and trans interactions. This also fits our model, in that nasal axons (high in ephrinA) are sensitive to temporal axons (high in EphA), while temporal axons (low in ephrinA) are oblivious to nasal axons (low in EphA). This hypothesis relies on the assumption that the sensitivity of cells to EphA increases with the concentration of the receptor to EphA, i.e., ephrinA. If this assumption is correct, it would be a repeat of the relationship that has already been established for the opposite interaction, in which increased EphA levels increase RGC axons' sensitivity to ephrinA.

It is not clear what role is played by the EphA expressed by the tectum. However, there is some evidence that tectal EphA does not actually play any role: Studies have shown that altering the expression of EphA in the tectum does not disrupt the formation of topographic projections (Knöll and Drescher, 2002).

### Binary behaviour gives resilience to changing conditions

As shown above, the model of topographic map formation works under a wide array of conditions, including variations in the geometric arrangement of cells as well as the genetic state of RGC axons. The key to why this model shows such resilience is the simple binary behaviour of the growth cones. The behaviour depends on comparisons with neighbouring axons, causing one behaviour if the neighbour is above in the "pecking order", and another if it is below. *How much* above or below its own level is not relevant. From the comparison, the growth cone makes a binary decision, either to move up or down the ephrin gradient. This can be restated as follows: Depending on the state of the growth cone, the tectal ephrin gradient is either attractive or repellent, and each growth cone may rapidly switch back and forth between being attracted or being repelled by this gradient.

Such a binary behavioural pattern of the growth cone is consistent with a number of experimental observations: As mentioned in section 2.6, growth cones may interpret the same signal as attractive or repellent depending on the concentration within the growth cone of the signaling molecule cAMP (Song et al., 1997; Ming et al., 1997). cAMP is an important intracellular signaling molecule used in many different contexts, and its concentration can change rapidly, e.g. in response to external stimuli. Such rapid change in behaviour is seen when growth cones reach the so-called "guidepost cells" in the grasshopper limb bud (O'Connor et al., 1990). Here, the growth cone switches from a behaviour where it follows one signal and disregards another, to a behaviour where it ignores the first and follows the second in stead. While the ephrinA/EphA interaction is typically repellent, it is also known to mediate attraction in the olfactory system (Knöll et al., 2001, 2003). Similarly, the EphB/ephrinB interaction can be attractive or repellent, depending on the location of the axon relative to the axon's proper termination zone (McLaughlin et al., 2003a). Together these findings show that every aspect of our map formation model find support in experimental observations.

### 5.2.5 Conclusion

As shown by the experiments reported above, NeuroGene includes the functionality required to simulate complex axon guidance processes. By explicitly simulating axonal growth cones with filopodia, we have developed a new model for the activity-independent formation of topographic maps within the vertebrate visual system. This model takes into account the receptors and ligands carried both by RGC axons and tectal target cells. It accounts for a number of observations from experiments involving disruption of the developmental process through surgery or genetic manipulation. Since the model is expressed in the biological terms of the NeuroGene genetic language, the model relates directly to the biological interactions as they occur *in vivo*, such as protein expression and perception, growth cone motility, etc. NeuroGene has proved a powerful tool for investigating the properties of the model, in particular its behaviour under subtly different conditions, such as variations in gradient shapes and magnitude, axon density, etc. We have also been able to show that the model can form the topographic projections of the olfactory system.

The simulated genes which form the basis of the simulation of topographic map formation fall into three categories: Simulated genes which represent actual genes known to exist in nature, simulated genes for which genes with corresponding functions can be inferred to exist in nature, and finally genes for which corresponding genes may not exist in nature, but whose properties and behaviour fall within the gamut of what has been observed in nature. In the first category fall the EphA/B and ephrinA/B genes. These are known to exist in nature, and they are known to be expressed in concentration gradients across the source and target areas, respectively, of topographic projections. This expression pattern is replicated within the NeuroGene simulation. The gene *knockin*, which is used to implement the EphA knockin experiments, represents the gene *Isl2*, which was manipulated by Brown et al. (2000), and therefore also falls in this category.

In the second category fall the genes defining the cardinal axes of the retina and the tectum. The regulatory systems underlying the expression pattern of ephrinA/B and EphA/B in nature are only now beginning to be mapped out (Díaz et al., 2003; Takahashi et al., 2003). In order to simulate the expression behaviour of these genes, a simple framework of four regulatory genes is postulated. These postulated genes are expressed in linear concentration gradients defining two perpendicular axes in each of the retina and the tectum. Since these proteins do not decay, their concentrations are set within each cell at simulation initialization, and subsequently never change. These simulated genes fulfill the roles of regulatory genes which we can infer exist in nature, but whose properties, and precise mechanism of affecting EphA/B and ephrinA/B expression, are unknown. Another gene in this category is *retinoTectalGradient*, which represents the guidance cue or cues which lead RGC axons from the retina to the tectum in nature.

Cell identity genes are in the last category: To our knowledge, no single gene is known to be expressed only in RGCs or only in tectal cells. However, different types of cells are commonly distinguished by the genes which they express (see section 1.5.2). Postulating genes which are expressed only in RGCs or tectal cells, and which determine their behaviour, is therefore in line with phenomena which are commonly observed in nature.

The three simulated genes *growRgcAxon, FindTectum* and *SortMap* do not represent genes in nature (see appendix B.2.2, lines 104, 115 and 138 respectively). These simulated genes have the purpose of triggering *actions* — either the formation of an initial axon or the creation of axonal growth cones using different growth cone functions. By placing these actions in distinct genes, the actions are triggered by the state of the cell and the concentrations of proteins, rather than the rate of expression of other genes. These simulated genes then represent *molecular interactions* (as specified by their regulation clauses) which cause changes (specified in their effects clauses) in cellular structure and behaviour. These interactions are encoded in the form of genes in order to overcome the limitation in the NeuroGene gene model (see section 2.5.2 – Gene effects). It follows that the concentrations of the *growRgcAxon*, *FindTectum* and *SortMap* proteins have no influence over any aspect of the simulation[4].

The NeuroGene genetic framework is closely modeled after the mechanisms of genetic and developmental control in nature. This means that where detailed biological knowledge exists, it can be directly incorporated into the simulated genes which control the developmental simulation. This includes the expression profiles of proteins which establish concentration gradients used to guide RGC axons within the tectal target area, as well as RGC growth cones' reaction to encountering ephrinA and ephrinB gradients. Where biological knowledge does not exist, but where biologically plausible mechanisms can be inferred or postulated, these can also be incorporated to give a possible account for the developmental process as a whole.

## 5.3 Activity dependent development: Ocular dominance and refinement

The third simulation we present involves activity-dependent developmental processes, and showcases NeuroGene's neuro-genetic coupling. The developmental system we simulate is not accurately described as *learning*, although it does rely on modification of synaptic conductances according to a Hebbian learning rule. This distinction can be expressed as follows: In learning, the synaptic changes are related to the experiences of a particular individual, while in activity dependent development, the result of development is the same in all individuals, as long as their experiences fall within what can be considered *natural.*

As an example, consider the process of imprinting in birds: A wide range of birds will associate "mother" (a concept that is somehow inherent) with the first moving thing they

[4]Each of these genes is labeled `terminal` in the gene script. This means that the proteins produced by these genes are not actually represented within NeuroGene. This is done for performance reasons, but the `terminal` tag also indicates that a gene is not involved in the regulation of any other genes.

see after birth. This association is life-long and cannot be changed. In nature this is normally the actual mother of the young bird, but experiments have shown that birds can even be imprinted on a balloon. Here, the result of what must be a genetically predetermined process is highly dependent on exactly what it is that the bird sees, making this an example of learning. By contrast, consider human infants born in a modern hospital, in a South-American jungle, or in an arctic igloo. Each child experiences completely different visual images. However, these visual stimuli drive developmental mechanisms of visual field refinement, as simulated below, which ultimately give rise to mature visual systems which are functionally indistinguishable. This is possible because development only relies on the common properties of the visual images experienced by each, such as spatial and temporal coherence, and is not affected by the particulars of their disparate experiences. Only exposure to *unnatural* visual experience, such as being raised in the dark (Takacs et al., 1992) or under strobe light (Brickley et al., 1998), can alter the activity dependent developmental program.

Some activity dependent development is driven by endogenously generated neural activity, which ultimately originate in spontaneously firing neurons. In such cases, influence of experience from the external world can be ruled out. Yet, in some cases, processes which are driven by such endogenous activity early stages, are driven by external stimuli later in development, such as after birth and eye-opening.

Activity dependent development may then be seen as yet another means by which genes exert control over the developmental process. This applies to endogenous neural activity, since tendencies for neurons to fire spontaneously is under genetic control. It also applies to exogenous neural activity, since a general anticipation of what constitutes "natural" external stimuli can be incorporated into the genome through natural selection.

A general tool for the simulation of neurodevelopment such as NeuroGene must be able to simulate these types of processes. NeuroGene incorporates a framework for the simulation of neural activity, including the ability to support multiple neural models at the same time, as well as bidirectional coupling between neural activity and gene expression, see section 2.7.1. This allows activity dependent and independent developmental processes to be investigated within the same simulation.

In order to showcase these capabilities, we have simulated the activity dependent refinement process of the topographic map, which also causes formation of ocular dominance columns. These processes occur after the initial retinotectal connections have been formed by activity independent processes based on axon guidance, as discussed above. In the activity dependent stage of topographic map formation (Debski and Cline, 2002; Goodhill and Löwel, 1996), the initial map is refined through a process in which synaptic weights are adjusted, new synapses formed and inappropriate synapses removed, leading to a highly ordered topographic map. Resulting from this process, each RGC connects to a very small number of target cells, and adjacency-relationships between RGCs is maintained in their connections. Abolishment of neural activity during this stage causes the map to remain in an unrefined state into adulthood, with diffuse RGC connections presumably giving rise to blurry vision (Gnuegge et al., 2001; McLaughlin et al., 2003b).

The source of the neural activity driving this process depends on the species in question, in particular on the timing of map refinement relative to the time of birth and eye-opening. In animals in which the refinement process occurs before eye opening, the process is driven by endogenous neural activity originating from spontaneously firing cells in the retina. In animals for which eye-opening occurs at a developmentally earlier stage, external visual stimuli drive the refinement process. We have simulated the refinement process both using endogenously generated neural activity and simulated visual stimuli.

The endogenous activity in retinal neurons prior to eye-opening takes the form of waves traveling across significant stretches of the retina (Wong, 1999). All RGCs which fall within the area covered by an activity wave, fire. At any point in time only a small fraction of RGCs fire — this means that two RGCs which fire at the same time are likely to be located close together in the retina — this information is extracted by the tectal cells and used to drive the refinement process. Disruption of these waves causes disruption of refinement and ocular dominance column formation (McLaughlin et al., 2003b). Simulations have shown that retinal waves in conjunction with Hebbian learning (Hebb, 1949) can establish highly refined topographic connections with ocular dominance (Swindale, 1996). See figure 5.16 for a schematic representation of how activity waves form basis of the refinement process.

Activity-driven changes in the retinotectal map simultaneously effect another kind of change: The formation of ocular dominance columns. This process causes each tectal neuron to receive input from only one eye, where initially all tectal cells receive approximately equal input from both eyes. Adjacent tectal cells tend to be dominated by the same eye, leading to contiguous areas of the tectum dominated by one eye or the other. For a review of earlier

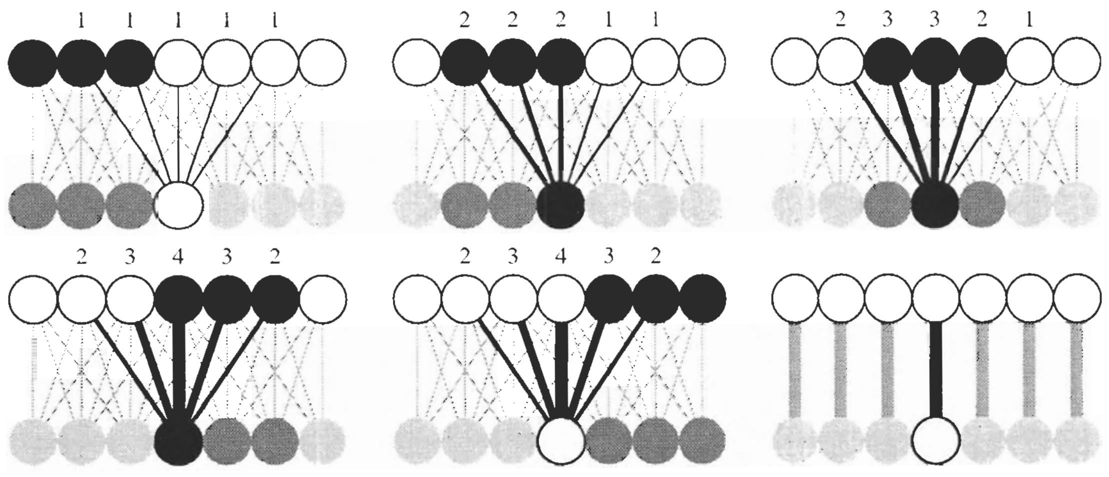


Figure 5.16: Refinement by moving waves and Hebbian learning. Each tectal cell (bottom row, one cell shown in detail) is connected to five retinal cell (top row) each with a connection strength of 1. Each tectal cell needs input from three or more retinal cells before it fires. Links between cells which fire in synchrony are strengthened, as represented by an increase in line width and numbers above the retinal cells. After the wave has passed, central links has been strengthened more than peripheral ones. Through normalization and repetition, this leads to refinement, eventually giving as result the topographic projection as shown at the bottom right.

modeling efforts to understand these processes, see Swindale (1996).

### 5.3.1 Simulated retinal waves

Spontaneously generated waves of neural activity travel across the retina prior to eye opening (Wong, 1999). Waves may originate at any point in the retina, and propagate across some portion of it before dying out. Waves often, but not always, die out in regions of the retina which recently supported another wave — this indicates that cells' refractive delays may be involved in regulating the extent of the waves.

Feller et al. (1997) developed a computational model which reproduces many of the features of retinal waves. According to this model, waves are driven by a class of retinal neurons known as "stellate" or starburst cells, named after the characteristic star-shapes of their dendritic structures. At the developmental stage when waves occur, these cells are connected to each other as well as to retinal ganglion cells. Elliott and Shadbolt (1999) used this model to generate waves of neural activity to drive their simulations of refinement, and we have chosen to do the same here.

According to Feller et al., waves are initiated by spontaneous firing of starburst cells. Through connections among the starburst cells, a sufficient number of spontaneous firing cells drive neighbouring starburst cells above their firing threshold which sets off the wave. As the wave propagates, it brings other starburst cells in front of the wave above their firing threshold, thus sustaining wave propagation. After the starburst cells cease to fire, they enter a refractive period during which they are unable to fire. As the wave enters an area of the retina in which a large proportion of starburst cells are refractive, an insufficient number of cells are brought to fire to sustain the wave, and it dies out.

In their model, retinal ganglion cells (which are about 10-fold more numerous in the retina than starburst cells) act as a "readout-layer": Through connections from nearby starburst cells, RGCs are caused to fire when a propagating wave passes by. While some starburst cells may be refractive within a wave, causing "holes" of inactive starburst cells within the wave, the wave of activity in the RGCs will always be continuous, since RGCs do not become refractive. This means that the probability that two adjacent RGCs fire in synchrony is higher than the corresponding probability for starburst cells. This is important, since the refinement mechanism relies on extracting this correlation from the neural activity originating from the waves.

While neural action potentials occur at a millisecond time scale, an elegant study by Butts and Rokhsar (2001) have shown that the information content in retinal waves with respect to the spatial relationships among RGCs exists on a much slower time scale, in the range from 0.1–2 seconds. This means that simulating retinal waves using simple neural models based on average firing states, such as those implemented in NeuroGene, is sufficient for capturing the functional aspects of the retinal waves.

### 5.3.2 Learning algorithm

While it is simple to write a NeuroGene gene which encodes a Hebbian learning rule (see figure 5.17), a more explicitly biological mechanism was used to implement learning in the simulations reported here. The learning rule, taken from Elliott and Shadbolt (1998a,b, 1999, 2002), is outlined in detail in appendix H. This learning rule uses the biological mechanisms

```
postsynapse gene HebbianLearningRule {
   regulation {
      if(cellIsFiring() && partner().cellIsFiring()){
         expressInternally(1.0);
      }
   }
   effects {
      setSynapticWeight(synapticWeight() + delta);
   }
}

postsynapse gene NormalizeWeights {
   regulation {
      expressInternally(1.0);
   }
   effects {
      setSynapticWeight(synapticWeight()/cellIncomingWeight());
   }
}
```

Figure 5.17: Simple Hebbian learning implemented in the NeuroGene gene language. The learning and multiplicative normalization are implemented in separate genes, both of which are expressed exclusively in postsynaptic terminals. Measures would need to be taken to ensure that these genes do not run in the same simulation time step, since different genes are executed in an arbitrary order within each cell component. This could for instance be done using timers, see section 2.5.4.

of ligand-receptor binding (see section 2.4.4) to implement the retrograde cross-synaptic communication needed to implement the Hebbian learning rule. Under this learning rule, when a presynaptic cell fires, it expresses an amount of receptor on its surface (see figure 5.18A). When a postsynaptic cell fires, it in turn produces a certain amount of a soluble protein which may bind to these receptors (figure 5.18B). The fact that a presynaptic cell has receptors with ligands bound to it can then be taken as an indication that both cells have fired in synchrony (figure 5.18C), and as such may be used as a basis for implementation of a Hebbian learning rule.

The learning rule proposed by Elliott and Shadbolt (1998a) exists merely as a postulate at this point, as it has not been shown that Hebbian learning is implemented in precisely this way in nature. However, the learning rule relies on biologically plausible mechanisms.

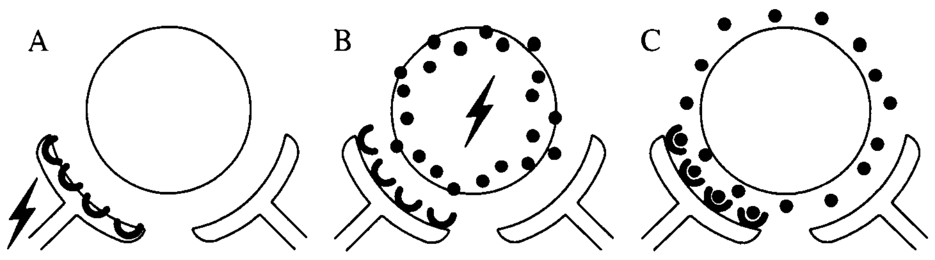


Figure 5.18: Hebbian learning mechanism as implemented using receptor-ligand interactions. One target cell (circle) received input from two axons. A: One of the two axons fires. In addition to the transfer of neural activity to the target cell, this causes the expression of receptor (small semicircles) on the surface of the presynaptic terminal. On the non-firing axon (right) no such expression occurs. Note that the receptors are not involved in the transfer of neural activity across the synapse to the target cell, this occurs through processes which are not shown here. B: As a result of the neural activation received from the left axon, the cell may fire. This causes the cell to express a soluble protein (*NT*, small dots). C: The *NT* ligands bind to the receptor proteins carried only by those axons which have fired. The presence of the receptor-ligand complex on the presynaptic terminal can be taken as an indication of the axon and the target cell firing in synchrony, and can therefore be used as the basis of a Hebbian learning rule.

Neet and Campenot (2001) reviews the evidence for presynaptic binding and internalization of neurotrophic factors. Neural activity triggering exocytosis at the presynaptic terminal forms the basis for synaptic transmission of action potentials, and similar exocytosis has also been observed at the postsynaptic terminal (Maletic-Savatic and Malinow, 1998a,b). Our simulation also rests on general observations of the integration of genetic and neural activity, as outlined in section 2.7.1. The competitive aspect of the mechanism is straightforwardly implemented in NeuroGene using its receptor-ligand binding functionality. The genes responsible for the actual learning algorithm are listed in appendix B, section B.2.3.

## Cell death *versus* no cell death

As outlined in appendix H, the learning rule above comes in two varieties (see equation H.2), one of which gives a competitive advantage to postsynaptic cells whose total incoming synaptic weight is lower than that of other cells (equation H.2, case 2). This competitive advantage ensures that no cells will lose all their connections, in the terminology of Elliott and Shadbolt (1998a), this would constitute the "death"[5] of that cell, and they call this form of the learning rule NCD ("no cell death"). Under the other form of the learning rule (equation H.2, case 1), there is no such competitive advantage, and cells consequently are allowed to "die" — this is the CD ("cell death") form of the learning rule. In Elliott and Shadbolt (1998b, 1999) the NCD form is used, although they do not discuss this choice. We found that using the CD form, a large minority of RGCs would lose all their connections to the tectum and "die" (data not shown). Consequently, we followed Elliott and Shadbolt and used the NCD form in all experiments. It is clear that in nature, neurons, including RGCs, do perish as result of losing all their synaptic connections.

### Ligand-receptor binding

The binding of NT to the NT receptor is governed by the receptor-ligand binding functionality as implemented in NeuroGene. The binding is characterized by the dissociation constant $K_d$, for which we use a value of $10^{-12}$M. This is a typical value for strong ligand-receptor interactions observed for neurotrophic factors in nature (Neet and Campenot, 2001). Experiments showed that the refinement process is not sensitive to variations in this value, giving similar results with weaker binding ($K_d = 10^{-8}$M).

The NT receptor with a molecule of NT bound to it is represented by the protein *NT-complex*. The gene encoding this protein is never expressed (indicated by the fact that it has no regulation or effects section, see lines 89–92). The concentration of *NTcomplex* and *NTreceptor* always reflect the amount of receptor that has a molecule of *NT* bound to it, and the amount that is free of *NT*, respectively, see algorithm 1.

With this machinery in place, the learning rule reduces to the following simple form (see appendix H, equation H.12)

$$s_{xi}^{n+1} = s_{xi}^{n} + \varepsilon\left([C]_{xi} - s_{xi}^{n}\right) \qquad (5.1)$$

where $s_{xi}^n$ is the synaptic weight between retinal cell $i$ and tectal cell $x$ in time step $n$, $[C]_{xi}$ is the concentration of the *NTcomplex* on the surface of the RGC presynaptic terminal connecting RGC $i$ to tectal cell $x$, and $\varepsilon$ is the learning rate. The gene which implements this learning rule is shown in figure 5.19. The regulation section of this gene serves to limit

[5]"Death" is put in quotation marks since Elliott and Shadbolts definition of cell death differs from the one used in NeuroGene, see section 2.2.2.

```
presynapse gene learn {
    regulation {
        if (surfaceConcentrationOf(retinalCellMarker) > 0) {
            expressInternally(0.1);
        }
    }
    effects {
        variable delta = surfaceConcentrationOf(NTcomplex)
                       - synapticWeight();
        setSynapticWeight(synapticWeight() + learning_rate*delta);
    }
}
```

Figure 5.19: The gene implementing the learning rule. The gene is expressed in RGC presynapses only, and requires the retinal cell marker protein in order to be expressed. The protein produced by this gene is not used for anything.

its expression to RGC presynaptic terminals only, while the learning is implemented in the effects section of the gene.

### 5.3.3 Implementation

The simulation involves three cell types, retinal ganglion cells (RGCs), tectal cells, and starburst cells. In the NeuroGene simulation, the cells are arranged in cell-type specific layers as shown in figure 5.20. Note that the arrangement of starburst and ganglion cells in separate layers in figure 5.20 and in the NeuroGene simulation are for clarity only, and does not correspond to the spatial arrangement of these cells within the retina. However, the topology of interconnections between startburst and ganglion cells are accurate to what is found in nature.

The retina consists of simulated starburst cells, each of which has an excitatory connection to all other such cells within a given diameter ($D_{ss}$ in figure 5.20). These cells generate propagating waves of neural activity (Wong, 1999) which are important in driving the activity-dependent development. Simulated retinal ganglion cells (RGCs) receive neural input from all starburst cells within a given diameter $D_{sr}$, and pass neural activity on to simulated tectal cells within. Retinal cells (both starburst and RGCs) are interconnected also across the edges of the cellular sheet so that the sheet forms a torus, i.e., cells near the

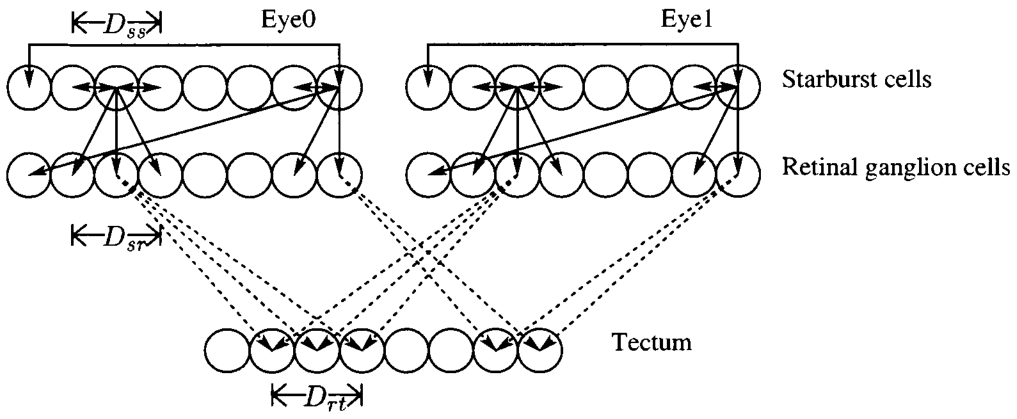


Figure 5.20: Experiment for ocular dominance/refinement with endogenously generated neural activity. Connections whose weight are subject to learning are shown in broken lines. The retinas contain connections spanning opposite edges to give a torus, the tectum does not. The arbor diameters ($D_{ss}$, $D_{sr}$ and $D_{rt}$) are much larger than shown here. Note that the starburst and ganglion cells are shown in separate layers for clarity only — in the retina the cells occupy the same space, but maintain the connection pattern indicated here.

top are connected to cells near the bottom, and the same from side to side.

The neural models implemented in NeuroGene compute the average activity level of each neuron during each simulation time step, see section 2.7. In the following, the details of the neural models used for the various cell types will be outlined.

### Retinal ganglion cells

As outlined in section 2.7.1, it has been shown that neural activity affects the expression of some genes. Neural activity more often enhances than represses gene expression. Several simulated genes which are subject to this form of regulation are expressed in retinal and tectal cells in our simulation. One such gene, expressed in RGCs, is *timeAverageActivity*. The concentration of the *timeAverageActivity* protein represents the recent activation level of the cell. This simulated gene is postulated, and is required in order to implement the learning algorithm of Elliott and Shadbolt. However, it's regulation is straightforward, and falls within the range of activity dependent gene regulation as seen in nature: When the cell is firing, an amount $\varepsilon$ (equal to the learning rate, see equation 5.1) of this gene is expressed. When the cell is not firing, the gene is not expressed. The decay rate of the timeAverageActivity protein is equal to $-\ln(1-\varepsilon)$, so that a fraction $1-\varepsilon$ of the existing protein concentration is retained at each time step. This means that the concentration of this protein at all times reflects the value of the term $\bar{a}_i^n$ as defined in appendix H, equation H.13.

The relationship between the expression rate and decay rate of this protein may seem contrived. This relationship emerges from Elliott and Shadbolt's formulation of the learning rule. We have run experiments which showed that the decay rate could be increased 10-fold or decreased 100-fold or more relative to the value of $-\ln(1-\varepsilon)$ without disabling the learning mechanism (data not shown). This means that the expression rate and decay rate of the protein *timeAverageActivity* are free to vary independently to a significant degree, making the emergence and maintenance of such a learning system under natural selection plausible.

Retinal cell presynaptic terminals also express a gene *NTreceptor*, which is a receptor to the protein *NT* ("neurotrophic factor"). Neurotrophic signaling molecules similar to NT, as well as their receptors have been found in nature, some of which do show activity-dependent expression (Bonhoeffer, 1996). However, the precise properties of *NTreceptor* are postulated as part of this learning algorithm. The decay rate of the *NTreceptor* protein is "fast", i.e., we use a value of 1.0, meaning that a fraction $\exp(-1.0) \approx 0.367$ of the receptor remains after each time step. Like the gene above, *NTreceptor* is expressed in an activity dependent fashion: When a retinal cell $i$ fires, the gene is expressed in a given synapse connecting the cell $i$ and tectal cell $x$ at a rate given by $s_{xi}^n \bar{a}_i^n$, where $s_{xi}^n$ is the synaptic weight in time step $n$, and $\bar{a}_i^n$ is the concentration of the protein *timeAverageActivity* as defined above. When the cell $i$ is not firing, the gene is not expressed.

As shown in table 5.1 , RGCs use a non-linear neural model (see section 2.7) with a high leakage rate, meaning that the cells have no "memory" of activation received in the past. They are not refractive, and fire for one time step — this means that their firing state is a reflection of the activation they received from spontaneously firing starburst cells in the previous time step.

| | Starburst | Retinal ganglion | Tectal |
|---|---|---|---|
| Properties of neurons | | | |
| Neural model | Non-linear | Non-linear | Linear |
| Spontaneous firing prob. ($P$) | 0.025 | 0.0 | — |
| Firing threshold ($T_i$) | 4.0 | 6.0 | 1.0 |
| Firing duration ($f$) | 5 | 1 | — |
| Refractive duration ($d$) | gaussian(45,15) | 0 | — |
| Leakage rate ($r_i$) | 1.0 | 1000.0 | 100.0 |
| Neural interconnections | | | |
| Receive activation from | Starburst | Starburst | Retinal ganglion |
| Synaptic weights ($w_{ij}$) | 1.0 | 1.0 | Variable |
| Arbor diameter | $D_{ss} = 4.2$ | $D_{sr} = 4.2$ | $D_{rt} = 13.4$ |
| Number of connections per afferent cell | 18 | 19 | 166 |

Table 5.1: Neural properties of starburst, retinal ganglion, and tectal cells. Synaptic weights from retinal ganglion cells to tectal cells are subject to learning, for their initial values, see main text. Symbols refer to the neural models as described in section 2.7. The arbor diameters (see figure 5.20) are given in units of the closest spacing of cells within the cell layers.

### Tectal cells

A neurotrophic factor $NT$ is expressed by tectal cells. The expression rate is $mT_1/w$ (see appendix H, equation H.3 with $T_0 = 0$) where $m$ is the membrane potential of the tectal cell, and $w$ is the sum of the synaptic weights of all synapses connecting to the tectal cell. The parameter $T_1$ has the values of 0.02. The decay rate of the $NT$ protein is also "fast", i.e., the decay rate is the same as that of *NTreceptor* above.

As shown in table 5.1, tectal cells use the linear neural model. The tectal cells do not have a threshold and do not fire. Instead the membrane potential represents the amount of neural activity received by the cell. Interestingly, there is some evidence that tectal cells are in fact not brought to fire by RGCs during the developmental stage simulated here (Kolls and Meyer, 2002). However, the details of this neural model are taken from Elliott and Shadbolt (1998a), and are not based on precise biological data. The leakage rate is set at the high value of 100, causing the membrane potential to effectively reset to zero at each time step. These cells consequently have no "memory" of previously received activation. While these parameters are chosen in adherence with the Elliott and Shadbolt learning rule, it can be seen from figure 5.16 that any normalized Hebbian-like learning rule should be able to capture the refinement process.

Synaptic weights from RGCs to tectal cells are initialized to values according to the expression (Elliott and Shadbolt, 1999, p. 7953)

$$w_{rt} = \beta\left(1 - \frac{2}{D_{rt}}d\right) + (1-\beta)\,r \tag{5.2}$$

where $d$ is the distance between the tectal cell and the projection of the retinal cell onto the tectum, $D_{rt}$ defines the size of the arbor (see figure 5.20 and table 5.1, the factor of 2 converting this diameter into the radius), $r$ is a stochastic value with a uniform distribution in the range $[0, 1]$ chosen individually for each synapse, and $\beta$ defines the level of noise in the initial weights, we used $\beta = 0.7$. This formulation of initial weights means that synaptic connections will be slightly stronger toward the centre of the arbor than at the edges, a bias which the refinement process will enhance until the central weight is many orders of magnitude higher than peripheral connections.

### Starburst cells

Starburst cells as we simulate them are genetically simple, in that they only express a single simulated "cell marker" gene. However, their neural model is interesting: They use a non-linear model (see table 5.1) with a non-zero probability of spontaneous firing, i.e., they may fire even though their membrane potential is below threshold. This phenomenon is observed in retinas in nature (Wong, 1999). Once they fire, they continue to do so for five simulation time steps, after which they go into a refractive period. The duration of the refractive period varies stochastically among the cells, and this duration is periodically re-randomized for each cell, as suggested in Elliott and Shadbolt (1999). The refractive periods are taken from a normal (Gaussian) distribution with mean of 45 and standard deviation of 15 simulation time steps.

### Starburst cells generate coherent waves of neural activity

Starburst cells drive spontaneous waves of neural activity in the retina (Wong, 1999). This occurs because starburst cells fire spontaneously, and because they are connected to other starburst cells within a given radius (see $D_{ss}$ in figure 5.20). When several interconnected starburst cells happen to fire around the same time, they bring neighbouring starburst cells above their firing threshold, causing a self-reinforcing chain reaction which is manifested as a traveling wave of activation. The wave is terminated when it reaches areas where most starburst cells are refractive and therefore unable to fire. After the wave has died out, the probability of a new wave being initiated will increase with time as more and more cells cease to be refractive.

The activity within the waves are transmitted to the RGCs, which in turn supply the neural activity which drives the refinement process in the tectum. The population of RGC cells becomes a "readout layer" (Feller et al., 1997) for the starburst cells, reflecting the current activation levels of the starburst cells within a given diameter $D_{sr}$. Importantly, an RGC is brought to fire even though less than all the starburst cells connected to that RGC fire. This means that while some starburst cells within a propagating wave may happen to be refractive and therefore do not fire, *all* RGCs within the wave fire. There will therefore not be "holes" of inactive RGCs within the activity wave.

As shown in figure 5.20, the connections between starburst cells and RGCs span the edges of the cell layers, topologically forming a torus. This ensures that the wave propagation patterns are uniform across the entire retina, both in frequency of waves and in the direction of propagation. If these connections where absent, waves would propagate predominantly toward the edges of the retina, which would disrupt the refinement process in the tectum.

Figure 5.21 shows the activity pattern as produced by this method. The resulting neural activity in the RGCs (shown in green) is highly coherent (i.e. adjacent cells tend to fire together, without any "holes" of inactive RGCs), even more so than the neural activity pattern of starburst cells (shown in red). This means that the activity pattern contains information about cell adjacency relationships. The learning mechanism uses this adjacency information to drive the refinement process. The lack of correlation between waves in either eye drives the development of ocular dominance.

### 5.3.4 Results with endogenously generated neural activity

Figure 5.22 on page 197 shows the results of a refinement experiment using endogenously generated waves of neural activity. At the outset (top row, page 197), each RGC is connected

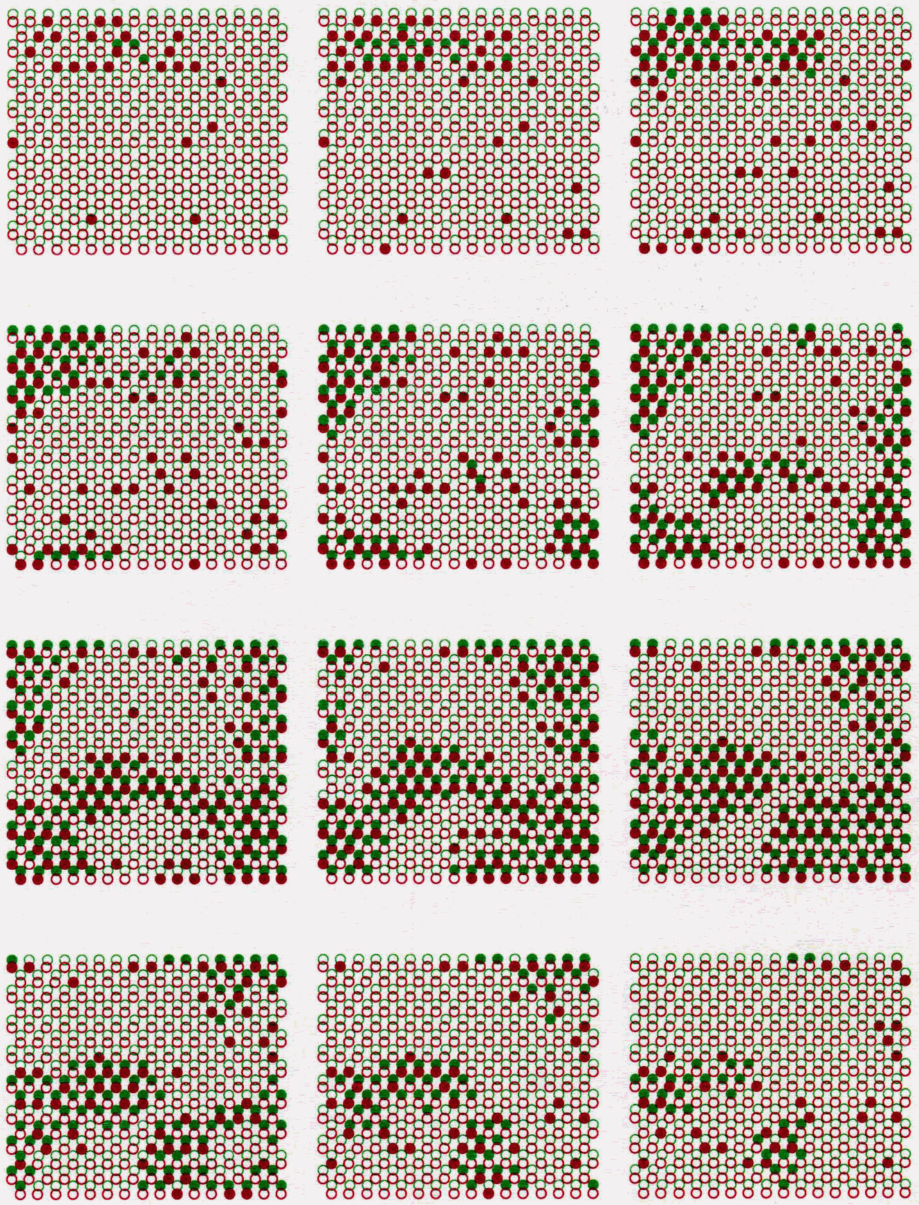


Figure 5.21: Patterns of endogenously generated neural activity in the retina, as simulated in NeuroGene. There is one time step between adjacent images, which should be read in comic-book order. Starburst cells are shown in red, and RGCs in green. Filled circles indicate cells firing. Note that while the waves may have "holes" of inactive cells among the starburst cells, there are no such "holes" in the activity pattern of RGCs.

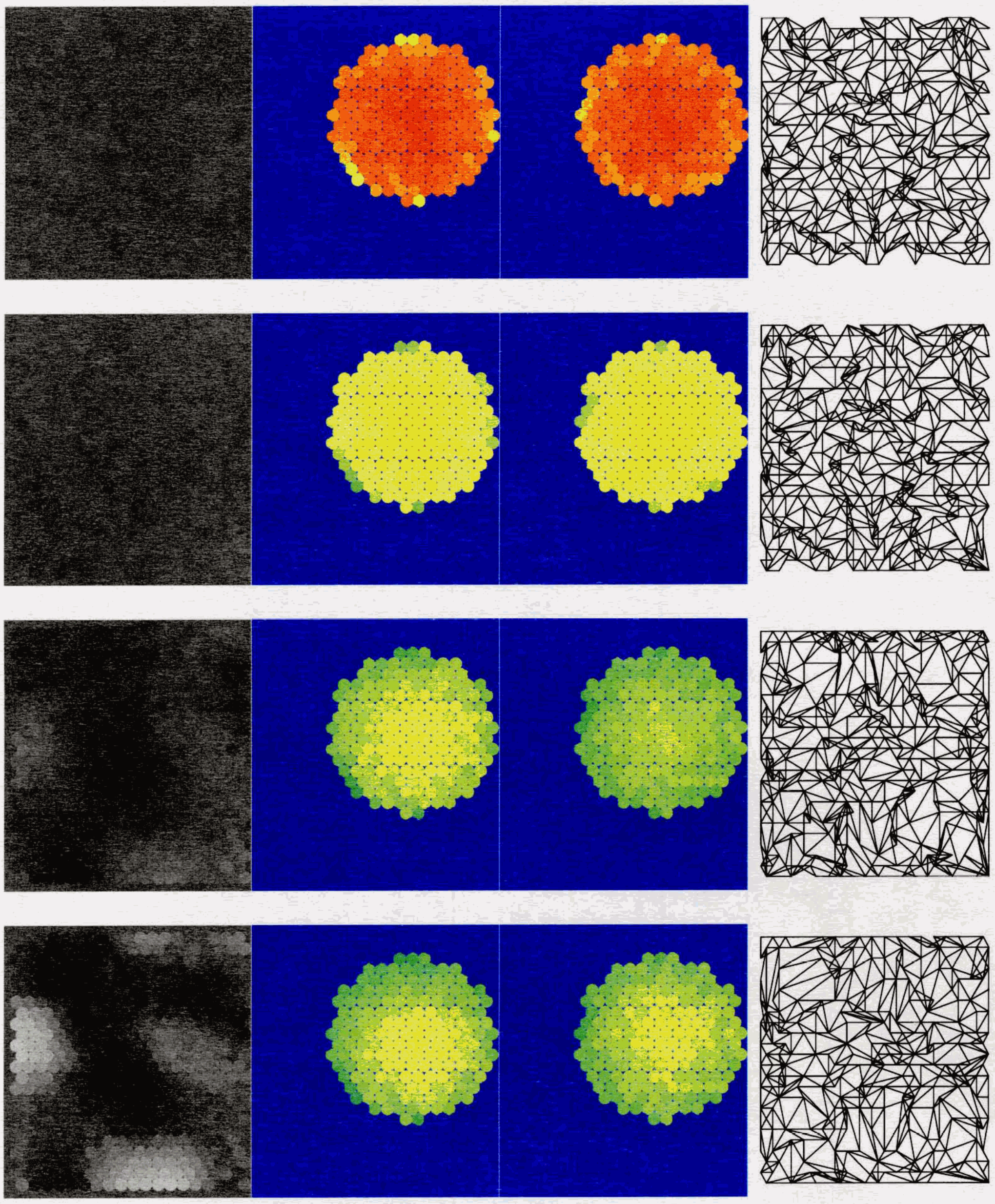


*figure continues...*

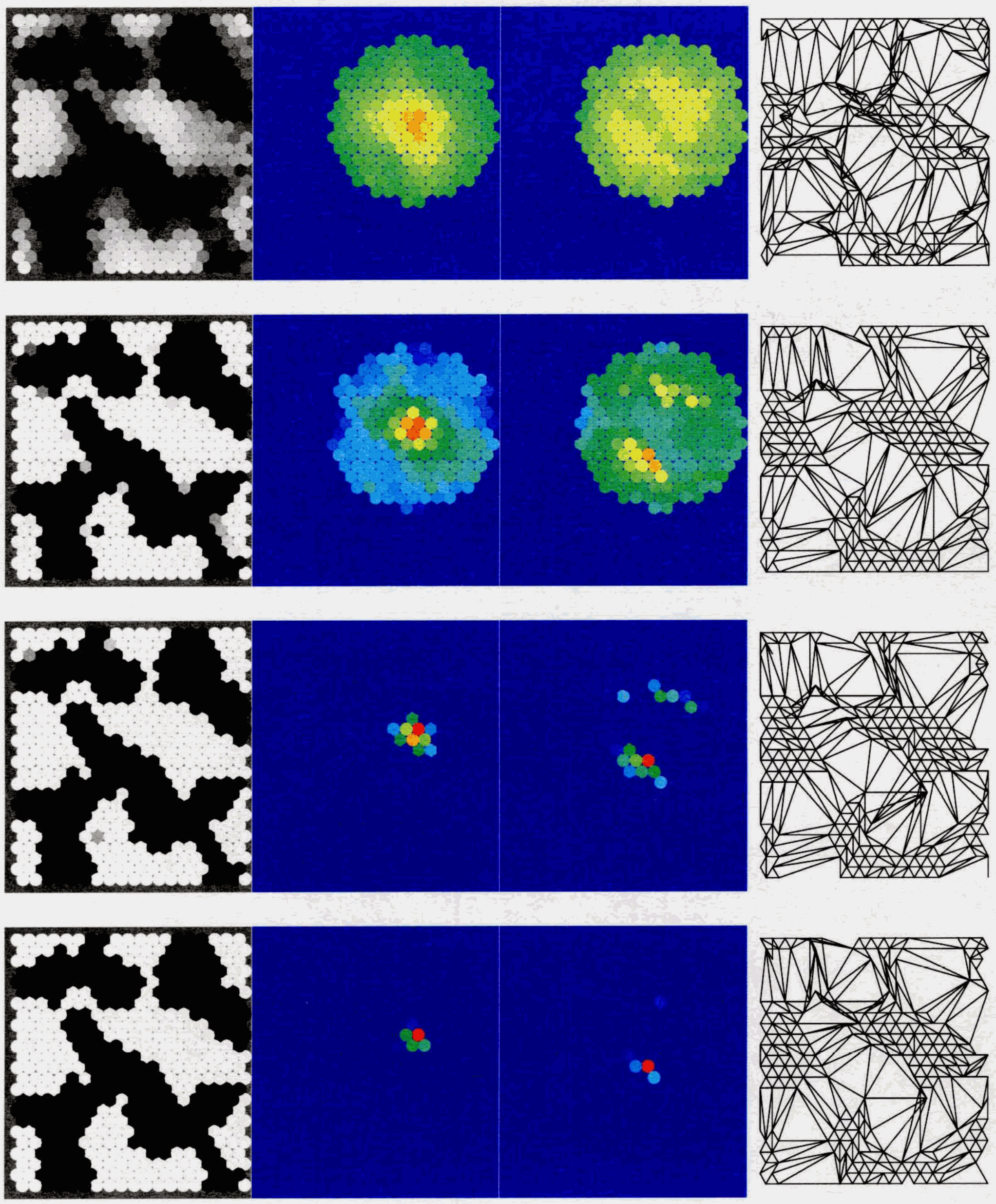


Figure 5.22: Refinement and ocular dominance with endogenously generated neural activity. Left hand column shows ocular dominance. Middle two columns show connections strengths from one RGC in eye0 (column 2) and eye1 (column3). Red represents a connection strength of 1 or greater, blue a connection strength of $10^{-12}$ or less. Right hand column shows topography of eye0, the more regular the pattern, the better the topographic projection from the eye to the tectum. Rows show situations at (previous page) 300, 600, 1,200, 2,000, 4,000, (this page) 8,000, 16,000 and 34,000 iterations respectively.

to a large number of tectal cells — this is shown in the middle two columns in figure 5.22. Each image shows the connections from a single RGC in one eye to the tectum, each circle represents the weight of the connection from that RGC to a particular tectal cell. As the simulation progresses, connections are selectively weakened until only a small number of connections from each RGC remains. Note that the choice of the "no cell death" form of the learning rule (see section 5.3.2) means that no RGCs will loose all its connection to the tectum. An RGC in one eye which is situated near the centre of an ODC dominated by the other eye will retain connections to the closest tectal cells dominated by its own eye, as can be seen in the right hand eye in figure 5.22. From this effect it follows that the widths of ocular dominance domains has an upper bound determined by the initial arbor size of connections from RGCs to the tectum.

The left hand image in each row shows the ocular dominance. Each circle represents a single tectal cell. A white circle indicates that the activation coming into that cell comes predominantly from the left eye (as determined from synaptic weights), a black circle indicates that the tectal cell is dominated by the right eye. A gray circle indicates that about equal amounts of neural activity is received from each eye. Initially all tectal cells receive similar input from either eye. As the simulation progresses, each tectal cell becomes dominated by one or the other eye. Neighbouring cells tend to be dominated by the same eye, giving rise to domains of equal dominance known as ocular dominance columns (ODCs).

The right hand image in each row shows the topographic ordering or the projection from the left eye. This visualization approach is taken from Elliott and Shadbolt (1999), and it is constructed as follows: For each RGC $x$ there exists a tectal cell $y$ such that the connection strength from $x$ to $y$ is higher than the connection strength from $x$ to any other tectal cell. The location of the RGC $x$ within the 2D layer of RGCs is indicated by $P^x_{RGC}$, similarly the location of the tectal cell $y$ is indicated by $P^y_{tect}$. Consider two RGCs $x$ and $x'$. If the distance from $P^x_{RGC}$ to $P^{x'}_{RGC}$ is less than some threshold, then a line is drawn connecting the points $P^y_{tect}$ and $P^{y'}_{tect}$ in the image. This is repeated for all possible RGC pairs $x$ and $x'$.

In an ideal topographic projection, $P^x_{RGC} = P^y_{tect}$ for all $x$ (since these are 2D locations within the plane of each cellular layer). This means that for an ideal projection, the resulting image would show a regular net of lines. All deviations from ideality will be represented by deviations of $P^y_{tect}$ from the value of the corresponding $P^x_{RGC}$, manifested in the image by irregularities in the net of lines. As the initial weights from RGCs to tectal cells are chosen

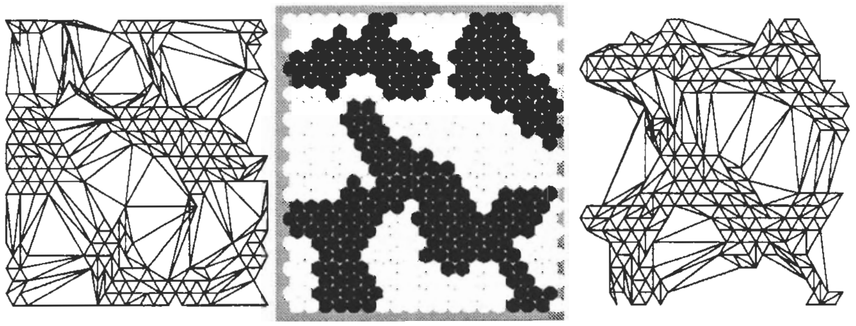


Figure 5.23: ODCs and topographies of the two eyes.

randomly, the initial topography is highly disordered. As the simulation progresses, areas of higher order emerge. This happens in lock step with the emergence of ocular dominance columns. As refinement improves, the topographic projection from the left eye is highly ordered in those areas where the left eye dominates, while disorder remains in areas where the right eye dominates. Figure 5.23 shows the topographic order of both eyes together with the pattern of ocular dominance. It shows how in areas where one eye dominates, that eye's topographic order is high, and the other eye's order is low. This is similar to results obtained in earlier simulations (Elliott and Shadbolt, 1999).

### 5.3.5 Results with simulated visual data

Ocular dominance columns and receptive field refinement occurs in some species after eye-opening, when neural activity in the retina is caused by visual stimuli from the outside world. In order to simulate such processes, a system for generating exogenous neural activity patterns has been implemented as part of NeuroGene. This system is described in detail in section 2.7.5, and the different sources and filters are described in appendix D.

An outline of the filter modules used to generate visual stimuli for these simulation is shown in figure 5.24. The binary value source generates random activity (1.0 or 0.0 with 50/50 probability) on which the rest of the filters will work. The data separates into two pipe lines, one for each eye. In the left eye pipe line, a recurrent filter introduces inter-time step coherence in the input data. The Gaussian filters smooth out the activity to create input that are plausibly similar to what might reach the brain from the visual sensory system.

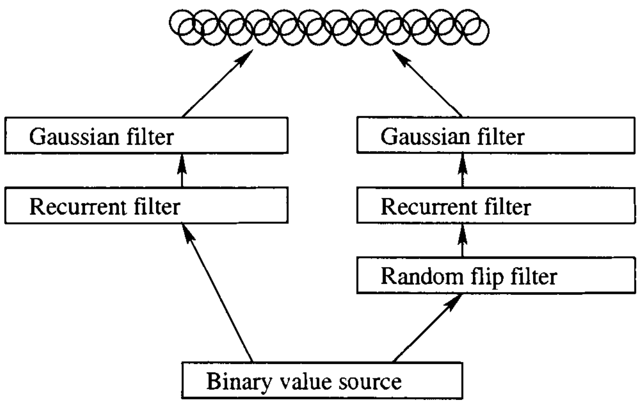


Figure 5.24: Exogenous neural activity modules assembled to give different but related visual input to the two eyes. Some helper filters have been omitted from this picture for clarity, see appendix D for a complete description.

The right-eye pipe line contains a random flip filter which alters the incoming data by a set amount (flips $0.0 \rightarrow 1.0$ and $1.0 \rightarrow 0.0$ with a given probability $p$), causing the information reaching each eye to differ by a well-defined amount. The cell layer shown at the top consists of two distinct cell populations, representing the two eyes, receiving input from the left and right pipelines, respectively. This method for generating simulated visual input is based on Elliott and Shadbolt (1999) and Feller et al. (1997).

The advantage of this approach over the endogenous neural activity scheme outlined earlier is that the inter-eye correlation $p$ can be modified in order to investigate any effects on refinement and ocular dominance. Such simulations can then be compared with experimental data produced by either reducing or increasing inter-eye correlation in animals. The inter-eye correlation may be increased by rearing animals under strobe-light (Brickley et al., 1998), while the correlation is decreased typically by inducing strabismus ("lazy eye") by cutting the muscles controlling the eye, or else using contact lenses.

Figure 5.25 shows the same visualizations as those in figure 5.22, except in this experiment, the refinement process was driven by simulated visual input. The experiment was done with a smaller number of cells than the endogenous activity experiment. This was necessary since the rate of convergence was much slower than in the experiments driven by endogenous neural activity, on the order of 200,000 time steps rather than 35,000. Both

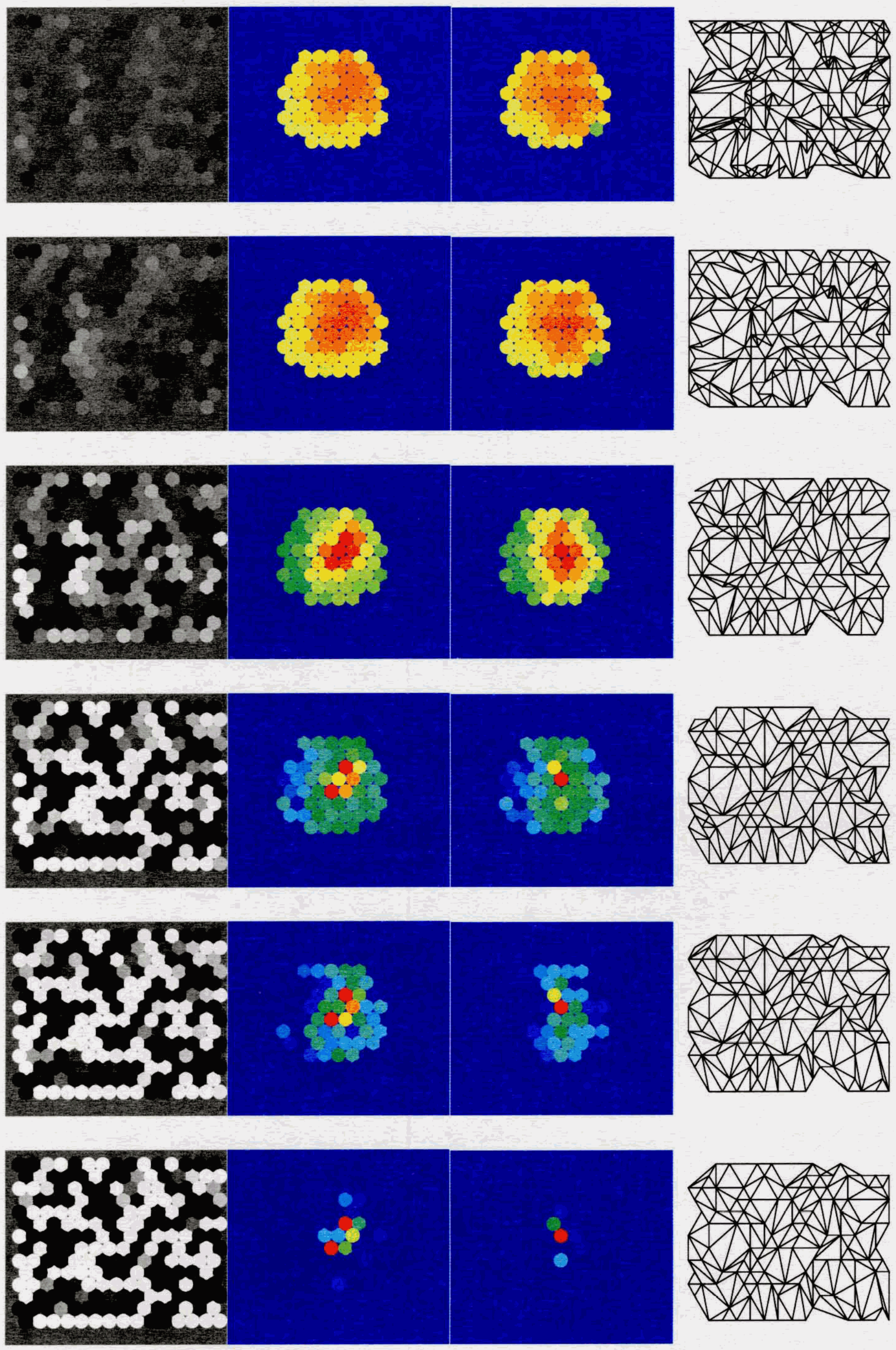


Figure 5.25: Refinement, ocular dominance and topography as simulated using exogenous simulated visual input. Rows show situations at 25,000, 50,000, 100,000, 149,000, 173,000 and 218,000 iterations.

sets of experiments show the gradual emergence of ocular dominance columns, as well as the progressive refinement of tectal cell receptive fields, the latter with a somewhat slower time course. The ocular dominance columns in the exogenous activity experiment are significantly narrower than in the endogenous activity experiment above. A number of different factors may have lead to this deviation. Together with the slower rate of convergence, the narrower ODCs represent the only significant differences between the two experiments. The reasons for these differences are not clear, but contributing factors may include:

- Inter-eye correlation, which is higher in the exogenous activity experiments. With endogenous activity, there is no inter-eye correlation. As will be shown below, inter-eye correlation greatly affects ODC widths.
- Overall level of neural activity, which may differ between the two kinds of experiments.
- Temporal coherence, which is higher in the endogenous activity experiment. The analysis of the Elliott and Shadbolt learning rule with respect to temporal coherence is complex. The authors do not investigate this in their publications, and I will not attempt to do so here. However, temporal coherence does have an effect, to the point that some temporal coherence was introduced in the simulated visual activity, see appendix D, figure D.1.

In general not too much should be made of the rate of convergence, or indeed the speed of any developmental process we simulate, given that we have no reliable way of relating the duration of a simulation time step to the developmental time of an organism.

### Effect of inter-eye correlation

Under endogenous generated activity as described above, the neural activity in the two retina is uncorrelated, i.e., the correlation is 0.5, if 1.0 indicates perfect correlation and 0.0 perfect anti-correlation. After eye-opening both eyes receive similar though not identical visual stimuli, giving an inter-eye correlation somewhere in the range 0.5–1.0. Experiments have been carried out in which the inter-eye correlation has been either increased or decreased through the use of contact lenses or surgery inducing strabismus to decrease inter-eye correlation (Lövel, 1994), or else rearing the animals under strobe-light which increases correlation

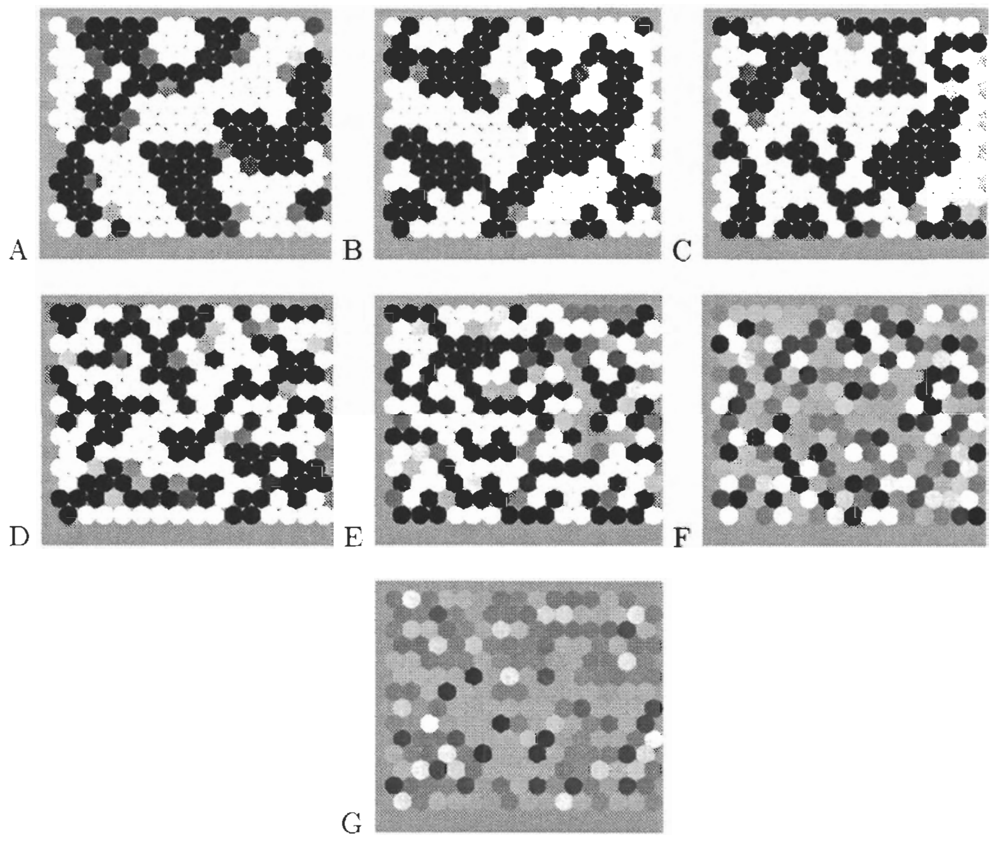


Figure 5.26: Occular dominance stripes generated with different inter-eye correlation of neural activity. A: 0.05, B: 0.25, C: 0.4, D: 0.5, E:0.6, F:0.75, G:0.95.

(Brickley et al., 1998). Such experiments have shown a marked influence of the inter-eye correlation on the ocular dominance columns. Decreasing inter-eye correlation by inducing strabismus causes sharper segregation of neural input into stripes, and increase stripe widths (Shatz et al. 1977; Lövel 1994, reviewed in Swindale, 1996, pp. 170–171).

Using the exogenous neural activity modules, the two retina can be exposed to simulated visual input with arbitrary inter-eye correlation by altering the flip-probability of the "Random flip filter" as shown in figure 5.24 from 0.0 (giving perfect correlation) to 1.0 (perfect anti-correlation). Figure 5.26 shows the ocular dominance columns are computed by NeuroGene at a number of different flip-probabilities. Anti-correlation causes very wide ocular dominance stripes to form. As the inter-eye correlation is increased, the stripes become gradually narrower until they finally disappear. These results match the experimental data summarized above, with increasing inter-eye correlation giving rise to narrower ocular dominance stripes (Lövel, 1994; Goodhill and Löwel, 1996).

### 5.3.6 Discussion

The simulated genes employed in the simulation of the activity dependent development may again be classified as to their justifications. The three genes *NT*, *NTreceptor* and *NTcomplex* together model a receptor-ligand system, with the two first representing the ligand and the receptor. A very large number of such systems are known to exist in nature, indeed such interactions (with soluble or membrane-bound ligands) form the basis for almost all inter-cellular communication in all species. *NTcomplex* represents the complex of ligand and receptor. Such genes do not exist in nature, however it is required by the NeuroGene simulation of ligand-receptor binding. Note that this gene is never expressed, as the corresponding protein is only formed by the association of molecules of *NT* and *NTreceptor.* We model the ligand-receptor binding using an equilibrium constant ($K_d = 10^{-12}M$) which is typical for strongly binding ligand-receptor systems in nature (Neet and Campenot, 2001).

The expression of the simulated gene *NT* is dependent on synaptic weights and the membrane potential, while *NTreceptor* is regulated by neural activity and regulatory proteins as well as synaptic weights. The expression of the simulated gene *timeAverageActivity* is similarly affected by neural activity and regulatory proteins. It is well established that neural activity can affect gene expression (see section 2.7.1). The influence of synaptic weight on gene expression is more speculative. However, such influence is implied by a wealth of connectionist models of learning, so it seems reasonable to make the same assumption in this thesis.

The three simulated cell identity genes *starburstCellMarker*, *tectalCellMarker* and *retinalCellMarker* are justified as before by the fact that many cell types are identifiable (if not actually defined) by unique patterns of gene expression. Actions which occur continuously within cells of a particular type is encoded in an effects section of the appropriate cell type gene, see *starburstCellMarker.*

Lastly, the role of the simulated gene *learn* is to execute actions within cell components, specifically to alter the weights of plastic synapses according to the learning rule. This action is placed in a separate gene in order to overcome the limitation of the NeuroGene gene model as currently implemented. This simulated gene consequently does not map to any gene in nature — rather it maps to molecular interactions which cause synaptic weights to be altered in response to neural activity.

The developmental processes which cause receptive field refinement and ocular dominance column formation have been simulated before, see Swindale (1996) for a review. Simulations are reported here in order to demonstrate the capabilities of NeuroGene to simulate learning processes. We have shown that the simulated genetic system of NeuroGene is able to control a learning process, and that chemical signaling, as simulated by NeuroGene, can convey the cross-synaptic retrograde information flow needed to implement a Hebbian learning rule.

## 5.4 Conclusion

In this chapter we have reported three different simulations we have carried out using the NeuroGene developmental simulator. These show the ability of NeuroGene to simulate developmental processes involving patterning, axon guidance and activity dependent development. The first, taken from the early stages of the development of fruit-fly, shows how NeuroGene can be used to simulate the development of patterns of differential gene expression, and consequently of differential cellular function and structure, from a starting point consisting of identical cells and broad, simple protein concentration gradients.

The second experiment shows the application of NeuroGene to the study of a complex axon guidance process, leading to the initial ordered connection from the eye to the midbrain in vertebrates. This simulation relies on an explicit model of the function of axonal growth cones, including their ability to sense proteins and other cells in their environment, and the competitive process underlying their navigational abilities.

In the third experiment, we have used NeuroGene to simulate an activity driven developmental process. This simulation relies on models of neural activity implemented as part of NeuroGene. The learning rule is implemented using the same genetic framework which controls all other developmental processes. The learning rule we use is based on biological mechanisms, including ligand-receptor binding, which is also implemented in NeuroGene.

Recent biological research shows that patterning, axon guidance and activity dependent development all play important roles in the formation of the nervous system (Marcus, 2004). Furthermore, many of the fundamental mechanisms by which the nervous system is formed are similar to those known to be involved in the formation of the rest of the body, including gene regulation and chemical signaling (*ibid.*). While the existence of additional developmental mechanisms unique to the nervous system can not be ruled out, evidence is lacking (*ibid.*). In this thesis I have demonstrated the capabilities of NeuroGene with respect to the simulation of patterning, axon guidance and activity dependent development. The simulator has been designed to be flexible, making it a tool which can be used to study these types of processes, concurrently, and under a wide range of different circumstances. I believe NeuroGene has the potential to become an powerful tool in the investigation of such developmental mechanisms, potentially including more complex mechanisms which presumably underlie the formation of more complex neural circuitry.

There are several processes of neural development and structures which are not involved in these simulations. Some, such as cell migration, are supported by NeuroGene, while others are not. Importantly, the support for the construction of elaborate dendritic structures similar to those seen in nature is currently lacking. Furthermore, the extracellular matrix (ECM, Browder et al., 1991), a rigid network of protein and complex carbohydrates which surround the neurons and give them structural support, can not be represented by NeuroGene at this stage. The ECM is thought to play a role in the guidance of axons. The inability of NeuroGene to represent or model physical contact between cells also makes it unable to model certain processes, including fasciculation (axon bundling) and the role of glial cells (Kandel et al., 2000) in giving structural support to neurons. However, as has been demonstrated by the three example simulations presented here, the capabilities of NeuroGene are sufficient for carrying out simulation directly incorporating biological information, and also for developing a novel model of axon guidance in topographic map formation.

# Chapter 6

# Conclusions and future work

In this chapter I summarize the work that is presented in this thesis, and put it in into the context of future research challenges in bringing together cognitive science and developmental biology. Our approach to simulating biological phenomena is summarized in section 6.1, while the software engineering aspects of the implementation are outlined in section 6.2. Inherent limitations to the NeuroGene simulation system are described in section 6.3. Our plans for future work on this project are described in section 6.4, before concluding remarks are given in section 6.5.

## 6.1 Simulating biology

A central challenge in the design and implementation of NeuroGene has been to capture a wide array of biological phenomena and mechanisms in the form of data structures and algorithms. Central issues pertain to how to ensure biological plausibility of NeuroGene simulations, and how to chose the appropriate level of abstraction in representing biological processes within the simulator.

### 6.1.1 Biological plausibility

A core goal of NeuroGene has been to be able to carry out simulations which may or may not be based on known biologically phenomena, but which would be biologically plausible. This raises the question of whether or not the NeuroGene should be designed to restrict the simulations to what is biologically plausible. After consideration we have come down firmly against such a design, for several reasons:

- We do not have a complete description of what is and what is not biologically plausible.
- NeuroGene should be a tool for exploring developmental concepts which might not be fully plausible, for instance where certain aspects of a developmental process are poorly understood.
- It is much harder to make a system which is limited to only biologically plausible simulations, since this involves creating limitations to capabilities which are not warranted from a software engineering point of view.
- Such a system is not just harder to implement, but it is also much harder to use, since barriers will be placed in the way of the user, many of which may be counterintuitive.

None the less, there are many areas where biologically implausible situations can be ruled out in a way that is conceptually and implementationally simple. An example is the encoding of protein concentrations: Negative protein concentrations are physically impossible. NeuroGene has therefore been designed so that concentrations are always greater than or equal to zero. Very high concentrations are also biologically implausible and (if they are high enough) physically impossible. However, since NeuroGene is agnostic with respect to the units of measurement used, there is no natural limit we could use as an upper limit to concentrations values. The only upper limit on the protein concentrations within NeuroGene is therefore the capacity of the **float** Java-type, approximately equal to $3.4 \times 10^{38}$.

### 6.1.2 Levels of abstraction

When designing NeuroGene, we were often faced with design questions regarding which level of abstraction to choose in implementing various features. These are difficult questions: If too much detail is included it becomes unnecessarily complicated to implement simulations, or too computationally expensive to run them. If too much is left out, we lose the biological accuracy which is the motivation behind making NeuroGene in the first place.

The NeuroGene data structures often acted as guidelines in choosing the level of abstraction. The data structure representing the NeuroGene cellular assemblies is necessarily limited in the level of detail with which it can represent the real world. For example, while protein concentrations can in principle be measured at any point within the 3D space of a real developing organism, NeuroGene can only represent those concentrations at a finite number of points in space, and values at intervening points must be approximations.

While abstractions are required in implementing the NeuroGene simulation, we have not attempted to achieve what might be described as a uniform level of abstraction. Firstly, it is not immediately clear how two abstract representations of two different processes might be compared to one another in terms of relative levels of abstraction. Secondly, in a number of cases we found that different levels of abstraction of the same underlying biological process might be appropriate in different contexts. Several such cases are described below. In these cases, the more abstract level is the "default", but there are mechanisms is place for simulating the details of the underlying biological process.

Careful examination of the many different biological processes simulated within NeuroGene shows that a mixing of different levels of abstraction is indeed a necessity. As counter-examples, if only low-level abstractions were permitted, then poorly understood mechanisms such as growth cone navigation or arborization could not be included in the system. Conversely, if a higher level of abstraction was chosen, well understood mechanisms such as ligand-receptor binding would be excluded. In both cases, the goal of creating a general tool for the study of all aspects of neurodevelopment would be impossible to realize.

As already mentioned, we have also wanted to make it possible to simulate processes for which a detailed understanding of the biological mechanism is lacking. In such cases, an abstract representation of the process is required to make up for the lack of detailed knowledge. The simulation of growth cones is an example of such a case. When it comes to other mechanisms, we do have a much better understanding, and we are able to model the process in more detail, such as in the case of ligand-receptor binding. However, also in these cases may a more abstract approach have benefits, particularly if the computational cost of the simulation may be significantly reduced.

In the following, a several aspects of the NeuroGene simulation system are discussed in terms of the chosen level of abstraction, including how simulations operating at different levels of abstraction may be implemented within NeuroGene.

### Gene expression

In designing the NeuroGene gene model, we were faced with the decision of either using an abstract representation of gene expression where genes are either "on" or "off", or else to use non-negative real values to represent the expression rate. See Smolen et al. (2000b) for a discussion of these two approaches to simulating gene expression. We found that in order to simulate the formation of protein concentration gradients, a binary "on"/"off" model of gene expression is not sufficient. Since such gradients play such an important role in developmental processes, including the topographic map formation simulated here, real-valued expression rates were required. However, due to the non-linear expression behaviour of many genes in nature, an encoding of simulated gene expression behaviors was chosen which can represent both linear and non-linear expression behaviours.

### Protein synthesis

In nature, protein synthesis is a process which consists of several stages (see figure 1.1): Transcription of DNA into mRNA, transport of mRNA to ribosomes, translation of mRNA into protein, possibly post-translational modification of the protein, and finally transport of the protein to its destination. In NeuroGene, these stages need not be simulated in detail — a simulated gene is expressed to give the protein in the appropriate location in one step. However, in nature, gene expression may be regulated at all stages of the gene expression process. In order to simulate such regulation in NeuroGene, the intermediary products, as well as the processes which interconvert them, must be explicitly represented. This can by done by representing each stage in the process as a separate simulated gene. The product produced by the first gene would be the mRNA. The second gene would require the presence of mRNA in order to produce the nascent protein, while a third gene, representing the post-translational modification, would require the presence of the nascent protein to produce the mature protein. Additional genes may be used to implement transport of the various intermediary gene products if needed. A simple example of this approach was seen in the case of the Hunchback gene in the fruit fly segmentation experiment, in which the mRNA and the protein are both modeled separately, see section 5.1.

### Soluble *versus* membrane bound proteins

We have chosen to distinguish between soluble and membrane bound proteins. These two kinds of proteins have different behaviours and play different roles in developmental processes, which warrants representing them differently. In NeuroGene, as in nature, soluble proteins diffuse freely through the extracellular space, while membrane bound proteins remain bound to the surfaces of cell components. Diffusion within the cell membrane is ignored in NeuroGene — only through simulated active transport can membrane bound proteins move between adjacent cell components within the same cell.

Within the interior of cell components, no distinction is made between the two kinds of proteins. Intracellular membrane bound proteins are imagined bound to the numerous intracellular membranes, such as the ER, golgi, etc., however, these structures are not represented within NeuroGene.

### Intracellular, surface-bound and extracellular proteins

We have also chosen to distinguish between the intracellular and surface concentrations of proteins. This allows us to simulate a range of phenomena which are important in chemical signaling, such as endocytosis and exocytosis, intracellular signaling cascades, etc. While many simulations would in principle be possible without representing the intracellular protein concentrations, using membrane bound proteins only to encode the "state" of each cell component, such simulations would in many cases be essentially different from the biological reality they represent. For instance, in cases where mRNA molecules are explicitly modeled, it would be inaccurate and counter-intuitive to represent mRNA as membrane bound and exposed on the cell surface. This would force the user to mentally convert between the biological situation which is to be simulated and NeuroGene's representation of that situation. This contradicts our overarching goal of biological accuracy.

### Communication from cell membrane to genome

In nature, passing of information from receptors on the surface to the genome in the cell nucleus is usually a complex process involving numerous intermediate signaling molecules. NeuroGene allows genes to directly query the concentrations at the cell surface. However, using genes which express intracellularly, the mechanisms of intracellular signaling pathways may be simulated.

### Molecular mechanism of protein targeting

In nature, when a given protein is found primarily in certain parts of cells (e.g. in synapses only), this is the result of intracellular transport of either mRNA or protein from the cell body to the synapses (Lodish et al., 2000, pp. 809–817). In the NeuroGene gene model, genes are potentially expressed in any part of the cell. Genes may be written so as to be expressed only in certain types of cell components, such as synapses or dendrites. This allows the user to implement protein targeting without the complex details of the intracellular transport. However, intracellular transport mechanisms are in place which allows such processes to be simulated in detail when that is needed.

### Molecular mechanism of morphology and cellular behaviour

All morphological change brought about under genetic control are in nature mediated by chemical changes to the cellular cytoskeleton or other intracellular structures (see section 2.2). For example, changes in cell shape may be brought about by causing the cellular skeletal structure to disintegrate in some parts of the cell and build up in other parts (Browder et al., 1991, chapter 9). These processes fall outside the scope of our simulator to simulate explicitly, since the cellular skeletal structures are not represented within NeuroGene. We have in stead devised primitive operations which make it simple to specify processes like cell division, axon growth etc.

## 6.2 Implementing NeuroGene

The implementation of NeuroGene has involved a number of software engineering challenges. In the area of computer graphics, mechanisms of scientific visualization have been implemented. Other modules have been implemented to visualize synaptic weights in various ways. The large majority of images representing simulations in this thesis have been produced directly from NeuroGene without any post-processing. Beyond the cellular views, a moderately complex graphical user interface has been constructed. The user interface was redesigned with valuable input from Eric Dewitt.

The parser used to process input scripts uses a stack-based symbol table to implement lexical scoping, as well as a dynamic library function table for lookup of language primitives. For optimal speed and responsiveness, NeuroGene is implemented using multi-threading, where the user interface, the simulation itself, and other secondary processes such as file I/O, each run in a different execution context. While each thread has its own program counter, all share the same memory space, which requires locking and other forms of concurrency control.

Implementing the physical aspects of the simulation requires diffusion, decay and ligand-receptor binding to be expressed within the context of the NeuroGene data structure and execution model. The 3D interpolation and extrapolation used to compute extracellular protein concentrations is also used. Verification of the correctness of the implementation required the symbolic analysis of these processes in order to express them in a form which is directly comparable to the numerical values produced by NeuroGene.

### 6.2.1 Designing the gene language

We decided that the best way to present genetic information to the NeuroGene system would be in the form of text written in a specially designed genetic scripting language. The design of a new programming language is a complex task, fraught with subtle issues that may only become apparent when implementation is complete and the language is in use. Our target audience for our genetic language was trained biologists, not computer scientists, which made the design more challenging.

The NeuroGene language got its basic syntactic elements from C, C++ and Java-like languages. The language evolved along with the rest of the system, and new features were added to the language as new functionality was added to the simulator.

The gene language contains the *gene* construct, for which general purpose languages have no counterpart, and which is designed specifically to capture the mechanism of developmental control through gene expression. It contains two parts, the first of which, the *regulation* part, is semantically similar to a function in a language like C or Java, but with a specific return type consisting of a tuple of a real value and a value designating the desired location for the protein being produced by the gene. The regulation part may also not return any value, indicating that the gene is not expressed.

Inherent in the idea of the regulation section is that it should have no side-effects. If the regulation section of a gene causes morphological changes or other kinds of changes to cells, it would be in conflict with a fundamental aspect of the biological concept of a gene. This justifies the the added complexity of incorporating this constraint into the language. This was accomplished by making certain language primitives available only in certain grammatical contexts, see section 3.1.7.

The gene *effects* section is designed to include the instructions which will be executed as a result of a gene's expression. Semantically the effects section is similar to a function in C or Java with a *void* return type (i.e. they don't return any value), but with side effects, involving changes to the cells' shapes, neural properties, etc. Finally, the construct of a *growth cone function* resembles the gene regulation section — again it has a return type consisting of a tuple of a real number (the bid) and an action type (migrate, form synapse, etc.). Like the regulation section of genes, growth cone functions should have no side effects.

While these elements do have semantic equivalents in general purpose languages, the creation of a special purpose language allows us to use a succinct syntax that is meaningful to people with a biological background, thereby fulfilling an important design goal of NeuroGene.

### 6.2.2 Biological primitives

Making NeuroGene has involved the development of a set of primitive actions and queries which can capture the developmental processes and genetic control mechanisms underlying the simulations that we have wanted to carry out. Queries are used within the context of genes and growth cone functions to retrieve information about the state of cell components and their environment, relating to protein concentrations, properties of the neural model, etc. They have been designed to be biologically meaningful, relating to the cellular structures which are represented by the NeuroGene data structures, rather than the data structures themselves. In this sense, the primitives form an important aspect of the NeuroGene user interface, and should therefore present a view of the cellular structures that is consistent with that presented by the visual GUI elements.

The actions have been designed to be equally biologically meaningful, but also as general as possible. As such, most take parameter values of different types. In the context of a gene,

```
1 gene Foo {
2    regulation {
3       ...
4    }
5    effects {
6       setSynapticWeight(surfaceConcentrationOf(Bar));
7       setThreshold(sqrt(synapticWeight()));
8    }
9 }
```

Figure 6.1: Example of a gene effects section which reveals the unusual semantic properties introduced by the parallel execution model of NeuroGene.

the action primitives are invoked with arbitrary parameter values, which may be supplied in the form of constant values specified in the gene script, or else computed from the return values from queries. In this, the NeuroGene language aligns closely with the conventions of general purpose programming languages.

### 6.2.3 Execution model

The goal of the NeuroGene execution model is to make it appear as if the genetic code of different genes within different cell components runs in parallel even though it is running in the sequential environment of the Java Virtual Machine. This is the reverse of the goal of parallel programming, which is commonly to implement, on parallel machines, algorithms which are typically conceived as sequential.

The NeuroGene execution model, and its implementation of simulated parallelism is simple — actions are not executed until all genes in all cell components have been processed, at which point all pending actions are executed. This does alter the semantics of the NeuroGene language relative to general purpose languages, as illustrated in figure 6.1. In this gene, the parameter value passed to the **setThreshold()** function in line 7 will be based on the old value of the synaptic weight, not the value set by **setSynapticWeight()** on line 6. This is because the parameter values passed to **setThreshold()** on line 7 are evaluated before the action resulting from the **setSynapticWeight()** command on line 6 is implemented.

## 6.3 Limitations

The most important limitations to the capabilities of NeuroGene derive from discretization of time and space. There are fundamentally different approaches which avoids either or both of these problems, but these introduce other drawbacks. For instance, continuous time can be simulated if all changes occurring during the simulation can be expressed in terms of differential equations. This is straightforward for changes in protein concentration, harder for such changes as cell migration, and virtually impossible for cell division and other changes which can be regarded as discontinuous on the time-scale NeuroGene simulations. Similarly, continuous space could be implemented if the extracellular concentration gradients of all proteins were represented by analytical functions. Such an internal representation would necessarily grow with the number of sources producing the protein and the duration of the simulation. This means that the complexity of the internal representation potentially grows without bound, severely limiting the capabilities of the simulation system.

In order to be free to implement a wide range of biological primitive actions, and also be able to support complex concentration gradient shapes arising from many sources, we chose a simulation design based on discrete time and space. As a consequence, the protein gradient shapes can never become more detailed than the pitch of the WorldNode grid allows. Similarly, the neural models are currently limited to those that can operate with discrete time.

In order to make simulations as fast as possible, the basic design decision was made that collision detection computations would not be carried out except where needed by growth cones. Consequently, any developmental process which relies on the direct interaction between cell components though other means than extracellular protein signaling, must involve the growth cone mechanism. Growth cones perform collision detection, and from within growth cone functions, the properties of other cell components can be queried. This, in principle, makes NeuroGene able to simulated complex axon guidance processes where extending axons navigate their environment though interaction and recognition of surrounding cell structures. The growth cone mechanism is sufficiently general that it may also be used to guide dendrite growth and even cell migration.

## 6.4 Future work

For the future, we are hoping to continue the work on NeuroGene, applying it to problems not only in the area of neurodevelopment, but also using it as a tool to further the understanding of cognitive processes. In more specific terms, there are certain areas of the system that may be the subject of improvement.

### 6.4.1 Improving the gene model

An improved model of gene expression would take better account of the fact that it is not gene expression in itself, but rather the presence of proteins, which cause cells to undergo changes in morphology and behaviour. Such an improved gene model would not invalidate the three developmental simulations presented in this thesis: The topographic map formation and ocular dominance/refinement simulations contain some simulated genes which do not represent genes as they occur in nature (*FindTectum* and *SortMap* in the former and *learn* in the latter). In the improved gene model, these would be modeled by something other than simulated genes, which I will here call *triggers*. Genes would then only cause the production of proteins, and the presence of proteins would in turn activate triggers, causing morphological and functional changes in cell components. It follows that the *effects* sections of simulated genes would no longer exist.

The activation of triggers would be implemented in the same way as the activation of genes. The triggers would therefore have a structure similar to the *regulation* portion of simulated genes, in that they would contain conditionals and actions, the conditionals determine in each cell component at each time step whether the actions are implemented or not. Since there will be no need for gene *effects* sections, genes in the new model would be a special case of triggers, as genes are limited to actions representing protein expression. Similarly, the regulation of simulated genes would represent the interactions of regulatory proteins with DNA of genes, while the corresponding regulation of triggers would represent a much wider range of possible molecular interactions.

The implementation of triggers would be done concurrently with a redesign and simplification of the gene language. This redesign would predominantly simplify the syntax of the language, while not substantially altering the semantics beyond the changes outlined above.

### 6.4.2 Framework for building dendritic structures

The formation of dendritic arbor structures is an important element of neurodevelopment which has not been explored in this thesis. The results of dendrite outgrowth, probably guided by growth cones (Fiala et al., 1998), these complex tree-like structures form numerous synapses with axons of other neurons. It is believed that their shape and the placement of synapses across their extent significantly affects neurons' information processing properties.

Biological science is still a long way from being able to understand the genetic and activity-dependent mechanism of dendrite growth. In order to include the ability of simulate such growth within NeuroGene requires a more abstract framework, such as that of the *turtle* mechanism introduced as part of the Logo programming language (Logo Foundation, 2004). Turtles were initially used in Logo programs which control mechanical robots. Turtles exhibit the properties of a location and an orientation in 3D space, and through rotation and migration commands, turtles can be used to build highly complex structures. Turtles can be associated with cell components, and queries and actions relating to the state of the turtle can be used both within genes and growth cones. Components of this system has already been implemented, but they have not yet been exploited in any developmental simulations.

Turtles form an integral part of the genetic control system as simulated in NeuroGene. This means that the invocation of the turtle mechanism occurs under genetic control (similarly to the invocation of the growth cone mechanism, see section 2.6.1), and that values of the various parameters which determine the turtle behaviour may be composed of queries, allowing the biochemical, morphological and neural state of a cell component to affect this behaviour.

The last two simulations presented in this thesis both simulate developmental stages of the same neural projection from the retina to the tectum. With the capabilities outlined above in place, we would be able to combine both into a single simulation starting with the initial axon growth from the retina, and ending with a fully formed and refined retinotectal connection with ocular dominance columns.

### 6.4.3 Developmental modules

An important technique used in the study of the brain and mind is that of divide-and-conquer modularization. The oldest approach to such decomposition was to locate *structural modules*, such as the cortex, hippocampus, thalamus, cerebellum, etc., each of which can be assigned more or less defined functions. Through experimental and accidental lesions of different parts of the brain, a lot has been revealed about the functions of these various structural modules. This has also led to a well-established map of arealization of the cortex where many regions are known to normally process information from particular senses, such as visual and auditory cortex, etc. (Krubitzer and Huffman, 2000). However, when it comes to higher cognitive functions such as language, a similar assignment of functions to areas of cortex breaks down (Marcus, 2004, pp. 128f).

A second approach is to look for *functional modules*, for example trying to map out the visual system from the eye to the cortex, or the auditory system from the ear to the cortex. As far as higher cognitive functions such as planning and problem solving are concerned, there is little biological evidence as to whether or not functional modules exist, or where they might be located. However, convincing theoretical arguments have been put forth for the existence of such modules also in higher cognition (Hadley, 1999, 2003).

The increasing understanding of developmental biology now allows us to map out a third kind of brain modularity, that of *developmental modules*. A particular developmental module is characterized by how it is built during the development of the brain. The same module may be used in different contexts, with minor modifications in the mechanisms.

Topographic maps constitute one example of a developmental module. Topographic maps are known to exist in most of the sensory pathways in the brain, including the visual, auditory, olfactory and somatosensory — in each case the information processed by the neural circuitry of the map is different, but in each case the map serves the role of transmitting information between different systems while maintaining the spatial encoding of that information. In many cases, these maps are built using closely related developmental mechanisms, and using the same set of molecular markers for axon guidance.

Some interesting relationships may be emerging between the different modes of modularization: We see that the developmental module of topographic maps are involved in structural modules with similar roles (the projection of sense data from the periphery to the brain) within different sensory modes. If this close relationship between developmental and structural modules is the rule rather than the exception, we can anticipate that developmental biology will become an important source of new information which will shed light on difficult questions within cognitive science.

How can NeuroGene be extended so as to make it a useful tool in the discovery and investigation of developmental modules? For NeuroGene to be able to handle large developmental simulations, the simulated genetic code must be modularized in some way. For such modules to be useful and self-contained, it is important that such modularization reflect the phenomenon of developmental modules. Through developing a scheme for modularization of NeuroGene gene code, it is possible that we may arrive at a better understanding of what developmental modules are, and the mechanisms by which they interact both during development and during evolutionary processes.

## 6.5 Conclusion

The mechanisms which support perception, higher cognition and motion control in humans and other organisms is implemented in a information-processing substrate which consists of neurons. Among the different areas of artificial intelligence research, connectionism alone takes this fact explicitly into account. It is therefore within the field of connectionism that biological knowledge about neuronal systems can be most directly incorporated. This can be done for instance by constructing artificial neural networks in which neurons have properties that are derived from those measured in real neurons, and in which the topologies of neural interconnections are similar to neural micro-circuits as observed in nature. Furthermore, the way in which neural networks respond to patterns of activation, e.g., by modification of synaptic conductances (or weights) according to learning rules, can also be modeled on biological data on synaptic plasticity. The current explosion in biological data on the structure and function of neural systems make this research approach potentially very fruitful.

Increasing understanding of how genes exert control over organisms have shown that "nature" and "nurture" are not separate mechanisms competing to shape the organism. A better understanding may be that genes (nature) determine how an organism responds to its environment (nurture) (Marcus, 2004). Seen over evolutionary history, the environment has in turn shaped genes through the process of natural selection. The resulting, highly adapted genomes may include elements that anticipate the environment in which the organism exists. This further blurs the line between the two, since such genetic elements can use naturally occurring environmental factors (such as spatially and temporally coherent visual input) to drive developmental processes.

There is no reason to believe that such tight interaction of genetic information and environmental influences does not also extend to higher cognitive functions (Marcus, 2004). While the influence of experience on the cognitive abilities of an individual has been recognized for centuries, the discovery of a single genetic mutation leading to a specific and serious language impairment (Lai et al., 2001; Pinker, 2001) reveals the central role that genes may also play in supporting cognitive function.

While in this case, a particular gene has been shown to be necessary for the development of normal language skills, no-one would argue that it is sufficient. Most cognitive abilities probably form as a result of both genetic and environmental influences, neither sufficient on its own (Marcus, 2004). To fully understand the processes that shape our cognitive abilities, we therefore need to understand how these two kinds of processes interact.

Artificial neural networks represent the environment and the network's interaction with the environment through its input and output nodes. However, few have extended the analogy of the artificial neural network to also include the developmental mechanisms which create and modify the network, and the genetic information which controls and coordinates these processes in nature (for some that have, see section 1.5.5). Through such an extension, the interplay between genetic and environmental factors in shaping neural systems can be investigated. NeuroGene has been designed to combine connectionism's approach to simulating neural processes with a representation of the genetic and developmental processes which create and modify natural neural networks.

The three developmental simulations reported in this thesis show that NeuroGene is able to capture a wide range of developmental phenomena as seen in nature, including patterning, axon guidance and activity-dependent development. NeuroGene has shown itself as a valuable research tool: Using NeuroGene we have developed a new model for the activity-independent formation of topographic maps through axon guidance. This model replicates a wide range of experimentally observed phenomena, and due to the direct relation between the NeuroGene language and biological mechanisms, the model is directly formulated in terms of the biological processes. In order to demonstrate the capabilities of the system, three examples of NeuroGene simulations have been included in this thesis. These show that NeuroGene is capable of simulating three developmental mechanisms which are important in the formation of the nervous system. We do not consider these simulations to represent the limit of the capabilities of NeuroGene. While there are some inherent limitations in the design of the system, we hope to be able to use NeuroGene as a vehicle for further investigations into the interactions between genetic and environmental factors in the development of neural systems.

# Appendix A

# List of terms

The following glossary has been compiled in order to make this interdisciplinary thesis accessible to as wide an audience as possible. Within this list, all words in *italics* have their own entries in the list.

– **A** –

**abstract syntax tree** A data structure used to represent a string of text written according to a particular *grammar*. The AST is built by a *parser* as it reads the input string.

**action** In the NeuroGene genetic programming language, actions represent changes in the states of cell components brought about under genetic control. Actions represent all the types of changes in cells' *morphology* and activity which may be brought about through gene expression. Actions take the form of function calls, most of which return no value. Many actions take parameters which modify the effect of the actions. The values of the parameters can depend on the state of the cell component or its environment through the use of *queries*.

**action potential** A transient reduction in the *membrane potential* of a *neuron*, traveling as a wave along an *axon*, thereby carrying information between neurons. Action potentials are sustained by *ion channels*.

**ActionQueue** A data structure used within NeuroGene to store pending changes brought about by the execution of *actions*, but not yet applied to the simulation data structure.

The ActionQueue is required in order to ensure *parallelism* between *genes* and *cell components*, and it plays an important role in the NeuroGene *execution model.*

**activity level** An abstract representation of the membrane potential which is better suited for the implementation of models of neural activity. The activity level of a neuron is a number between 0 and 1, with 0 representing the resting potential and 1 representing the cell generating an action potential. See section 2.7.

**amino acid** Building blocks of *proteins*. There are twenty naturally occurring amino acids which are used to build all proteins in all living organisms.

**AST** See *abstract syntax tree.*

**axon** The main output device of a *neuron*. The axon consists of a long thin extension of the cell, which may reach many millimetres away from the *soma*, carrying *action potentials* to target cells.

**– C –**

**cascade** A particular kind of *gene network*, which has the topology of a *tree*. The gene at the root of the tree is the master gene — activating the master gene causes all the genes within the cascade to be activated. Through cascades, very complex developmental processes can be controlled by a single gene.

**CellComponent** The abstract Java class used to represent all cell components in NeuroGene. Derived concrete classes include *Soma*, *Neurite*, *PreSynapse* and *PostSynapse.*

**cell components** In order to represent neurons of complex shapes, smaller components of the neuron are represented as separate entities within the NeuroGene data structure. Cell components represent the central portion of neurons containing the nucleus (the soma), as well as short, straight, unbranched segments of axons and dendrites and presynaptic and postsynaptic termini.

**collision detection** The computational process of detecting physical contact between simulated physical objects moving within a 3D environment. This is in general a computationally expensive operation, since every possible pair of objects need to be considered, giving $O(n^2)$ time-complexity where $n$ is the number of potentially interacting objects. In NeuroGene, collision detection is only implemented in the context of simulated growth cones.

**complex** See *ligand-receptor complex.*

**connectionist neural network** Abstract models of neural circuits which implement various forms of *learning.* Networks consist of nodes, which represent neurons, and connections, which represent synapses between neurons. Neural activity is usually represented by nodes either firing or not firing. Learning rules define how the weight of synapses (i.e. their conductance of activity) change as function of activity of nodes.

**– D –**

**decay** The process by which proteins and other molecules are broken down and disappear. In nature, many molecules are broken down by enzymes, making the rate of decay dependent on the abundance of these enzymes. In the context of NeuroGene, such dependences are not modeled. The rate of decay of each protein is characterized by a constant first-order decay rate $k$, giving the following time dependence of the concentration $C$: $\frac{dC}{dt} = -kC$.

**dendrite** The main input devices of a neuron. These form *synapses* with the *axons* and *somas* of other cells, through which the cell receives neural activity.

**design pattern** A recurring programming problem with a recognized and widely applicable solution. This solution typically require the capabilities of an object oriented programming language such as C++, Java or others. Important patterns used in the implementation of NeuroGene include the Visitor, Observer and Command patterns, as well as the Model-View-Controller (MVC) pattern.

**development** In biology, the term development refers to the process by which a fertilized egg changes into a complete organism capable of sustaining life, access nutrients and give rise to new offspring. This is distinct from *evolution.*

**differentiation** The process by which a cell takes on (or commits to) one of several potential roles during development. This is usually an irreversible process. See also *patterning.*

**diffusion** The process by which *soluble proteins* spread through the environment, away from their source. In NeuroGene, only diffusion through the *extracellular space* is simulated. The rate of diffusion is characterized by a diffusion coefficient D, giving the time dependence of the concentration $C$: $\frac{\partial C}{\partial t} = D\frac{\partial^2 C}{\partial x^2}$. See appendix F.

**DNA** DeoxyriboNucleic Acid. The chemical compound used to encode and store the inheritable information of plants and animals. DNA exists as two paired anti-parallel strands, each of which consists of a chain of nucleotides. There are four nucleotides, represented by the letters A, T, G and C. These are pairwise complementary, with A binding to T and G to C. The nucleotides in the two strands of a DNA molecules are arranged in such an order that each nucleotide in one strand is always paired with the complementary nucleotide in the other strand. The structure, including the stored information content, can be copied intact by separating the two strands and using one or both single strands as templates for the formation of novel strands. The novel strand is constructed while maintaining nucleotide complementarity with the original strand, thereby maintaining the information content in the novel strand. This mechanism is used both when the genome as a whole is duplicated prior to cell division, and also for the production of *mRNA*.

– **E** –

**enhancer** A region of the DNA sequence of a gene which influences under which conditions the gene is expressed. See figure 1.2.

**enzyme** A particular kind of *protein*, whose role it is to facilitate (catalyze) chemical reactions. In most enzymes, reactions are facilitated when the molecule or molecules which are to react bind to the enzyme in a particular orientation, which is optimal for the reaction to occur. Many enzymes increase the rate of reaction by many orders of magnitude.

**equilibrium** The term chemical equilibrium applies to reactions which are reversible, i.e., the reactants can react to form products, and the products may also react "backwards" to form reactants. Such a reaction is at equilibrium when the forward and reverse reactions happen at the same rate, meaning that the concentrations of reactants and products do not change. All reversible reactions are characterized by an equilibrium constant, which defines the relationship between the concentrations of the various chemicals when the reaction is at equilibrium. In the context of NeuroGene, the reversible reaction of the binding of *ligand* $L$ to a *receptor* $R$ for form a *ligand-receptor complex* $C$ is modeled. Such reactions are at equilibrium when the equation

$$K_d = \frac{[C]}{[R][L]}$$

is satisfied, with $K_d$ the equilibrium concentration, and square brackets signifying "concentration of".

**evolution** The process by which biological species become and remain adapted to their environment and other species with which they interact, such as predators and prey. Evolution occurs through gradual change to the genes within a population over many generations. The driving forces behind this process is the stochastic mechanisms of mutations to individuals' genomes, and the non-stochastic process of natural selection, in which highly adapted individuals survive at the expense of less adapted ones. This is distinct from *development.*

**expression** See *gene expression.*

**extracellular space** The space surrounding the cells that make up a (real or simulated) organism.

**– F –**

**filopodium** Transient fibrous extensions from a *growth cone.* Filopodia are involved in the growth cone's ability to sense its environment and in choosing an optimal path of axon growth.

**fire** Neurons are said to fire when the generate an *action potential.*

**firing duration** In a discrete time model of neural activity, once a simulated neuron reaches the firing state, it may remain in this state for an extended period. The duration of this period is the firing duration. See section 2.7.

– **G** –

**gene** The fundamental unit of inheritable information, encoded in *DNA* as part of the *genome* of an organism. A gene consists of *regulatory elements* and a *protein coding region.*

**gene expression** The process by which the protein encoded by a particular gene is produced. The properties and function of a cell is to a large extent determined by the proteins which it contains. It is therefore important that gene expression is tightly controlled. Such control involves *regulatory proteins* during *transcription*, and additional control is exerted during *translation.*

**gene network** A set of *genes*, whose corresponding *proteins* are *regulatory proteins* affecting the *expression* of other genes within the same set.

**gene regulation** See *gene expression.*

**genetic algorithm** A search algorithm which simulates the biological process of *evolution.* It is used to search for near-optimal solutions to computationally hard problems.

**genome** The set of all genes of an organism.

**genotype** The characteristics of an organism as encoded in the organism's genome. Individuals with different genotypes may have identical *phenotypes.* A common example is that of dominant/recessive genes: A gene which has two forms (alleles), represented by A and a, with A being dominant, will give rise to two phenotypes. Keep in mind that all cells in an organism contains two copies of each gene. The phenotype reflecting the recessive allele a will only be visible in individuals with two a alleles, i.e., aa. The two other genotypes, Aa and AA, will both have the phenotype reflecting the dominant allele A.

**grammar** A formal specification of the *syntax* of a formal language, typically a programming language. Most commonly used for specifying programming languages are context-free grammars. Such grammars consists of one or more productions. On the left hand side of each production is a single non-terminal symbol, and on the right hand side is a list of one or more terminal and/or non-terminal symbols. The grammar also defines a root symbol. For a string to be a valid member of the language, the string must be reducible to the root symbol through the conversions represented by the productions in the grammar.

**growth cone** A dynamic structure an the tip of a growing *axon* or *dendrite*. Through extending numerous *filopodia*, the growth cone senses its environment and navigates, sometimes over long distances, to reach its target area. Here the growth cone is involved in the formation of highly specific synaptic connections based on chemical signaling from the target tissue. See section 2.6.

**growth cone function** A formal definition of a behaviour of growth cones simulated according to the NeuroGene growth cone algorithm, see page 224.

– **H** –

**hash table** A data structure for storing and fast retrieval of elements. Retrieval is based on hash-values computed based on the contents of each element. Ideally all elements should have different hash values. When this is the case, retrieval of an element from the table is done in constant time, i.e., the time of the operation is independent of the number of elements in the table.

**hyper polarization** A hyper-polarized *neuron* has a *membrane potential* which is more negative than the *resting potential*. Such a neuron is less likely to generate an *action potential* than a comparable neuron at the resting potential. Processes which cause neurons to become hyper-polarized inhibit neural activity.

– **I** –

**ion** Atoms or molecules which carry an electrical charge. Different ions are abundant both within cells and in the *extracellular space*. Ions carry all currents which are involved in the transmission of *action potentials*.

**ion channel** Complex *proteins* which are embedded in the cell *membrane*, and which allow *ions* to pass through the cell membrane. Ion channels are highly specific to which ions they let through (there are channels for $Na^+$, $K^+$, $Ca^{2+}$, $Cl^-$ and others). They are also specific as to when and why they open or close. Some open in response to chemical signaling (see *neurotransmitter*), other open or close in response to changes in the *membrane potential.* Finally, ion channels have different and highly specific temporal behaviours, with some staying open for an extended period while others close rapidly, etc. Together the distribution of different ion channels across the surface of a neuron determines to a large extent its electrical and information processing properties.

**– L –**

**leakage rate** The rate of decay of the *membrane potential* toward the *resting potential.* In order for a neuron to be brought above the *threshold potential*, *action potentials* coming into the cell must outweigh the leakage rate. This rate applies prior to the generation of an action potential, and should not be confused with the return to resting potential after an action potential has beacrossen generated.

**learning** The modification of synaptic conductances (or weights) resulting from neural activity. In the context of abstract *connectionist neural networks*, even simple learning rules have been shown to be able to capture complex learning behaviours. The first and best known learning rule is that of Hebb (1949), which states that if activity originating in neuron A often contributes to bringing some other neuron B to fire, then the connection from A to B tends to become stronger, increasing in conductivity or synaptic weight.

**ligand** A molecule which binds to a *receptor* to form a *ligand-receptor complex.* In general, ligands may be either soluble or membrane-bound. In the NeuroGene simulation of ligand-receptor binding, ligands must be soluble.

**ligand-receptor complex** The molecule formed when a *ligand* binds to a *receptor.*

**– M –**

**master gene** See *cascade.*

**membrane** The cell membrane surrounds the cell, maintaining its structural integrity, and separating the intracellular environment from the *extracellular space.* This function is vital for the function of the cell, since the intracellular space is completely different from the extracellular space in terms of dissolved chemicals, electric potential, etc. Cells also have numerous intracellular organs (known as organelles) which are also surrounded by membranes. This internal structure is not represented in NeuroGene. See also *membrane potential.*

**membrane bound** Proteins which remain associated with the cell membrane of the cells that produce the proteins are membrane bound. An important class of membrane bound proteins is *receptors*, which act as "antennae" allowing the cell to react to chemical changes in the *extracellular space* close to the cell.

**membrane potential** The electronic voltage across the *membrane* of cells. While most cells have non-zero membrane potentials, these play a particularly important role in *neurons*, where changes in the membrane potential serve as a means to transmit information between interconnected neurons. See also *resting potential* and *threshold potential.*

**moles** One mole is defined as $6.022 \times 10^{23}$ molecules — for a compound with molecular weight of $x$, one mole corresponds to $x$ grams. $6.022 \times 10^{23}$ is known as Avogadro's number. See also *molar.*

**molar** One molar is defined as $6.022 \times 10^{23}$ molecules per litre — for a compound with molecular weight of $x$, the concentration of $1M$ corresponds to $x$ grams per litre. See also *moles.*

**morphology** The technical term for the physical size and shape of cells or cellular structures. Morphological changes covers changes in cell shape (such as growth of axons and dendrites and the formation of synapses), as well as the location of cells (i.e. cell migration) and number of cells (cell division).

**mRNA** Messenger RNA, a particular type of RNA which is used to transmit genetic information from the *genome* to sites of *protein* production (*translation*). *Transcription* is the process of creating a copy of a gene's DNA code in the form of a messenger RNA molecule. See also *RNA*.

– **N** –

**Neurite** The Java class used to represent *axons* and *dendrites.*

**neurite** A common term covering both *axons* and *dendrites.* Sometimes used to refer to non-differentiated nerve fibres which subsequently become either axons or dendrites. In this thesis the term has the former meaning.

**neuron** A cell specialized for the processing and transmission of information through electric impulses.

**neurotransmitter** A chemical which is used in *synapses* to transmit an *action potential* from the *presynaptic* to the *postsynaptic terminal.* The presynaptic terminal has specialized machinery for the production, release and re-absorption of neurotransmitter, while the postsynaptic terminal contains machinery for the detection of the neurotransmitter. The presynaptic terminal responds to an action potential by releasing neurotransmitter, and the postsynaptic terminal responds to the neurotransmitter by opening *ion channels*, thereby making the membrane potential less negative and potentially causing an action potential in the postsynaptic cell.

**nucleus** Cellular component which contains the cell's only copy of the entire genome stored in DNA.

– **P** –

**parent** In the context of *tree* data structures, the parent node $p$ of a node $n$ is defined as follows: there is an edge connecting $n$ and $p$, and the shortest path from $p$ to the root node is shorter than the shortest path from $n$ to the root node.

**parser** A software component which can read an input string and generate an *abstract syntax tree* representation of that string. Parsers are typically generated automatically based on an annotated grammar of the underlying programming language.

**patterning** The process through which an initially uniform population of cells become subdivided into (usually spatially contiguous) domains of cells with different properties. See also *differentiation.*

**phenotype** The characteristics of an organism as expressed in the organism's shape and function. See also *genotype.*

**PreSynapse** The Java class used to represent *presynaptic termini.*

**presynaptic terminal** See *synapse.*

**protein** Complex molecules involved in all processes of a living organism. Proteins consist of long chains made up of hundreds or thousands of *amino acids.* The properties of the proteins are determined by the sequence of amino acids. Proteins are produced through the process of *translation.*

**protein coding region** The part of a gene which contains the information which through the processes of *transcription* and *translation* specifies the sequence of amino acids which make up a protein expressed by the gene. Through this process, a given gene can be said to code for a particular protein.

**PostSynapse** The Java class used to represent *postsynaptic termini.*

**postsynaptic terminal** See *synapse.*

**– Q –**

**query** In the NeuroGene genetic programming language, queries represent ways in which the state of cell components and their environment may affect gene expression within those cell components. A query takes the form of a function call, which returns a value of the appropriate type. This value can then be used within the gene to compute its expression rate or to modify the effects of gene expression through parameters to *actions.* Queries play a similar role in *growth cone functions.*

**– R –**

**receptor** A protein which is bound to the cell membrane and which can bind to extracellular signaling molecules. Such binding triggers intracellular signaling events which may lead to changes in gene expression. Receptors are also important in intracellular processes, but this cannot be represented in NeuroGene. See *ligand-receptor binding.*

**refractory delay** A period of insensitivity to incoming neural activity after a neuron has produces an action potential (i.e. "fired").

**regulation** See *gene regulation.*

**regulatory element** See *regulatory protein.*

**regulatory protein** A protein which binds to specific sequences of DNA. The DNA sequences which bind regulatory proteins are known as regulatory elements. The presence of regulatory proteins at regulatory elements of a given gene facilitates or hinders the transcription of the genes, thereby affecting the rate of expression. The activity of regulatory proteins is the most important mechanism by which *gene regulation* occurs.

**resting potential** The *membrane potential* of a *neuron* which has not generated an *action potential* recently, typically in the range of -70mV, with the inside of the cell being more negative. After a neuron generated an action potential, the membrane potential decays gradually toward the resting potential.

**ribosome** Cellular machinery responsible for the translation of *mRNA* into *protein.*

**RNA** RiboNucleic Acid. A chemical compound similar to DNA, also consisting of four different nucleotides. The information content in DNA is translated into RNA using nucleotide complementarity. See also *mRNA* and *DNA.*

**– S –**

**soluble** Soluble proteins do not associate with membranes, but diffuse freely through solutions, such as that which fills the *extracellular space.*

**Soma** The Java class used to represent *somas.*

**soma** The central part of a *neuron*, containing the cell nucleus with the *genome*. All activities of the cell are controlled from the soma through gene expression. The decision for the cell to fire an action potential is also taken at or near the soma.

**spontaneous firing** Some *neurons* generate *action potentials* spontaneously, i.e., even when they have not received action potentials from other cells. See section 2.7.

**stack** A data structure which holds a collection of items. The data structure supports only two operations, push, which adds an element to the collection, and pop, which removes an element. Pop always removes the element which was the last to be added to the collection.

**synapse** A neural connection between two neurons, consisting of a *presynaptic* and a *postsynaptic* terminal. Synapses are asymmetric structures, and the flow of *action potentials* can only occur in the direction from the presynaptic to the postsynaptic terminal. Action potentials travel across the synapse in the form of a pulse of chemical signaling molecules, which at the postsynaptic side is converted back into an electric action potential. See *neurotransmitter*.

**– T –**

**threshold potential** When a *neuron* receives *action potentials* from other cells, its *membrane potential* becomes less negative. When the membrane potential reaches the threshold potential, the neural generates an *action potential*. The threshold potential is typically close to zero mV.

**transcription** The first step of *gene expression*. The information in the coding region of a gene is copied from the DNA of the genome to a molecule of mRNA. The process is regulated by *regulatory proteins*. This process occurs in the cell nucleus. *Gene regulation* is primarily implemented through the regulation of transcription. See also *translation*.

**transcription factor** See *regulatory protein*.

**translation** The second step of *gene expression*. The process by which protein chains are assembled from *amino acids*. The sequence of amino acids in the chain is determined by the information carried in a molecule of *mRNA*. This information originates in the DNA of the organisms genome. Amino acids are added one by one to the growing protein chain. At each addition, the type of amino acid is determined by the information carried by a molecule of mRNA using the *universal genetic code*. This process is carried out by cellular machinery known as ribosomes, which are dispersed throughout the cell. Through interactions between the mRNA and proteins, translation can be enhanced or reduced, giving an additional level control over protein production, in addition to the regulation of *transcription*.

– **U** –

**universal genetic code** The sequences of *amino acids* in the of an organism's *proteins* is encoded in the *DNA* of the organism's *genome* using the genetic code. This code translates from triplets (or "codons") of DNA nucleotides to particular amino acids. There are four different nucleotides, giving $4^3 = 64$ different codons. Each of these code for one of the 20 naturally occurring amino acids, or for the special symbol "stop" which terminates *translation*. Almost all known biological organisms share the same genetic code.

– **W** –

**World** NeuroGene class used to represent the simulation space.

**WorldNode** NeuroGene class used to represent a small, cubic volume element of the simulation space.

# Appendix B

# Gene scripts

In this appendix, gene and geometry scripts used to initialize and control various NeuroGene simulations are listed. In addition to giving details about the relevant experiments, these scripts also give examples of the syntax of gene and geometry scripts. Comments are indicated in the same way as in C++ and Java, using `/*...*/` and `//...`. The notation of the form "`5.3e-2`" used throughout these scripts represents numbers in exponential form, i.e., $5.3 \times 10^{-2}$.

## B.1 Scripts for verification of implementation

The following scripts were used to construct the experiments reported in chapter 4, in order to demonstrate the correctness of implementations of various algorithms in NeuroGene.

### B.1.1 Verification of concentration measurements

The first set of experiments were used to validate the measurements of extracellular protein concentrations, which are computed by linear interpolation from values in particular WorldNodes, see section 4.1.

#### Gene script

```
/*
 * This gene is never expressed, see the geometry script.
 */
gene Protein{
    diffusible = true;
    decay = 0;
    diffusion = 0;
}
/*
 * The role of this gene is to report the measurements by printing them
 * to the console using the print() action. It also moves the cell
 * component to a new location each simulation time step.
 */
terminal gene Measure {
    regulation {
        express();
    }
    effects {
        print("" +
            () + "\t" +
            externalConcentrationOf(Protein) + "\t" +
            towardsSignal(Protein));
        migrate([0.1,0,0]);
    }
}
```

### Geometry script

```
import "ConcMeasurement.gene";
space(10,10,10);
// An exponential protein concentration gradient is established.
signal(Protein, exp(x/3));
// Build the cell component which will do the measurement.
```

```
buildSoma(-0.4,5,5);
```

### B.1.2 Measurement of gradient field

The following scripts were used to set up the experiments used to verify the evaluations of protein concentration gradient vectors. This computation also relies of linear interpolation, see section 4.2. The genome is very similar to the one shown above, except that the simulated gene *Measure* does not cause cell components to migrate.

#### Gene script

```
gene Protein{
    diffusible = true;
    decay = 0;
    diffusion = 0;
}
terminal gene Measure {
    regulation {
        express();
    }
    effects {
        print("" +
            cellXCoordinate() + "\t" +
            cellYCoordinate() + "\t" +
            externalConcentrationOf(Protein) + "\t" +
            towardsSignal(Protein));
    }
}
```

### Geometry script

This script builds a 2D array of cell components. All of these will measure the gradient vectors during a single simulation time step.

```
import "GradMeasurement.gene";
// These constants define the size of the simulation space.
variable X = 6;
variable Y = 6;
space(X,Y,1);
// Establish the protein concentration gradient with spherical symmetry.
signal(Protein, 0.1*sqrt(2*X*X - x*x - y*y));
// This constant defines the distance between the cell components.
variable PITCH = 0.25;
for(variable xx=-0.4999; xx <= X - 0.5; xx += PITCH) {
    for(variable yy=-0.4999; yy <= Y - 0.5; yy += PITCH) {
        buildSoma(xx, yy, 0);
    }
}
```

## B.1.3 Protein decay

The following scripts were used to verify the NeuroGene protein decay simulation. The results of this verification are given in section 4.3.

### Gene script

```
/*
 * The decay of this protein will be measured.
 */
gene Protein{
    diffusible = false;
    decay = 0.05;
}
/*
 * This gene is used to keep track of the duration of the simulation.
 */
gene Counter {
    diffusible = false;
    decay = 0;
    regulation {
        expressExternally(1.0);
    }
}
/*
 * This gene will measure the concentrations of both proteins
 * defined above.
 */
terminal gene Measure {
    regulation {
        express();
    }
    effects {
        print("" + 1 + surfaceConcentrationOf(Counter) +
           " " + surfaceConcentrationOf(Protein));
    }
}
```

### Geometry script

This geometry scripts constructs a simulation with a single cell component. The cell component is initialized to express a concentration of 1.0 of *Protein* on its surface.

```
import "Decay.gene";
space(1,1,1);
```

```
buildSoma(0, 0, 0) {
    expressExternally(Protein, 1.0);
};
```

### B.1.4 Ligand-receptor binding

The following scripts were used to verify the computation of ligand-receptor binding. The simulation is constructed so that ligand is produced by one cell component, and two other cell components, carrying differing amounts of receptor, compete for this ligand. As the simulation progresses, increasing amounts of ligand is produced. The results of this verification are given in section 4.4.

#### Gene script

```
/*
 * This gene defines the propertied of the receptor protein. In this
 * simulation, the gene is never expressed, instead the cell components
 * are initialized to carry a certain amount of receptor, see the
 * geometry script.
 */
gene Receptor {
    diffusible = false;
    decay = 0;
}
/*
 * This gene defines the properties of the ligand. Its expression depends
 * on the Producer gene.
 */
gene Ligand {
    diffusible = true;
    decay = 1;
    diffusion = 0;
    regulation {
        if(internalConcentrationOf(Producer) > 0){
            expressExternally(internalConcentrationOf(Producer));
        }
    }
}
/*
 * This gene defined the properties of the ligand-receptor complex. Such
 * genes are never expressed.
 */
gene Complex {
    diffusible = false;
    decay = 0;
}
/*
 * This statement defines the receptor-ligand relationship, with a
 * binding constant of 0.01.
 */
receptorDefinition(Ligand, Receptor, Complex, 1e-2);
/*
 * This gene is self-enhancing, so that its concentration is doubled
 * every time step.
 */
gene Producer {
    regulation {
        if(internalConcentrationOf(Producer) > 0){
            expressInternally(internalConcentrationOf(Producer));
        }
    }
}
/*
 * This gene measures the concentratons of ligand, receptor and complex.
 * Its expression is suppressed by Producer.
 */
terminal gene Measure {
    regulation {
        if(internalConcentrationOf(Producer) == 0){
            express();
        }
    }
    effects {
        print(
            "" + externalConcentrationOf(Ligand) +
            "\t" + surfaceConcentrationOf(Receptor) +
            "\t" + surfaceConcentrationOf(Complex));
    }
}
```

### Geometry script

A simple geometry scripts creates a simulation consisting of one producer soma and two somas carrying receptor and competing to binding the ligand produced by the producer cell. Note that the competitor cells express different amounts of the receptor.

```
import "LigandBinding.gene";
space(1, 1, 1);
buildSoma(0, 0, 0, "Producer") {
    expressInternally(Producer, 1e-12);
};
buildSoma(0.1, 0.1, 0.1, "CompetitorA") {
    expressExternally(Receptor, 0.1);
};
buildSoma(-0.1, -0.1, -0.1, "CompetitorB") {
    expressExternally(Receptor, 0.02);
};
```

### B.1.5 Simulation of protein diffusion

The following scripts were used to verify the NeuroGene simulation of diffusion. All experiments use the same gene script, while different geometry scripts are used to define the various different situations under which diffusion simulation is carried out. The results of this verification is given in section 4.5.

#### Gene script

```
/*
 * This gene defines the properties of the protein whose diffusion
 * is simulated.
 */
gene Protein {
    diffusible = true;
    diffusion = 0.05;
    decay = 0.0;
}
/*
 * This gene measured the concentration of the protein and prints
 * it to the console.
 */
terminal gene Observe {
    regulation {
        if(internalConcentrationOf(Protein) == 0){
            print(externalConcentrationOf(Protein));
        }
    }
}
```

### Geometry script: For "open" experiments

The following geometry script is used to initialize simulations in which the protein is free to diffuse out of the simulation space. In the 1D case, a long narrow simulation space ($1 \times 1 \times 32$ WorldNodes) is created, which is open to diffusion only through the $1 \times 1$ end surfaces. In the 2D case, a flat simulation space ($1 \times 32 \times 32$ WorldNodes) is created, where the thin $1 \times 32$ edge surfaces are open to diffusion. In the 3D case, a cubic simulation space ($32 \times 32 \times 32$ WorldNodes) which is open to diffusion through all surfaces is created.

In each case a 1D row of cells parallel to the X axis and passing through the centre of the simulation space is constructed. These cells report the extracellular concentration of the diffusing protein in each simulation time step.

```
1  import "Diffusion.gene";
2  variable pi = 180;
3  // These variables define the dimensions of the simulation space.
4  variable Lx = 32;
5  variable Ly = 1;
6  variable Lz = 1;
7  // Create the initial protein concentration gradient.
8  signal(Protein, sin(pi*(x+0.5)/Lx)*sin(pi*(y+0.5)/Ly)*sin(pi*(z+0.5)/Lz));
9  space(Lx, Ly, Lz);
10 // Define which faces are permeable to diffusing protein. The first
11 // parameter applied to faces perpendicular to the X-axis, the second
12 // to the Y-axis and the third to the Z-axis.
13 closed(false, true, true);
14 // Build a row of cells.
15 for(variable x = 0; x < Lx; x++){
16     buildSoma(x, 0, 0, "" + x);
17 }
```

To create the 2D and 3D experiments, the same script is used as is shown above, except lines 4–6 are replaced with

```
variable Lx = 32;
variable Ly = 32;
variable Lz = 1;
```

to create the 2D experiment, and with

```
variable Lx = 32;
variable Ly = 32;
variable Lz = 32;
```

to create the 3D experiment. In addition the statement on line 13 is replaced with "`closed( false, false, true);`" or "`closed(false, false, false);`" in the 2D and 3D cases, respectively.

### Geometry script: For "closed" experiments

The following script was used to initialize the simulations used to verify diffusion computations in the case where the simulation space is closed, i.e., proteins are not free to diffuse out of the simulation space. Again, 1D, 2D and 3D experiments of the same dimensions as above were run, using the three variables `Lx`, `Ly` and `Lz` to define those dimension.

```
import "Diffusion.gene";

variable pi = 180;
variable Lx = 32;
variable Ly = 1;
variable Lz = 1;

signal(Protein, 1+cos(pi*(x+0.5)/Lx));
space(Lx, Ly, Lz);
closed(true,true,true);

for(variable x = 0; x < Lx; x++){
    buildSoma(x, 0, 0, "" + x);
}
```

The initial concentration functions were also varied between the 1D, 2D and 3D experiments, so that line 8 was changed to

```
signal(Protein, 1+cos(pi*(x+0.5)/Lx)*cos(pi*(y+0.5)/Ly));
```

in the 2D version and to

```
signal(Protein,
   1+cos(pi*(x+0.5)/Lx)*cos(pi*(y+0.5)/Ly)*cos(pi*(z+0.5)/Lz));
```

in the 3D version.

## B.2 Neurodevelopmental simulations

In the following, the gene scripts used as basis for the developmental and neurodevelopmental simulations reported in chapter 5 are listed.

### B.2.1 Gene script for fruit-fly Even-stripe system

The following gene script is used to simulate the development of initial segmentation in fruit-fly. This experiment is reported in section 5.1.

```
gene Nanos {
   diffusible = true;
   decay = 0;
   diffusion = 0;
}
gene Bicoid {
   diffusible = true;
   decay = 0;
   diffusion = 0;
}
```

```
gene Hunchback_mRNA {
   diffusible = true;
   decay = 5e-2;
   diffusion = 1e-2;
   regulation {
      if (externalConcentrationOf(Bicoid) > 2e-6) {
         expressExternally(1e-5);
      }
   }
}
gene Hunchback {
   diffusible = true;
   decay = 5e-2;
   diffusion = 5e-2;
   regulation {
      if (externalConcentrationOf(Nanos) < 1e-6) {
         expressExternally(externalConcentrationOf(Hunchback_mRNA));
      }
   }
}
gene Knirps {
   diffusible = true;
   decay = 5e-2;
   diffusion = 1e-2;
   regulation {
      if (externalConcentrationOf(Hunchback) < 1e-11) {
         expressExternally(1e-5);
      }
   }
}
gene Kruppel {
   diffusible = true;
```

```
    decay = 5e-2;
    diffusion = 1e-2;
    regulation {
        if (externalConcentrationOf(Hunchback) < 1e-7
                && externalConcentrationOf(Knirps) < 1e-7) {
            expressExternally(1e-5);
        }
    }
}
gene Giant {
    diffusible = true;
    decay = 5e-2;
    diffusion = 1e-2;
    regulation {
        if (externalConcentrationOf(Bicoid) > 3e-6) {
            expressExternally(1e-5);
        }
    }
}
gene EvenStripe {
    diffusible = true;
    decay = 5e-2;
    diffusion = 1e-2;
    regulation {
        if(externalConcentrationOf(Bicoid) > 5e-7
                && externalConcentrationOf(Hunchback) > 1e-9
                && externalConcentrationOf(Kruppel) < 1e-6
                && externalConcentrationOf(Giant) < 1e-10) {
            expressExternally(1e-5);
        }
    }
}
```

### B.2.2 Gene script for topographic map

The following gene is used in the simulation of the development of initial (activity-independent) topographic map formation between the retina and the optic tectum. The first few genes are cell identity gene for the various cell types involved in the simulation. These genes are all self-enhancing, and they are also important in enhancing other genes which are expressed only in certain cell types.

```
gene tectalCellMarker {
    diffusible = false;
    decay = 0.3;
    regulation {
        if (surfaceConcentrationOf(tectalCellMarker) > 0) {
            expressExternally(1e-4);
        }
    }
}
gene retinalCellMarker {
    diffusible = false;
    decay = 0.2;
    regulation {
        if (surfaceConcentrationOf(retinalCellMarker) > 0) {
            expressExternally(0.1);
        }
    }
}
```

The next set of genes are the Eph receptor genes. These are expressed in retinal ganglion cells (RGCs), including their axons and growth cones. The expression pattern of these genes is defined by a number of numerical parameters.

```
constant EphA3_A = 0.25;
constant EphA3_B = -1.2;
constant L0 = 0.25; // amplitude of the high end of the gradient
constant L1 = 0.075; // amplitude of the low end of the gradient

constant B = -5;
constant A = (L1-L0)/(exp(B)-1);
constant C = L0 - A;
/*
 * The EphA3 gene is expressed in a gradient along the horizontal
 * dimension of the retina.
 */
gene EphA3 {
    diffusible = false;
    decay = 0.6;
    regulation {
        if (surfaceConcentrationOf(retinalCellMarker) > 0) {
            variable x = internalConcentrationOf(nasalTemporalGradient);
            expressExternally (A*exp(B*x)+C
                                + L0 * internalConcentrationOf(knockin));
        }
    }
}
/*
 * The EphB3 gene is expressed in a gradient along the vertical
 * dimension of the retina. It is otherwise a direct copy of the
 * EphA3 gene.
 */
gene EphB3 {
    diffusible = false;
    decay = 0.6;
    regulation {
        if (surfaceConcentrationOf(retinalCellMarker) > 0) {
            variable y = internalConcentrationOf(dorsoVentralGradient);
            expressExternally (A*exp(B*y)+C;
        }
    }
}
```

The next set of genes are the ephrins. These are expressed in gradients in the tectal target area which are innervated by RGC axons.

```
constant ephrinA2_intercept = 0.5;
constant ephrinA2_exponent = log(5);
/*
 * ephrinA2 is expressed in a gradient along the horizontal dimension of
 * the tectum.
 */
gene ephrinA2 {
    diffusible = false;
    decay = 0.6;
    regulation {
        if (surfaceConcentrationOf(tectalCellMarker) > 0) {
            variable x = internalConcentrationOf(ApGradient);
            expressExternally(ephrinA2_intercept
                                * exp(-ephrinA2_exponent * x));
        }
    }
}
/*
 * ephrinB2 is expressed in a gradient along the vertical dimension of
 * the tectum. It is otherwise a direct copy of the ephrinA2 gene.
 */
gene ephrinB2 {
    diffusible = false;
    decay = 0.6;
    regulation {
        if (surfaceConcentrationOf(tectalCellMarker) > 0) {
            variable y = internalConcentrationOf(LmGradient);
            expressExternally(ephrinA2_intercept
                                * exp(-ephrinA2_exponent * y));
        }
    }
}
```

The following genes are used to i) define the coordinates of RGCs and tectal cells (the four first) or ii) defining the level of EphA3 knockin in the experiments after Brown et al. (2000). The concentrations of the proteins defined by these genes are all initialized at simulation setup, and do not change during the simulation. These genes all have default properties: They are soluble with decay rate and diffusion rate of zero.

```
gene nasalTemporalGradient {}
gene dorsoVentralGradient {}
gene ApGradient {}
gene LmGradient {}
gene knockin {}
```

The following gene defines a soluble protein which is expressed in the tectum, and which diffused towards the retina, thereby establishing a concentration gradient which RGC axons follow in order to navigate to the tectum.

```
gene retinoTectalGradient {
    diffusible = true;
    decay = 0.1;
    diffusion = 0.1;
    regulation {
        if (surfaceConcentrationOf(tectalCellMarker) > 0) {
            expressExternally (0.1);
        }
    }
}
```

The following gene and growth cone function define how RGC axons navigate from the retina to the tectum.

```
terminal gene growRgcAxon {
   regulation {
        if (surfaceConcentrationOf(retinalCellMarker) > 0 &&
           insideSoma() && timerValue(wait)==0 && numberOfAxons()==0) {
            express();
        }
    }
    effects {
        growAxon();
    }
}
```

The next gene defines the condition under which growth cone are formed using the growth cone function **findTectum**. These conditions are such that this will only occur in RGC axons, and it is suppressed by the external presence of tectal cell marker protein, which indicates that the axons have arrived in the tectum and the growth cone behaviour needs to change.

```
terminal gene FindTectum {
    regulation {
        if (insideAxon()
            && surfaceConcentrationOf(retinalCellMarker) > 0
            && externalConcentrationOf(tectalCellMarker) < 1e-4) {
            express();
        }
    }
    effects {
        growthcone(findTectum);
    }
}
/*
 * The growth cone function is simple, the bid value being equal to the
 * extracellular concentration of the guiding protein.
 */
growthconeFunction findTectum {
    filopodiumCount = 4;
    filopodiumLength = 0.2;
    searchRange = 0.0;
    neighbourCount = 0;
    migrate(externalConcentrationOf(retinoTectalGradient));
}
```

The next gene and growth cone function define the behaviour of RGC axons which cause these to form and maintain a sorted topographic projection in two dimensions. This behaviour is enhanced by the extracellular presence of tectal cell marker protein, meaning that RGC axons will change to this behaviour once they reach the tectum.

```
terminal axon gene SortMap {
    regulation {
        if (insideAxon()
            && surfaceConcentrationOf(retinalCellMarker) > 0
            && externalConcentrationOf(tectalCellMarker) >= 1e-4) {
                express();
        }
    }
    effects {
            growthcone(sortMap);
        }
}
```

The following growth cone function encodes the topographic map formation algorithm as it has been outlined in section 5.2.

```
growthconeFunction sortMap {
    filopodiumCount = 8;
    filopodiumLength = 0.15;
    searchRange = 1.7;
    neighbourCount = 15;

    // the filter ensures that only retinal axons
    // are considered as neighbours
    filter = surfaceConcentrationOf(retinalCellMarker) > 0
       && insideAxon();

    // The bid value is computed based on a number of different gradients
    // which are evaluate here. These are either between the tip and the
    // base of the filopodium (e.g. retinotec_gradient) or between the
    // axon itself and its neighbour (e.g. EphA_difference).
    variable retinotec_gradient =
        externalConcentrationOf(retinoTectalGradient)
            - growthconeConcentrationOf(retinoTectalGradient);

    variable ephrinA_gradient = externalConcentrationOf(ephrinA2)
        - growthconeConcentrationOf(ephrinA2);

    variable EphA_difference =
        surfaceConcentrationOf(EphA3) - neighbourConcentrationOf(EphA3);

    variable ephrinB_gradient = externalConcentrationOf(ephrinB2)
        - growthconeConcentrationOf(ephrinB2);

    variable EphB_difference =
        surfaceConcentrationOf(EphB3) - neighbourConcentrationOf(EphB3);

    // the score is used to store the bid value as it is being evaluated.
    variable score = 50 * retinotec_gradient;

    // The relative pecking order with the neighbouring RGC axon
    // determines whether the ephrinA gradient is attractive or
    // repulsive. EphA_difference>0 indicates that the axon is higher in
    // pecking order than the neighbouring RGC, in which case the ephrinA
    // gradient is repulsive.
    if (EphA_difference > 0) {
        score -= ephrinA_gradient;
    } else if (EphA_difference < 0) {
        score += ephrinA_gradient;
    }
    // Here the pecking order in the vertical dimension is determined.
    // If the axon is higher in the pecking order than the neighbour
    // (when EphB_difference>0), then the ephrinB gradient is attractive,
    // otherwise 'it is repulsive.
    if (EphB_difference > 0) {
        score += ephrinB_gradient;
    } else if (EphB_difference < 0) {
        score -= ephrinB_gradient;
    }
    // This is the actual growth cone command with the bid value.
    migrate(score);
}
```

### B.2.3 Gene script for refinement and ocular dominance

The following gene script is used to simulate the activity-dependent developmental processes reported in section 5.3. As in the genome listed above, the first genes are self-enhancing cell identity genes. In the case of the starburst cell marker gene, the gene's effect section causes the duration of the refractive period of each cell to be continuously re-randomized with stochastic values from a Gaussian distribution.

```
soma gene starburstCellMarker {
    diffusible = false;
    decay = 0.2;
    regulation {
        if(surfaceConcentrationOf(starburstCellMarker) > 0) {
            expressExternally(1e-2);
        }
    }
    effects {
        // continuously re-randomize the refractive period of
        // starburst cells.
        setRefractoryDelay(gaussian(60,20));
    }
}
soma gene tectalCellMarker {
    diffusible = false;
    decay = 0.3;
    regulation {
        if(surfaceConcentrationOf(tectalCellMarker) > 0) {
            expressExternally(1e-4);
        }
    }
}
gene retinalCellMarker {
    diffusible = false;
    decay = 0.2;
    regulation {
        if(surfaceConcentrationOf(retinalCellMarker) > 0){
            expressExternally(1);
        }
    }
}
```

The following genes together implement the Elliott and Shadbolt learning algorithm, see appendix H. The following parameters define the parameter values of the algorithm.

```
constant learning_rate = 1e-2;
constant fast_decay = 1;
constant T1 = 2e-2;
constant T0 = 0;
constant lambda = 1e6;
```

The concentration of the protein expressed by this gene represents the recent average firing rate of the cell component.

```
presynapse gene timeAverageActivity {
    diffusible = true;
    decay = -log(1 - learning_rate);
    regulation {
        if (cellIsFiring()
           && surfaceConcentrationOf(retinalCellMarker) > 0) {
            expressInternally(learning_rate);
        }
    }
}
```

The gene NT encodes the soluble protein which acts as a retrograde messenger to the presynaptic terminal notifying this of the current level of the membrane potential of the postsynaptic cell.

```
soma gene NT {
    diffusible = true;
    decay = fast_decay;
    diffusion = 0;
    regulation {
        if (surfaceConcentrationOf(tectalCellMarker) > 0) {
            variable weight = 0;
            foreach (cellvariable i in allPostsynapses()) {
                weight += i.synapticWeight();
            }
            variable expression = T0;
            if (weight > 0 && membranePotential() > 0) {
                expression += T1 * membranePotential()/weight;
            }
            expressLocally(expression);
        }
    }
}
```

The receptor of the NT intercellular signaling molecule is expressed by presynaptic termini.

```
presynapse gene NTreceptor {
    diffusible = false;
    decay = fast_decay;
    regulation {
        if(surfaceConcentrationOf(retinalCellMarker) > 0) {
           if(cellIsFiring()) {
                // since the expression of TimeAverageActivity that
                // happens in the current time step is not yet detectable
                // in another gene, the expressed amount (equal to the
                // learning rate) needs to be included here
                variable rho = lambda*(learning_rate +
                    internalConcentrationOf(timeAverageActivity))
                        /arborOutgoingWeight();
                expressExternally(synapticWeight()*2*rho);
            } else {
                variable rho = lambda*
                    internalConcentrationOf(timeAverageActivity)
                        /arborOutgoingWeight();
                expressExternally(synapticWeight()*1*rho);
            }
        }
    }
}
```

The complex gene has, as usual, no regulation or effects sections.

```
gene NTcomplex {
    diffusible = false;
    decay = fast_decay;
}
```

This statement defines the receptor-ligand relationship implementing the competitive aspect of the Elliott and Shadbolt learning algorithm. For the significance of *local* receptors, see below.

```
local receptorDefinition(NT, NTreceptor, NTcomplex, 1e-12);
```

This gene implements the actual learning algorithm using the information computed by the receptor-ligand relationship above. The gene is expressed in all RGC presynaptic terminals.

```
94  terminal presynapse gene learn {
95      regulation {
96          if(surfaceConcentrationOf(retinalCellMarker) > 0) {
97              express();
98          }
99      }
100     effects {
101         variable delta =
102             learning_rate * (
103                 surfaceConcentrationOf(NTcomplex) - synapticWeight()
104             );
105         setSynapticWeight(synapticWeight() + delta);
106         print("weight = " + (synapticWeight() + delta));
107     }
108 }
```

## B.3 Local signal expression

A special form of the ligand-receptor simulation system has been implemented to allow the trans-synaptic competition for protein to be implemented. As far as the gene language is concerned, this functionality is accessed using the `local` tag on the definition of the receptor-ligand relationship (see line 93 above), as well as the `expressLocally()` action used on line 62 above. The command `expressLocally()` is only valid for the expression of soluble proteins.

Consider a situation with a cell component $A$, which forms synapses with a set of $N$ other cell components $B_i$ with $0 \leq i < N$. For the synapse between $A$ and $B_i$, the two synaptic termini are labeled $a_i$ and $b_i$, where $a_i$ is part of the same cell as $A$, and $b_i$ is part of the same cell as $B_i$. If $A$ then expresses an amount of a soluble signal locally (i.e. using `expressLocally()`), that amount of signal can bind to receptors carried by the synaptic termini $b_i$ *only* for $0 \leq i < N$. It can not bind to receptors on $a_i$, nor on the cell components $B_i$. Conversely, if $B_i$ for some value of $i$ expresses a soluble protein locally, it can only bind to receptors on the single synaptic terminus $a_i$.

The advantage over this system over that outlined in section 2.4.4 is that it avoids *cross-talk* among cells that are close together in space. Using normal extracellular expression of the protein $NT$, the $NT$ signals from two cells $A$ and $A'$ which both are located in the same WorldNode can not be distinguished. Using local expression, the protein signals these cells emit are only received by synapses which connect directly to these cells, thereby removing the cross-talk effect.

Comparing to algorithm 1 listed on page 63, the local version makes the following changes:

Line 1: $[L]_0$ is the local concentration of the ligand at some cell component $C$.

Lines 2-3: $[R]_0$ and $[C]_0$ are the total surface concentrations of receptor and complex on all synaptic terminals facing $C$, i.e., all synaptic terminals whose partners' parent is equal to $C$.

Line 7: The adjustments of ligand concentration is made to the local concentration at the cell component $C$.

Line 9: For-loop iterates over the synaptic terminals described for lines 2-3 above.

Lines 11-12: The adjustments of receptor and complex concentrations are made to presynaptic terminals synaptic terminals described for lines 2-3 above.

# Appendix C

# Automatic cell naming

All cells in NeuroGene are identified by a name. The name may be the empty string, and it need not be unique. When cells are divided, new names for the two daughter cells may be specified. If new names are *not* specified, the daughter cells are automatically given names which are derived from the name of the parent cell. This makes it possible to trace the ancestry of any cell in a simulation. The naming scheme is as follows. Given a cell $C$ with name "Foo", which will be divided to form two daughter cells $C_1$ and $C_2$. The cells will be named according to the following rules:

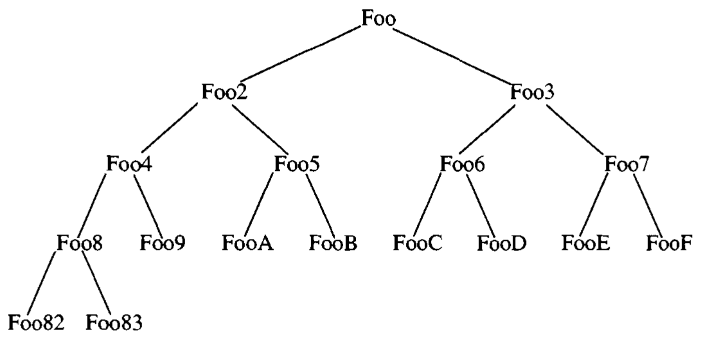


Figure C.1: Example of naming of a set of related cells ("Foo" being the parent) according to the rules outlined here.

If $S$ ends with one of the characters in the range '2' - '7', then that character is replaced by the value $2c$ to give the name of $C_1$ and with the value $2c + 1$ to give the name of $C_2$, where $c = 2$ if the character was '2', etc. Values in the range 10-15 are represented by the characters 'A' – 'F'. For example, if $S$ = "Foo6", then $C_1$ will be named "FooC" (where 'C' represents 12) and $C_2$will be named "FooD" (where 'D' represents 13).

If $S$ ends in some character not in the range '2' – '7', then the character '2' is appended to $S$ to give the name of $C_1$, and the character '3' is appended to $S$ to given the name of $C_2$.

Figure C.1 shows the names given to daughter cells through four generations of a cell named "Foo". The cell names become one character longer for every three generations, as is shown in the case of the offspring of the cell "Foo8".

This naming scheme can be used to trace back the ancestry of a given cell. E.g. a cell with the name "FooE86" has parents (going back eight generations) "FooE83", "FooE8", "FooE4", "FooE2", "FooE", "Foo7", "Foo3" and "Foo".

## C.1 Regular expressions

Regular expression matching is implemented in NeuroGene using classes from **`java.util.-regex`** (Sun Microsystems, 2004, package java.util.regex). Regular expressions work as follows: For any string $S$ and any well-formed regular expression $E$, it can be determined whether $S$ *matches* $E$. Regular expressions represent a powerful tool because for any set of strings $S_1 - S_n$, there exists a regular expression which will match these strings *and no others.*

A regular expression consists of *characters* and *meta-characters.* A regular expression which consists only of characters, will match only itself, such as the regular expression **`"Example"`**, which matches only the string "Example". Regular expressions which contain meta-characters may match more than one string, for example, the regular expression **`"[Ee]xample"`** matches both "Example" and "example".

The most commonly used meta-characters are listed below:

. (any character) A dot matches any character, so that the regular expression **`"Ex.mple"`** matches the string "Example", as well as all strings where the "a" has been substituted with any other character, such as "ExAmple", "Exwmple", "Ex8mple", "Ex&mple", "Ex mple", etc.

`[...]` (range) Square brackets indicate a choice of possible matching characters, so that **`"Ex[ac]mple"`** matches the two strings "Example", "Excmple", and no others.

Ranges can also be specifies, so that ``Ex[a-c]mple'' matches the three strings "Example", "Exbmple", "Excmple" and not others. Finally, these can be combined as in ``Ex[a-cpx-z]mple'', in which the choice is between the letters a, b, c, p, x, y, and z.

? (zero or one) A normal character followed by a ``?'' in a regular expression means that the character may be absent or present, so that the regular expression ``Exa?mple'' will match the two strings "Example", "Exmple", and no others.

* (zero or more) A star is used in the same way as the ``?'', and means that zero or more repeats of the ordinary character is matched, so ``Exa*mple'' will match all the strings with zero or more "a"s in them, such as "Exmple", Example", Exaample", etc.

+ (one or more) A plus matches one or more occurrences of the preceding ordinary character, so that ``Exa+mple'' will match the same strings as "Exa*mple", except that "Exa+mple" does not match "Exmple".

.* (any string) Meta-characters can be combined in many ways. The most common combination is perhaps ".*", which matches any string. Since ``.'' matches any character, and ``*'' modifies the preceding character to match zero or more repeats, ``.*'' matches zero or more repeats of any character, which is to say that it matches any string (and *not* just the strings of repeating characters such as "aaaaa"). The regular expression ``Exa.*m.*ple'' matches all strings which start with "Exa", have an "m" somewhere in the middle, and end with "ple", such as "Example", "Exa@#$mple", "Exa28m(bla)ple", etc.

() (grouping) Grouping is used to make a meta-character apply to more than one preceding character, so that the expression ``E(xa)+mple'' matches the strings "Example", "Exaxample", "Exaxaxample", etc.

\ (escape) Escapes are used to remove the "magic" from one of the meta-characters, e.g. to make a * represent the character "*" rather than the "zero or more" qualifier. So the regular expression ``Ex*mple'' with escaped "*" only matches the string

"Ex*mple", while the expression `"Ex*mple"` without the escape matches the strings "Emple", "Exmple", "Exxmple", etc.

Using these meta-characters, we can construct a regular expression which matches all positive or negative integers and floating point numbers

```
[+,-]?[0-9]+(\.[0-9]*)?
```

The initial part `[+,-]?` will match a leading positive or negative sign, but the `?` means that numbers which don't have a sign are also matched. `[0-9]+` matches one or more occurrences of any digit — this will match numbers before the period. The next part, which matches the period and any digits in the fractional part, is enclosed in `()?`, meaning that the part is optional. This is so that integers will be matched. The period is escaped so that it represents the character "." and not the meta-character "any character". Finally, there can be zero or more digits after the period. This expression will match the string "1" (first part `[+,-]?` is absent, one digit matches the second part, and the third part is absent. It will also match "-562", "0.34", "+34.12", etc. Note that it will match "23.", since the fractional part matches zero or more (*) digits, but it will not match ".513" because the part preceding the period matches one or more (+) digits.

This outline only shows a small fraction of what can be done with regular expression. For a full description of the capabilities of the Java regular expression library, see Sun Microsystems (2004), class `java.util.regex.Pattern`.

# Appendix D

# Exogenous neural input filters

As part of NeuroGene we have developed a set of modules which generate simulated exogenous neural activity. Exogenous in this context means originating in neural tissues which are not part of the NeuroGene simulation. A modular approach has been chosen, where different modules can be combined in various ways to produce a wide range of neural input.

Modules come in two types, sources and filters. Sources generates neural activity, while filters alters the neural activity originating in sources or coming through other filters. By combining filters, many varied forms of neural activity patterns can be generated. Any number of other filters can receive activity from a given filter or source. Some filters are created to incorporate neural activity originating from multiple other filters or sources. It is therefore possible to create exogenous activity systems in which filters and sources form the topology of a directed acyclic graph (DAG).

Implementationally, this system is based on a polling model, where simulated neurons will retrieve their activation from the exogenous neural activity system. This means that each time a neuron computes its activity level, it polls the exogenous activity source for the current activity level coming into the cell during the current simulation time step. This implementation is well suited to the discrete-time neural models used in NeuroGene.

Some sources and filters process neural activity based on the location in 3D space of the cell which is receiving the activity. This allows for the design of exogenous input systems which are spatially coherent, i.e., where adjacent neurons tend to receive similar neural activity. This property is used in the generation of simulated visual images.

In the following, the different types of neural activity sources are described, followed by various categories of filters. At the end of the appendix is a detailed description of the exogenous source used to generate simulated visual information in simulation of refinement and ocular dominance column formation presented in chapter 5.

## D.1 Neural activity sources

The following neural activity sources generate neural activity according to simple rules. The simple activity patterns generated by these sources can be modified using filters to create more interesting neural activity patterns.

ConstantValueSource The source is initialized with a real value v. This value $v$ is returned each time the source is queried about the activity at some location. While not very useful by itself, when used in combinations with the additive and multiplicative filters (see below), a wide range of useful configurations may be created.

RandomValueSource This source returns random values from some probability distribution. Available distributions include uniform between two values *lowerBound* and *upperBound*, Gaussian with a given mean and standard deviation, and binomial which returns some value *high* with probability $p \in [0, 1]$ and some other value *low* with the probability $(1 - p)$. Note that this source will *not* return the same value if queried repeatedly about the same location in the same time step. To achieve this normally desired property, the random source must be wrapped with a caching filter, see below.

TagFileActivitySource This source reads data from a file in comma-separated values format (a file format easily generated using common spread sheet applications). Each cell which is to receive neural input from this input file tagged with a string, and the cell will receive data from the column in the spread sheet file with the corresponding string as column header. The string tag need not be identical to the cell name (see appendix C), and more than one cell may share the same tag, which means that they will all receive identical neural input from the input file.

## D.2 Transform location

As outlined above, many filters process neural activity according to the location of the neuron which is going to receive that information. The following filters applies transformations of different kinds to this location information.

TransformLocationFilter This filter changes the location of the query. The filter supports the common linear transformations rotation, uniform scale, non-uniform along each of the three principal spatial axes (which can also be used to implement reflections), translation and shear. For example, a non-uniform scaling filter which scales the $x$, $y$ and $z$ coordinates by the factors 1, $-1$ and 0 respectively (representing a mirror operation through the XZ-plane and a collapse of the Z coordinate into the XY plane) will, when queried about the neural activity arriving at some location $[x, y, z]$ query the underlying source about the location $[x, -y, 0]$ in stead.

WrapLocationFilter This filter ensures that all queries fall within a given area of 3D space, typically the simulation space. Some filters (notably the Gaussian filter) may lead to queries about neural activity at locations that fall outside the simulation space. This filter may be used to wrap such queries so they fall inside the simulation space. If the simulation space extends from $x_{min}$ to $x_{max}$ along the $x$ dimension, then a query to this filter with the $x$ coordinate $x_q < x_{min}$ or $x_q \geq x_{max}$ will be converted to the location

$$x'_q = x_q + n(x_{max} - x_{min})$$

where $n$ is a positive or negative integer such that $x_{min} \leq x'_q < x_{max}$. The $y$ and $z$ coordinates are converted in the same way. The filter may be initialized to use arbitrary values for $x_{min}$, $x_{max}$, $y_{min}$, etc., or may be set to automatically use the extent of the simulation space.

UnPerturbLocationFilter This filter is initialized with the definition of an infinite grid of points in 3D space, $G = [x_0 + Ix_1, y_0 + Jy_1, z_0 + Kz_1]$, $I$, $J$ and $K$ integers. When the filter is queried about the activity at some general location $L = [x, y, z]$ it queries the underlying source about the location which is closest to $L$ but which is also in $G$. This filter can be used to feed activity from locations that fall on a regular grid to cells whose locations do not fall on a regular grid.

## D.3 Transform activity

The following filters apply transformations to the actual neural activity values. Some integrate neural activity from multiple sources and/or filters, allowing activity from different filters or sources to be combined.

AdditiveFilter This filter takes the activity from two other source filters, and returns the sum of the two as resulting activity. The contributions from the two filters are scaled by two independent scalar factors.

MultiplicativeFilter This filter takes the activity from two other source filters, and returns the product of the two as resulting activity.

RandomFlipFilter This filter takes the activity $v$ from an underlying source filter, and returns the value $(1 - v)$ with some probability $p \in [0, 1]$ and the value $v$ with the probability $(1 - p)$. Just like the random value source above, this filter will *not* return the same value if queried repeatedly about the same location in the same time step. Again a caching filter should be used to give consistency within simulation each time step.

StrobeFilter This filter it written to simulate the effect of strobe lighting sometimes used in animal rearing experiments. The effect of the strobe light is to rapidly shift the visual experience between a light and dark domain. The filter retrieves neural activity from two underlying source filters called *light* and *dark*. These filters may be simple constant value sources with values 1 and 0, respectively, or they may be assigned to arbitrary filter sources built from other filters listed here. The filter returns the neural activity from the *light* source for $N_{light}$ time steps, then it returns activity from the *dark* source for $N_{dark}$ time steps, before going back again to the *light* filter.

GaussianFilter This filter does a Gaussian filtering (in 1D, 2D or 3D) over the activation returned by some other filter source. When this filter is queried about the activity at some location $L = [x, y, z]$, it queries the underlying filter source at several points close $L$, and scales these values by a 1D, 2D or 3D Gaussian filter core of the form

$$G(x,y,z) = \exp\left(-\frac{x^2}{2s_x^2}\right)\exp\left(-\frac{y^2}{2s_y^2}\right)\exp\left(-\frac{z^2}{2s_z^2}\right)$$

centered at $L$. The filter core is normalized so that it sums to 1 after discretization. In order to define the location of "several points close $L$", a grid is defines just like the one defined in the case of UnPerturbLocationFilter above.

## D.4 Transform timing

The following filters alters the timing of the generation of neural activity. They can be used to introduce delays in the transmission of neural activity, or to combine current and previous neural activity levels. Note that these filters (with the exception of the caching filter) will not work properly if cells move, since the same cell will not have the same location in each time step.

The most commonly used of these filters is the CachingFilter, which needs to be used in conjunction with all stochastic filters and sources in order to ensure that for a given 3D location in space, a constant value is returned within the same time step.

CachingFilter Each time this filter is queried about the activity at some location $L$, it queries some underlying filter source. It retains the value in a lookup table and returns it. On subsequent queries about the same location $L$, the value is retrieved from the lookup table and returned, and the underlying source is not queried. At the end of each simulation time step, the lookup table is cleared, so that values for the subsequent time step will come from the underlying source filter. This filter has two important uses: One is to improve performance with filters which are computationally expensive. The second is to give consistency to stochastic sources and filters.

TimeDelayFilter This filter causes delays the neural activity by one or more time steps. The filter is initialized with some non-zero parameter $N \geq 1$. In time step $T$, the filter will return activity that was retrieved from the underlying filter source in time step $T - N$. At the same time, the underlying filter is queried for the value for time step $T$, this value is stored in the filter, and will be returned in time step $T + N$. For the first $N$ time steps of the simulation this filter returns zero for all activity queries.

TimeDilateFilter This filter stretches the time domain. The filter is initialized with some parameter $N > 1$. When the filter is queried in some time step $T$ for which the modulus $\mod(T, N) = 0$, then the filter will return activity retrieved from the underlying filter source. It will also store this activity in a local lookup table. In time steps for which $\mod(T, N) \neq 0$, the filter returns the value from the lookup table for the same location. This means that the underlying filter is only queried every $N$ time steps, and the same values are returned for each of the subsequent $(N - 1)$ time steps as well.

RecurrentFilter This filter is similar to the TimeDelayFilter, in that it takes a parameter $N \geq 1$, and stores activity for the last $N$ time steps. However, it stores the output activity computed by the filter itself in an earlier time step, and uses that to compute the activation level in later time steps. In time step $T$, the output $O(T)$ of this filter is computed from the input from the underlying filter source in the same time step, $I(T)$, and the output from the recurrent filter itself in an earlier timestep, $O(T - N)$, to give

$$O(T) = (1 - w)I(T) + wO(T - N)$$

The factor $w \in [0, 1]$ determines the relative weight of the recurrent data and new data. Note that the value of $N$ will in most cases be 1.

## D.5 Neural activity sinks

Activity sinks records the current neural activity level of cells. Currently the system only supports saving the activity to a file, but the system may be extended to support actuators, letting the simulated organism interact and modify its environment.

TagFileActivitySink This activity sink saves neural activity to a file is CSV (comma separated values) format. It acts as a compliment to the TagFileActivitySource

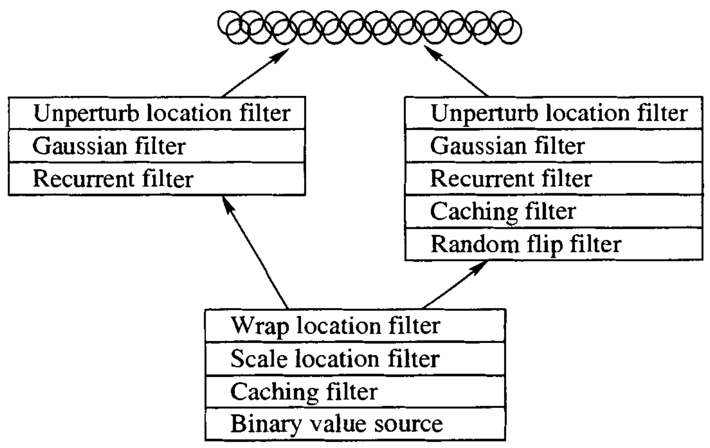


Figure D.1: The complete exogenous filter assembly used to simulate the visual stimuli of the ocular dominance/refinement experiments.

described above, and the file formats are generally compatible. Incompatibilities arise when the set of cells whose activity is saved to file changes during the simulation.

## D.6 Complete example

Figure D.1 shows the entire filter assembly used to generate simulated visual stimuli with well-defined inter-ocular correlation. The filter stack is queried from the cells at the top, using the location in 3D space as the key to determine the value of the generated neural activation.

The first filter is then the UnperturbLocationFilter. This filter ensures that even if the cells do not fall on a regular grid of points in 3D space, the underlying filter will be queried only about points on such a regular grid. This filter makes it possible to stagger the RGCs by small random amounts.

The Gaussian filter has the effect of smoothing out the activity coming from other filters below. In order to compute the activation at some point $[x, y, z]$ the filter needs to query the underlying filter at several points arranged in a regular grid which includes the point $[x, y, z]$. Furthermore, when queried about activation at an adjacent point, e.g. $[x, y, z+\Delta z]$, the queries to the underlying filter needs to be made to the same grid points as those for the earlier query — if not, the values returned from the Gaussian filter for adjacent points will be unrelated, which means the data is not being properly filtered. To achieve this goal, the Gaussian filter uses the same grid as the UnperturbLocationFilter do determine the locations around each query point $[x, y, z]$ at which to query the underlying filter.

The recurrent filter creates temporal coherence in the data by returning the sum of the activation received from the underlying filter in time steps $i$ and that returned by the recurrent filter itself in time step $i - 1$, i.e. $N = 1$. The recurrent weight $w$ as described above is 0.667.

In eyel (on the right in figure D.1) a caching filter is used to retain the activation level from the underlying filter for one time step. Caching filters must be used in conjunction with all stochastic filters to ensure that all calls to the filter with the same location within the same time step returns the same value.

The random flip filter introduces a well defined inter-ocular correlation of neural activity. If the activity level this filter receives from the underlying filter is $x$, then the filter returns $1 - x$ with some probability $p$ and returns the value $x$ unchanged with the probability $1 - p$. $p = 0$ thus creates identical input to both eyes, $p = 1$ creates anti-correlated input to both eyes, and $p = 0.5$ creates non-correlated input to both eyes. Note that because of the Gaussian filtering, the probability of flip at this filter is not going to be the same as the probability of corresponding RGCs having different firing state. However, an increase in $p$ will always result in a decrease in the inter-ocular correlation.

The wrap location filter adjusts for the fact that the Gaussian filters may generate queries that fall outside the simulation space. The filter wraps such queries around so they fall within the simulation space, see above. This has the effect of connecting the edges of the retina similar to how connections where used in the endogenous neural activity experiment, see figure 5.20.

The scale location filter is used to project all queries onto the XY plane. This means that the two layers of retinal ganglion cells may be places at different z coordinates, while still receiving related neural activity. For example, an RGC in one eye might have the coordinates $[x, y, z]$ and another RGC in the other eye might have the coordinates $[x, y, z + \Delta z]$. As these two cells retrieve their neural activation inputs from the filter assemblies, their locations will both be converted to $[x, y, 0]$, which means they will both get the same activity value from the underlying filter. However, these values will subsequently be subject to the different filter stacks, giving rise to different but related values in the end.

The caching filter is again used to make sure that all queries about the same point in space in the same time step returns the same value. Finally, the binary value source returns a value of 1.0 with a probability of 0.5, and a value of 0.0 with probability of 0.5.

## D.7 Conclusion

Using a modular approach, we have been able to create complex simulated exogenous neural activity patterns to a pre-existing specification. Most of the modules are simple, and additional modules can be added to the system relatively easily.

# Appendix E

# JavaBean coding standard

JavaBeans are Java classes which are written according to a coding standard, as described below. Classes written according to this standard may be manipulated in a standardized way using Java's reflection mechanism (Sun Microsystems, 2004). A number of powerful tools exist for instantiating, altering, inspecting and using JavaBean classes, including saving instances to file and transmitting them over a network connection.

All JavaBean classes must have a default constructor, i.e. a constructor which does not take any parameters. The classes may also have other constructors. This means that there is a standard way of creating new instances of any bean class.

The properties of JavaBean classes are accessed through accessor functions, called *getter* (for functions to access the current value of the property) and *setters* (for functions to change the value of the property). The accessor functions are named according to the property they access. As an example, for a property called *age* of type *int*, the getter and setter functions should have the signatures (respectively)

```
public int getAge()
public void setAge(int newAge)
```

If the data type of some property *boring* is *boolean*, the getter may be called either *isBoring()* or *getBoring()*. If both functions exist, then the former is used. In this way JavaBean tools can discern the name and data type of all properties from the signature of the accessor functions.

The NeuroGene system contains a module called a "property sheet" (`org.neurogene.-gui.propertysheet.PropertySheet`), which takes an instance of any JavaBean class, and displays the values of all the instance's properties. The property sheet allows the values of all properties to be altered, and automatically updates the JavaBean instance accordingly. This functionality is used in conjunction with secondary views (see section 3.3.3), but can easily be extended to other areas. Writing these classes according to the JavaBean standard also makes it possible to save view instances to file using already existing XLM-based tools, see section 3.5.

There are a number of methods in the JavaBean standard used in conjunction with the Observable pattern (see section 3.2.1) which allows a tool to monitor and even veto changes in the properties of a JavaBean instances. It is possible to register and deregister *PropertyChangeListener* instances with a JavaBean instance using functions which always have the signatures

```
public void addPropertyChangeListener(PropertyChangeListener lstnr)
public void removePropertyChangeListener(PropertyChangeListener lstnr)
```

All registered listeners are notified whenever a property of the bean is altered. It is also possible to register a *VetoableChangeListener* with a JavaBean instance. It is also notified of changes just like the *PropertyChangeListener*, however, the listener may veto the change by throwing an exception.

Together these standardized facilities allows the design of powerful utilities for interacting with Java classes that are written according to the JavaBean coding standard.

# Appendix F

# Solution to the diffusion equation

## F.1 General solution

Diffusion in a 1D space is described by the ordinary differential equation (Kreyszig, 1988, p. 661)

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} \tag{F.1}$$

where $u(x,t)$ is the concentration of some compound as function of time and space, and $D$ is the coefficient of diffusion. This important equation is treated in most general textbook on partial differential equations, it describes diffusion as well as heat flow and electrostatic potentials.

In three dimensions this becomes

$$\frac{\partial u}{\partial t} = D\nabla^2 u = D\left[\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right]$$

Separation of variables using

$$u(x,y,z,t) = X(x)Y(y)Z(z)T(t)$$

gives

$$\frac{\partial T}{T\partial t} = D\left[\frac{\partial^2 X}{X\partial x^2} + \frac{\partial^2 Y}{Y\partial y^2} + \frac{\partial^2 Z}{Z\partial z^2}\right] \tag{F.2}$$

where the function parameters have been dropped for brevity. Since these terms are functions of different free variables, they must all be constant. We assume for the moment that these constants are all negative.

$$\frac{\partial^2 X}{X\partial x^2} = -p_x^2$$
$$\frac{\partial^2 Y}{Y\partial y^2} = -p_y^2$$
$$\frac{\partial^2 Z}{Z\partial z^2} = -p_z^2$$
$$\frac{\partial T}{T\partial t} = -Dp^2$$

It follows from (F.2) that

$$p_x^2 + p_y^2 + p_z^2 = p^2$$

Solving the spatial derivatives, we get

$$\frac{\partial^2 X}{\partial x^2} + p_x^2 X = 0$$

which has the general solution

$$X(x) = A\cos(p_x x) + B\sin(p_x x) \tag{F.3}$$

The alternative equation

$$\frac{\partial^2 X}{\partial x^2} - p_x^2 X = 0$$

(with positive constant $p^2$) has the solution

$$X(x) = \exp(px)$$

which represents an exponential growth. This solution has no physical meaning and may be excluded from the following analysis. The above assumption regarding the sign of the terms in equation F.2 is therefore justified.

## F.2 Open simulation space

Assume a rectilinear 3D space of dimensions $L_x$, $L_y$, $L_z$ along the $x$, $y$ and $z$ directions. Given the boundary condition that $u$ is zero at the boundaries of this space, i.e. $u(0, y, z, t) = u(L_x, y, z, t) = u(x, 0, z, t) = \ldots = 0$. This corresponds to a situation in which the diffusing compound is free to leave the simulation space, and the concentration of the compound outside the simulation space is zero.

### F.2.1 The spatial components

Using the boundary conditions that $X(0) = 0$ and $X(L_x) = 0$, we get that

$$X(0) = A_x \cos(0) + B_x \sin(0) = A = 0$$

and

$$X(L_x) = B_x \sin(p_x L_x) = 0$$

which has the non-trivial solution for $B \neq 0$

$$\begin{aligned} \sin(p_x L_x) &= 0 \\ p_x L_x &= n\pi \end{aligned}$$

where $n$ is a positive whole number. This gives the solution

$$X_n(x) = B_{xn} \sin\left(\frac{n\pi x}{L_x}\right). \tag{F.4}$$

Solving for $Y(y)$and $Z(z)$ in the same way we get

$$\begin{aligned} p_y &= \frac{n\pi}{L_y} \\ p_z &= \frac{n\pi}{L_z} \\ p_n^2 &= n^2\pi^2\left(\frac{1}{L_x^2} + \frac{1}{L_y^2} + \frac{1}{L_z^2}\right) \end{aligned}$$

### F.2.2 The temporal component

This solution for the spatial components of $u$ allows us to solve for the temporal component.

$$\frac{\partial T}{\partial t} = -Dp_n^2 T$$

which has the solution

$$T_n(t) = B_{tn} \exp\left(-Dp_n^2 t\right) \tag{F.5}$$

Combining the constant terms $B_n$ from the four different solutions into a single constant, we get the following solution in three spatial dimensions and time:

$$u_n(x, y, z, t) = B_n \sin\left(\frac{n\pi x}{L_x}\right) \sin\left(\frac{n\pi y}{L_y}\right) \sin\left(\frac{n\pi z}{L_z}\right) \exp\left(-Dp_n^2 t\right)$$

Expressions of this form are all solutions to the diffusion equation. The general solution is taken to be the infinite sum

$$u(x,y,z,t) = \sum_{n=1}^{\infty} u_n(x,y,z,t) \tag{F.6}$$

### F.2.3 Computing the coefficients

The coefficients $B_n$ are determined from the initial condition at $t = 0$, at which point the concentration is defined by some function $f(x,y,z)$ over the spatial variables only:

$$u(x,y,z,0) = f(x,y,z)$$

To simplify, the solution to finding the coefficients $B_n$ will be shown for the case with two spatial dimensions only:

$$f(x,y) = u(x,y,0) = \sum_{n=1}^{\infty} u_n(x,y,0) = \sum_{n=1}^{\infty} B_n \sin\left(\frac{n\pi x}{L_x}\right) \sin\left(\frac{n\pi y}{L_y}\right) \tag{F.7}$$

Multiplying on either side with $\sin\left(\frac{m\pi x}{L_x}\right)\sin\left(\frac{m\pi y}{L_y}\right)$ we get

$$f(x,y)\sin\left(\frac{m\pi x}{L_x}\right)\sin\left(\frac{m\pi y}{L_y}\right) = \sum_{n=1}^{\infty} B_n \sin\left(\frac{n\pi x}{L_x}\right)\sin\left(\frac{m\pi x}{L_x}\right)\sin\left(\frac{n\pi y}{L_y}\right)\sin\left(\frac{m\pi y}{L_y}\right)$$

Using the identity

$$\sin U \sin V = \frac{1}{2}\left(\cos(U-V) - \cos(U+V)\right) \tag{F.8}$$

twice, we get

$$\begin{aligned} f(x,y)\sin\left(\frac{m\pi x}{L_x}\right)\sin\left(\frac{m\pi y}{L_y}\right) &= \frac{1}{4}\sum_{n=1}^{\infty} B_n \left(\cos\left(\frac{(n-m)\pi x}{L_x}\right) - \cos\left(\frac{(n+m)\pi x}{L_x}\right)\right) \\ &\quad \times \left(\cos\left(\frac{(n-m)\pi y}{L_y}\right) - \cos\left(\frac{(n+m)\pi y}{L_y}\right)\right) \end{aligned}$$

Integrating both sides

$$\begin{aligned} &\int_0^{L_x}\int_0^{L_y} f(x,y)\sin\left(\frac{m\pi x}{L_x}\right)\sin\left(\frac{m\pi y}{L_y}\right) dy\, dx \\ &= \int_0^{L_x}\int_0^{L_y} \frac{1}{4}\sum_{n=1}^{\infty} B_n \left(\cos\left(\frac{(n-m)\pi x}{L_x}\right) - \cos\left(\frac{(n+m)\pi x}{L_x}\right)\right) \\ &\qquad \times \left(\cos\left(\frac{(n-m)\pi y}{L_y}\right) - \cos\left(\frac{(n+m)\pi y}{L_y}\right)\right) dy\, dx \end{aligned}$$

Since $\cos(x)$ is an *even* function, i.e. $\cos(-x) = \cos(x)$, the integrand above is also even. For all even functions $f(x)$ it holds that

$$\int_0^L f(x)dx = \frac{1}{2}\int_{-L}^{L} f(x)dx \tag{F.9}$$

From this we get

$$\begin{aligned}
&\int_0^{L_x}\int_0^{L_y} f(x,y)\sin\left(\frac{m\pi x}{L_x}\right)\sin\left(\frac{m\pi y}{L_y}\right)dy\,dx \\
&= \frac{1}{2}\int_{-L_x}^{L_x}\frac{1}{2}\int_{L_y}^{L_y}\frac{1}{4}\sum_{n=1}^{\infty} B_n\left(\cos\left(\frac{(n-m)\pi x}{L_x}\right)-\cos\left(\frac{(n+m)\pi x}{L_x}\right)\right) \\
&\qquad \times\left(\cos\left(\frac{(n-m)\pi y}{L_y}\right)-\cos\left(\frac{(n+m)\pi y}{L_y}\right)\right)dy\,dx
\end{aligned}$$

Since

$$\int_{-L}^{L}\cos\left(\frac{n\pi x}{L}\right)dx = 0 \tag{F.10}$$

for all $n \neq 0$, all terms of the sum vanish except for those where $m = n$:

$$\begin{aligned}
&\int_0^{L_x}\int_0^{L_y} f(x,y)\sin\left(\frac{n\pi x}{L_x}\right)\sin\left(\frac{n\pi y}{L_y}\right)dy\,dx \\
&= \frac{1}{16}B_n\int_{-L_x}^{L_x}\int_{-L_y}^{L_y}\left(1-\cos\left(\frac{2n\pi x}{L_x}\right)\right)\left(1-\cos\left(\frac{2n\pi y}{L_y}\right)\right)dy\,dx \\
&= \frac{1}{16}B_n\int_{-L_x}^{L_x}\int_{-L_y}^{L_y}dy\,dx = \frac{1}{16}B_n(4L_xL_y) \\
&= \frac{1}{4}B_nL_xL_y
\end{aligned}$$

This gives the following solution to $B_n$ in the 2D case

$$B_n = \frac{4}{L_xL_y}\int_0^{L_x}\int_0^{L_y} f(x,y)\sin\left(\frac{n\pi x}{L_x}\right)\sin\left(\frac{n\pi y}{L_y}\right)dy\,dx$$

which can be easily extended to the 3D case

$$B_n = \frac{8}{L_xL_yL_z}\int_0^{L_x}\int_0^{L_y}\int_0^{L_z} f(x,y,z)\sin\left(\frac{n\pi x}{L_x}\right)\sin\left(\frac{n\pi y}{L_y}\right)\sin\left(\frac{n\pi z}{L_z}\right)dz\,dy\,dx \tag{F.11}$$

## F.3 Closed simulation space

Assuming the same rectilinear space as before, but now the spatial partial derivatives of $u$ are zero at the boundaries. This corresponds to diffusion within a closed space. Since there is no diffusion across the edges of this space, there will be no concentration gradient at the edges either. This gives the following boundary conditions, one for each of the eight faces of the simulation space

$$\begin{aligned}
\frac{\partial u(0,y,z,t)}{\partial x} &= \frac{dX(0)}{dx} = 0 \\
\frac{\partial u(L_x,y,z,t)}{\partial x} &= \frac{dX(L_x)}{dx} = 0 \\
\frac{\partial u(x,0,z,t)}{\partial y} &= \frac{dY(0)}{dy} = 0 \\
&\dots \\
\frac{\partial u(x,y,L_z,t)}{\partial z} &= \frac{dZ(L_z)}{dz} = 0
\end{aligned}$$

### F.3.1 Solution in 1D

Initially I will show the solution to this problem with one spatial dimension only, where the separation of variables gives

$$u(x,t) = X(x)T(t)$$

The same general spatial solution as before holds in this case

$$X(x) = A\cos(p_x x) + B\sin(p_x x)$$

Since the boundary conditions are given in terms of $\frac{dX(x)}{dx}$, the solution is differentiated

$$u_x(x) = \frac{dX(x)}{dx} = -A\sin(p_x x) + B\cos(p_x x)$$

$u_x(0) = 0$ and $p_x \neq 0$ gives $B = 0$. The non-trivial solution of $u_x(L_x) = 0$ with $A \neq 0$ gives $p_x = \frac{n\pi}{L_x}$ as before. This gives

$$X_n(x) = A_{nx}\cos\left(\frac{n\pi x}{L_x}\right) \qquad \text{(F.12)}$$

for integer $n \geq 0$.

### F.3.2 The temporal component

The temporal component has the same solution as before

$$\begin{aligned} T_n(t) &= A_{nt}\exp\left(-Dp_n^2 t\right) \\ p_n &= \frac{n\pi}{L_x} \end{aligned}$$

Combining the spatial and temporal components

$$\begin{aligned} u_n(x,t) &= X_n(x)T_n(t) \\ &= A_{nx}\cos\left(\frac{n\pi x}{L_x}\right)A_{nt}\exp\left(-D\left(\frac{n\pi}{L_x}\right)^2 t\right) \\ &= A_n\cos\left(\frac{n\pi x}{L_x}\right)\exp\left(-D\left(\frac{n\pi}{L_x}\right)^2 t\right) \end{aligned}$$

This gives the general solution

$$\begin{aligned} u(x,t) &= \sum_{n=0}^{\infty} u_n(x,t) \\ &= \sum_{n=0}^{\infty} A_n\cos\left(\frac{n\pi x}{L_x}\right)\exp\left(-D\left(\frac{n\pi}{L_x}\right)^2 t\right) \\ &= A_0 + \sum_{n=1}^{\infty} A_n\cos\left(\frac{n\pi x}{L_x}\right)\exp\left(-D\left(\frac{n\pi}{L_x}\right)^2 t\right) \end{aligned}$$

### F.3.3 Computing the coefficients

The coefficients are computed from the initial condition

$$u(x,0) = f(x)$$

which gives

$$f(x) = \sum_{n=0}^{\infty} u_n(x,0) = A_0 + \sum_{n=1}^{\infty} A_{nx}\cos\left(\frac{n\pi x}{L_x}\right).$$

Multiplying on either side with $\cos\left(\frac{m\pi x}{L_x}\right)$ gives

$$f(x)\cos\left(\frac{m\pi x}{L_x}\right) = A_0\cos\left(\frac{m\pi x}{L_x}\right) + \sum_{n=1}^{\infty} A_{nx}\cos\left(\frac{n\pi x}{L_x}\right)\cos\left(\frac{m\pi x}{L_x}\right)$$

Using the identity

$$\cos U \cos V = \frac{1}{2}\left(\cos\left(U+V\right)+\cos\left(U-V\right)\right) \tag{F.13}$$

we get

$$\begin{aligned} f(x)\cos\left(\frac{m\pi x}{L_x}\right) &= A_0\cos\left(\frac{m\pi x}{L_x}\right) \\ &+\frac{1}{2}\sum_{n=1}^{\infty} A_{nx}\left(\cos\left(\frac{(n+m)\pi x}{L_x}\right)+\cos\left(\frac{(n-m)\pi x}{L_x}\right)\right). \end{aligned}$$

Integrating on both sides gives

$$\begin{aligned} \int_0^{L_x} f(x)\cos\left(\frac{m\pi x}{L_x}\right)dx &= \int_0^{L_x} A_0\cos\left(\frac{m\pi x}{L_x}\right)dx \\ &+ \frac{1}{2}\int_0^{L_x}\sum_{n=1}^{\infty} A_{nx}\left(\cos\left(\frac{(n+m)\pi x}{L_x}\right)+\cos\left(\frac{(n-m)\pi x}{L_x}\right)\right)dx. \end{aligned}$$

Since cosine is an even function, the limits on the integrals can be replaced:

$$\begin{aligned} \int_0^{L_x} f(x)\cos\left(\frac{m\pi x}{L_x}\right)dx &= \frac{1}{2}\int_{-L_x}^{L_x} A_0\cos\left(\frac{m\pi x}{L_x}\right)dx \\ &+ \frac{1}{4}\int_{-L_x}^{L_x}\sum_{n=1}^{\infty} A_{nx}\left(\cos\left(\frac{(n+m)\pi x}{L_x}\right)+\cos\left(\frac{(n-m)\pi x}{L_x}\right)\right)dx. \end{aligned}$$

The first integral vanishes except for when $m = 0$, in which case the second integral also vanishes. This offers a solution to $A_0$:

$$\int_0^{L_x} f(x)dx = \frac{1}{2}\int_{-L_x}^{L_x} A_0 dx = A_0 L_x$$

In the second integral, all terms of the sum vanish except for when $m = n$, in which case the first integral also vanishes, giving a solution to $A_{nx}$:

$$\int_0^{L_x} f(x)\cos\left(\frac{n\pi x}{L_x}\right)dx = \frac{1}{4}\int_{-L_x}^{L_x} A_{nx}dx = \frac{1}{2}A_{nx}L_x$$

In summary, the coefficients can thus be determined from the following expressions

$$\begin{aligned} A_0 &= \frac{1}{L_x}\int_0^{L_x} f(x)dx \\ A_{nx} &= \frac{2}{L_x}\int_0^{L_x} f(x)\cos\left(\frac{n\pi x}{L_x}\right)dx \end{aligned}$$

### F.3.4 Solution in 2D

The solution for two spatial dimension will now be given. It will be possible to extend this solution to three spatial dimensions as well. From the separation of $u(x,y,t)$ into three components $X(x)$, $Y(y)$ (which has the same form as $X(x)$) and $T(t)$ we get the general solution in 2D

$$\begin{aligned}
X_n(x) &= A_{nx}\cos\left(\frac{n\pi x}{L_x}\right) \\
Y_n(y) &= A_{ny}\cos\left(\frac{n\pi y}{L_y}\right) \\
T_n(t) &= A_{nt}\exp\left(-Dn^2\pi^2\left(\frac{1}{L_x^2}+\frac{1}{L_y^2}\right)t\right) \\
u_n(x,y,t) &= X_n(x)Y_n(y)T_n(t) \\
&= A_n\cos\left(\frac{n\pi x}{L_x}\right)\cos\left(\frac{n\pi y}{L_y}\right)\exp\left(-Dn^2\pi^2\left(\frac{1}{L_x^2}+\frac{1}{L_y^2}\right)t\right)
\end{aligned}$$

Here the constants have been combined so that $A_n = A_{nx}A_{ny}A_{nt}$. The general solution is the infinite sum as before

$$\begin{aligned}
u(x,y,t) &= \sum_{n=0}^{\infty} u_n(x,y,t) \\
&= \sum_{n=0}^{\infty} A_n\cos\left(\frac{n\pi x}{L_x}\right)\cos\left(\frac{n\pi y}{L_y}\right)\exp\left(-Dn^2\pi^2\left(\frac{1}{L_x^2}+\frac{1}{L_y^2}\right)t\right) \\
&= A_0 + \sum_{n=1}^{\infty} A_n\cos\left(\frac{n\pi x}{L_x}\right)\cos\left(\frac{n\pi y}{L_y}\right)\exp\left(-Dn^2\pi^2\left(\frac{1}{L_x^2}+\frac{1}{L_y^2}\right)t\right)
\end{aligned}$$

The initial condition gives the values of the coefficients

$$\begin{aligned}
f(x,y) &= u(x,y,0) \\
&= A_0 + \sum_{n=1}^{\infty} A_n\cos\left(\frac{n\pi x}{L_x}\right)\cos\left(\frac{n\pi y}{L_y}\right)
\end{aligned}$$

Multiplying with $\cos\left(\frac{m\pi x}{L_x}\right)\cos\left(\frac{m\pi y}{L_y}\right)$ on both sides

$$\begin{aligned}
f(x,y)\cos\left(\frac{m\pi x}{L_x}\right)\cos\left(\frac{m\pi y}{L_y}\right) &= A_0\cos\left(\frac{m\pi x}{L_x}\right)\cos\left(\frac{m\pi y}{L_y}\right) \\
&\quad + \sum_{n=1}^{\infty} A_n\cos\left(\frac{n\pi x}{L_x}\right)\cos\left(\frac{m\pi x}{L_x}\right) \\
&\quad \times \cos\left(\frac{n\pi y}{L_y}\right)\cos\left(\frac{m\pi y}{L_y}\right)
\end{aligned}$$

Integrating on both sides gives

$$\int_0^{L_x}\int_0^{L_y} f(x,y)\cos\left(\frac{m\pi x}{L_x}\right)\cos\left(\frac{m\pi y}{L_y}\right)dy\,dx$$

$$\begin{aligned} = &\int_0^{L_x}\int_0^{L_y} A_0\cos\left(\frac{m\pi x}{L_x}\right)\cos\left(\frac{m\pi y}{L_y}\right)dy\,dx \\ &+\int_0^{L_x}\int_0^{L_y}\sum_{n=1}^{\infty} A_n\cos\left(\frac{n\pi x}{L_x}\right)\cos\left(\frac{m\pi x}{L_x}\right)\cos\left(\frac{n\pi y}{L_y}\right)\cos\left(\frac{m\pi y}{L_y}\right)dy\,dx\end{aligned}$$

Using the identity F.13 again

$$\int_0^{L_x}\int_0^{L_y} f(x,y)\cos\left(\frac{m\pi x}{L_x}\right)\cos\left(\frac{m\pi y}{L_y}\right)dy\,dx$$

$$\begin{aligned} = &\int_0^{L_x}\int_0^{L_y} A_0\cos\left(\frac{m\pi x}{L_x}\right)\cos\left(\frac{m\pi y}{L_y}\right)dy\,dx \\ &+\frac{1}{4}\int_0^{L_x}\int_0^{L_y}\sum_{n=1}^{\infty} A_n\left(\cos\left(\frac{(n+m)\pi x}{L_x}\right)+\cos\left(\frac{(n-m)\pi x}{L_x}\right)\right) \\ &\times\left(\cos\left(\frac{(n+m)\pi y}{L_y}\right)+\cos\left(\frac{(n-m)\pi y}{L_y}\right)\right)dy\,dx\end{aligned}$$

Replacing the limits gives

$$\int_0^{L_x}\int_0^{L_y} f(x,y)\cos\left(\frac{m\pi x}{L_x}\right)\cos\left(\frac{m\pi y}{L_y}\right)dy\,dx$$

$$\begin{aligned} = &\frac{1}{4}\int_{-L_x}^{L_x}\int_{-L_y}^{L_y} A_0\cos\left(\frac{m\pi x}{L_x}\right)\cos\left(\frac{m\pi y}{L_y}\right)dy\,dx \\ &+\frac{1}{16}\int_{-L_x}^{L_x}\int_{-L_y}^{L_y}\sum_{n=1}^{\infty} A_n\left(\cos\left(\frac{(n+m)\pi x}{L_x}\right)+\cos\left(\frac{(n-m)\pi x}{L_x}\right)\right) \\ &\times\left(\cos\left(\frac{(n+m)\pi y}{L_y}\right)+\cos\left(\frac{(n-m)\pi y}{L_y}\right)\right)dy\,dx\end{aligned}$$

For $m = 0$ the second integral disappears for all values of $n$ (since $n$ is an integer greater than zero), giving the solution for $A_0$:

$$\int_0^{L_x}\int_0^{L_y} f(x,y)dy\,dx = \frac{1}{4}\int_{-L_x}^{L_x}\int_{-L_y}^{L_y} A_0 dy\,dx = A_0 L_x L_y$$

The second integral vanishes for all values of $m$ different from $n$, giving the solution for $A_n$:

$$\begin{aligned}\int_0^{L_x}\int_0^{L_y} f(x,y)\cos\left(\frac{n\pi x}{L_x}\right)\cos\left(\frac{n\pi y}{L_y}\right)dy\,dx &= \frac{1}{16}\int_{-L_x}^{L_x}\int_{-L_y}^{L_y} A_n dy\,dx \\ &= \frac{1}{4}A_n L_x L_y\end{aligned}$$

The coefficients can be computed from the following expressions

$$\begin{aligned}A_0 &= \frac{1}{L_xL_y}\int_0^{L_x}\int_0^{L_y} f(x,y)dy\,dx \\ A_n &= \frac{4}{L_xL_y}\int_0^{L_x}\int_0^{L_y} f(x,y)\cos\left(\frac{n\pi x}{L_x}\right)\cos\left(\frac{n\pi y}{L_y}\right)dy\,dx\end{aligned}$$

These expressions can easily be extended to 3D as well

$$\begin{aligned}A_0 &= \frac{1}{L_xL_yL_z}\int_0^{L_x}\int_0^{L_y}\int_0^{L_z} f(x,y,z)dz\,dy\,dx \\ A_n &= \frac{8}{L_xL_yL_z}\int_0^{L_x}\int_0^{L_y}\int_0^{L_z} f(x,y,z)\cos\left(\frac{n\pi x}{L_x}\right)\cos\left(\frac{n\pi y}{L_y}\right)\cos\left(\frac{n\pi z}{L_z}\right)dz\,dy\,dx\end{aligned}$$

## F.4 Conclusion

We have given the solutions to the diffusion equation in three spatial dimensions and time for two kinds of boundary conditions. These represent a simulation space which is either open or closed to the diffusion of protein out of the space. These solutions may be used to estimate the accuracy of the simulated diffusion as computed numerically by NeuroGene.

# Appendix G

# Models of topographic projections

In this appendix I summarize earlier work on modeling topographic projections. This work goes back to the important arrow model proposed by Hope et al. (1976), and culminates in a recent model proposed by Yates et al. (2004) as well as the work presented in this thesis.

In the following I will first summarize the models and their properties (section G.1). In section G.2 the degree to which the various models can adapt to changes in the properties of the developing system is estimated. In section G.3 a summary is given of the experiments which each model is able to replicate. Finally, in sections G.4 and G.5 the models by Overton and Arbib (1982) and Yates et al. (2004) are described in detail.

## G.1 Summary of earlier models

The "arrow model" by Hope et al. (1976) makes some very simple assumptions that in our view have stood the test of time to the extent that our model has many elements in common with this early effort. It is assumed that each RGC axon innervating the tectum takes directional (as opposed to absolute location) cues from the tectal gradients. Axons also make comparisons with neighbouring RGC axons, and go through repeated pairwise swaps to implement a 2D version of the "bubble-sort" sorting algorithm to form the topographic projection.

Whitelaw and Cowan (1981) take a very different approach based on a modified form of Hebbian learning, where matching chemical gradients enhances the growth of synaptic weights. This model assumes parallel chemical gradients (i.e. high concentration axons map

| Model | Algorithm |
|---|---|
| Hope | Non-random walk — bubble-sort |
| Whitelaw | Hebbian learning modulated by chemical markers |
| Gierer | Not clear, relies on modeling intracellular signaling within growth cones |
| Overton | Non-random walk determined by inter-axon forces |
| Fraser | Simulated annealing |
| Weber | Eulers method |
| Yates | Axon branching |
| NeuroGene | Simulated growth cones (a form of non-random walk) |

Table G.1: Algorithms used by the various models.

to high concentration tectum). This assumption is problematic in view of more recent data showing the importance of counter-gradients in the formation of topographic maps.

Overton and Arbib (1982) propose an extension of the arrow model. The main motivation is to incorporate a more biologically plausible model of axon growth, by which each RGC axon projects a number of individually migrating branches to the tectum. They also allow for each RGC branch interacting with more than one other branch simultaneously, although the element of pairwise comparisons is retained from the original. It is questionable whether their more specific mode of axon growth has in fact been borne out be more recent research, or whether this model represents a step backward from the conceptually much simpler original.

Gierer (1983) bases his model on the intracellular signaling events that may be occurring within growth cones when these transduce tectal concentration gradients. Like several of the later models, his assumes that the tectal concentration gradient relates absolute (location) information to the invading RGC axons, rather than relative (directional) information. This assumption is also problematic in view of recent data, in particular that of Brown et al. (2000). However, Gierer's approach to understanding topographic map formation through an understanding growth cone function is one that we have also adopted.

The multiple-component model of Fraser and Perkel (1990) incorporates a large number of possible retino-tectal interactions. Using their own labels for these interactions, they are:

*C* A general attraction and adhesion of RGC axons to the tectum. This is similar to the role of the protein retinoTectalGradient in the NeuroGene simulation, see section 5.2.1.

$R$ A general competition among RGC axons for tectal space.

$N$ A neural activity-dependent interaction by which the competition among RGC axons originating in adjacent sites in the retina is reduced. The neural activity is thus not modeled explicitly, but the correlated retinal activity is taken into account. This allows for the investigation of the effects of removing neural activity.

$DV$ and $AP$ (for *Dorso-Ventral* and *Anterior-Posterior*) Specific attraction and adhesion of RGC axons to the area of the tectum to which it projects in an ideal projection. This contribution imposes the topographic ordering on the projection. Note that it is again assumed that the tectum provides absolute location information to the RGC axons.

They found that the map formed best when the strengths of these interactions were in the order $C > R > DV, AP > N$. The maps were constructed by minimizing the sum of these different contributions for all RGC axons. The minimization was carried out using simulated annealing, which is probably not representative of the actual biological mechanism of map formation.

It appears that an important goal in designing the model by Weber et al. (1997) was to be able to replicate the polarity-reversal experiment (Meyer, 1979). The general approach is similar to that of Fraser and Perkel (1990): The model contains contributions representing axon-tectum interactions, representing interactions between protein concentration gradients ($A_{\alpha x}$), as well as axon-axon interactions representing both chemically mediated ($F_{\alpha x}^{int}$) and Hebbian ($F_{E\alpha x}^{act}$) processes. However, the formulation of the model as a whole seems somewhat arbitrary, including the approach of using Eulers method to evaluate the model (see table G.1).

The recent model by Yates et al. (2004) is based on experimental data indicating that the formation of axonal side-branches within the tectum is an important step toward a fully formed topographic map (Yates et al., 2001). This is a fact that is not included in the NeuroGene model represented here. The mechanism used to form the topographic map is very close to what is believed to occur in nature. However, the model is weakened by the fact that is includes a large number of parameters, many of which do not have straightforward biological interpretations. Given the importance of this model, it is outlined in more detail in section G.5 below.

Table G.2 shows a comparison of the features of the models outlined above, including the NeuroGene model of topographic map formation. Willshaw and Malsburg (1976) is not included in this overview, since this model simulates activity-dependent map-formation only.

## G.2 Flexibility of models

Biological systems are characterized by their relative insensitivity to variations in protein concentrations, size and location of organs, etc., as witnessed by e.g. experiments in which surgically or biochemically modifications to developing organism does not perturb the formation of topographic maps, see section 5.2. Computational models of map formation which do not show a similar insensitivity to variations in parameters may be of questionable biological relevance. The models considered here can be compared by how well they adapt to variations in the retinal and tectal concentration gradient shapes, and how they adapt to changes in their own parameters.

The model by Hope et al. (1976) can probably deal with changes in tectal and retinal gradient shapes — however, this was not reported. Whitelaw and Cowan (1981) demonstrate that their model is consistent with a number of different gradient shapes. Presumably any smooth and monotonously increasing or decreasing gradient shape would work. However, the model relies on parallel retinal and tectal gradients, as opposed to the counter-gradients which have been shown to play an important role in the formation of topographic maps. The model is able to replicate the mismatch experiment (see table G.3), which indicates a significant flexibility with respect to gradient shapes.

The model by Gierer (1983) is not dependent on particular gradient shapes, but it does rely on a particular inter-relationship between the retinal and tectal gradient shapes. The model supports a wide range of different gradient shapes, but a particular mathematical relationship between the retinal and tectal gradients must be maintained. The multiple-component model (Fraser and Perkel, 1990) does not explicitly represent retinal or tectal gradient shapes. In stead, each RGC axon binds with slight preference to its termination zone in the tectum, with linearly decreasing binding strength with increasing distance from

| | Hope | Whitelaw | Overton | Gierer | Fraser | Weber | Yates | NeuroGene |
|---|---|---|---|---|---|---|---|---|
| model map dimensionality | 2D | 1D | 1D | 1D | 2D | 2D | 1D | 2D |
| axon-axon pairwise comparisons | ✓ | ✗ | ✓ | ✗ | ✗ | | ✗ | ✓ |
| simple axon-axon repulsion | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ |
| specific axon-axon attraction[a] | ✗ | ✓[b] | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ |
| number of gradients in retina[c] | – | 1 | – | 1/2 | –[d] | 2 | 2 | 1 |
| absolute tectal locations[e] | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ | ✗ | ✗ |
| number of gradients in tectum[c] | – | 1 | – | 1/2 | –[d] | 2 | 2 | 1 |
| concentration *counter*-gradients[f] | – | ✗ | – | – | – | – | ✓ | ✓ |
| independent gradients[g] | ✓ | ✓ | ✓ | ✗ | – | ✓ | ✓ | ✓ |
| RGCs axons alter tectal gradients | ✗ | ✗ | ✗ | ✓[h] | ✗ | ✗ | ✓[i] | ✗ |
| growth cones explicitly modelled | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✓ |
| incorporates neural activity | ✗ | ✓ | ✗ | ✗ | ✓ | ✓ | ✗ | ✓ |
| emergent arbor size upper bound | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ |
| number of parameters[j] | 0 | 6 | 6 | 0 | 8 | 13 | 8 | 1[k] |
| biological paramter interpretations[l] | – | ✗ | ✗ | – | ✓[m] | ✗[n] | ✗ | ✓[m] |
| general level of biological detail | med | low | low | high | low | low | high | high |

[a]Is there an attractive interaction between axons which originate in nearby retinal locations?

[b]Implemented through correlated neural activity of neighbouring retinal cells.

[c]In 2D models, this is the number of gradients in each dimension.

[d]Concentration gradient shapes are not explicitly represented or specified.

[e]✓ indicates that RGC axons have access to their absolute location in the tectum.

[f]✓ means counter-gradients; ✗ means parallel gradients; – means model is not committed to either.

[g]Can gradient shapes in the retina and the tectum be altered independently of one another without disrupting map formation?

[h]Tectal cells are induced through some unspecified process to change their protein concentrations.

[i]The Eph and ephrin carried by RGC axons contributes to the tectal concentratons of the same molecules.

[j]Included are parameters which are not related to gradient shapes. This measure is somewhat subjective.

[k]Includes relative strength of tectal adhesion *vs.* specific retino-tectal interactions only. Parameters of the ODC simulation are excluded.

[l]Do the parameters of the model have clear biological interpretations. This measure is somewhat subjective.

[m]Relates primarily to the relative strengths of various interactions.

[n]While some parameters relate to relative strengths of interactions, several also describe the range of these interactions.

Table G.2: Summary of properties of the map formation of various models.

the termination zone. Unlike the previous models, that by Fraser and Perkel contains several parameters which determine the relative strengths of the various components of the model. According to the paper, some of these parameters can be varied quite widely, however this was not fully explored.

Weber et al. (1997) showed that their model works for a number of different gradient shapes. They state that "... the exact shape of the fibre-tectum interaction function does not really matter, as long as cues for the correct polarity are present and as long as the interaction is not too specific" (Weber et al., 1997, p. 1613). They also state that the values of all parameters of the model can be varied at least by a factor of two without disrupting the map formation. However, the model is highly constrained by the requirement that it model the polarity-reversal experiment. This experiment seems to imply that the specific, attractive fibre-fibre interaction is so strong that it can override the fibre-tectum interaction to give rise to a topographic domain which is inverted with respect to the tectum, stabilized by fibre-fibre interactions only. This experiment is not modeled by any of the other models listed.

The recent model by Yates et al. (2004) defines a large set of parameter values, corresponding to a high-dimensional parameter space. They show that their model gives good results at two locations within this parameter space. They offer no information on whether other parameter values were attempted used, and with what results, or whether the given parameter values are within large domains of parameter space which consistently give rise to valid models. Also, when it comes to the concentration gradient shapes, the model requires that the retinal and tectal gradients be reciprocal. This indicates that the model would not be able to model the mismatch experiment, see table G.3.

The model for topographic map formation presented in this thesis, like the arrow-model (Hope et al., 1976), makes no assumptions about the shapes of the concentration gradients either in the tectum or the retina, except that they be monotonically increasing or decreasing to form a wild-type map. Our model also does not include very many parameters. One important requirement of our model is that the attraction of RGC axons to the tectum is stronger than the interactions that cause map topography to form. This is very similar to the role of the $C$ interaction in the model of Fraser and Perkel (1990).

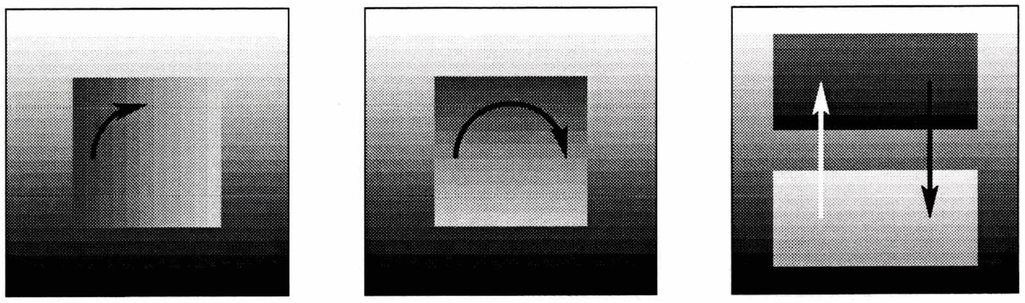


Figure G.1: Three experiments that are replicated by other models, but which have not been attempted by us. The all consist of cutting out a small square or rectangular piece of the tectum, and re-implanting it in a different orientation (left, centre), or cutting out two such pieces and exchanging their locations while maintaining their orientation (right). In all cases, experiments *in vivo* have shown that the RGC axons follow the gradients with altered orientations, giving rise to topographic projections that are distorted in predictable ways. Left: the 90° degree rotation experiment. Centre: The 180° rotation experiment. Right: The translocation experiment.

## G.3 Experiments modeled

Table G.3 shows the experiments that are replicated by the various models listed above. Most of these experiments are described in section 5.2. However, some of the experiments in table G.3 have not been matched by NeuroGene, namely the rotation and translocation experiments. The rotation experiments involve cutting out a small, square piece of the tectum, and re-inserting it after a 90° or 180° rotation (see figure G.1 left and centre, respectively). The translocation experiment involves cutting out two small pieces, and exchanging them without any rotation (figure G.1 right).

## G.4 The model of Overton and Arbib

The model of Overton and Arbib (1982) is of particular interest, since it represents an extension of the arrow model of Hope et al. (1976), to which our model is also related. In this model, each RGC axon projects a number of branches to the tectum, each of which can move independently. For each branch, only the location of the branch end-point is

| | Hope | Whitelaw | Overton | Gierer | Fraser | Weber | Yates | NeuroGene |
|---|---|---|---|---|---|---|---|---|
| wild type | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| rotation 90°/180° | ✓ | ✓[a] | ✗[a] | ✓[a] | ✓[a] | ✓[b] | | ✗ |
| expansion | ✓ | ✓ | ✓ | ✓[c] | ✗[d] | ✓[e] | | ✓ |
| compression | ✓ | ✓ | ✓ | ✓[c] | ✗[d] | ✓ | | ✓ |
| compound eye | | ✓ | ✓ | | | | | ✓ |
| one-and-a-half eye | | | | | | | | ✓ |
| mismatch | | ✓ | ✓ | | | ✓ | | ✓ |
| translocation | ✗ | ✓ | ✗ | | ✗ | ✓[b] | | ✗ |
| ignores small grafts | | | | | ✓ | ✓ | | |
| polarity reversal | | | | | | ✓[f] | | |
| ephrinA knockout | | | | | | | ✓[g] | |
| EphA knockin | | | | | | | ✓[g] | ✓ |
| ODCs | | | | | ✓ | ✓ | | ✓ |
| ODC mono-ocular deprivation | | | | | ✓ | ✓[h] | | |
| ODC width correlation | | | | | ✗ | ✗ | | ✓ |
| nasal projection | | | | | | | | ✓ |
| all use same parameter values | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ✓[i] | ✓ |

[a]Only 180° rotation attempted.

[b]Experiment works when a 1D topographic map is modeled, but breaks down in the 2D case.

[c]Requires that the altered distribution of RGC axons induces changes in the tectal gradients.

[d]Topography of projection is distorted.

[e]but see figure 5f.

[f]The experiment is "brittle", and requires a very strong specific axon-axon attractive force.

[g]In "mouse" only, "chick" not reported, see footnote *i*.

[h]Distorted, see page 1615.

[i]Two set of parameter values are used, one representing chick and one representing mouse. It is not clear what fraction of the parameter space as a whole give rise to valid models.

Table G.3: Table of experiments successfully replicated by the various models. ✓ indicates that the experiment was successful, ✗ indicates that it failed. A blank indicates that the experiment was not reported.

represented in the model. The motion of a particular branch $b$ is given by the weighted sum

$$\overrightarrow{M_b} = a_1\overrightarrow{I_b} + a_2\overrightarrow{E_b} + a_3\overrightarrow{A_b} \tag{G.1}$$

where $a_1$, $a_2$ and $a_3$ are weighting constants. The term $\overrightarrow{A_b}$, the "average influence", represents the force acting on the branch $b$ due to the forces acting on all other branches belonging to the same RGC axon, and is given by the expression

$$\overrightarrow{A_b} = \frac{1}{m}\sum_{k\in F_b}\left(a_4\overrightarrow{I_k} + a_5\overrightarrow{E_k}\right) \tag{G.2}$$

where the set $F_b$ of all branches belonging to the same retinal cell as $b$, $m$ is the number of branches in $F_b$, and $a_4$ and $a_5$ are weighting constants.

Each RGC branch $b$ interacts with all other branches of all other RGCs which fall within a certain radius — call the set of such branches $B_b$. For each branch $k \in B_b$, let $\overrightarrow{U}(b,k)$ be the unit vector parallel to the vector from the retinal location of $b$ to that of $k$, projected onto the tectal plane. It is through this term that the comparison of branches with respect to their retinal points of origin occurs. This gives rise to the following term describing the interaction between retinal branches:

$$\overrightarrow{I_b} = \sum_{k\in B_b} W_d(b,k)\, W_g(b,k)\, \overrightarrow{U}(b,k) \tag{G.3}$$

Finally, the terms $W_d$ and $W_g$ are defined as follows

$$W_d(b,k) = \begin{cases} 1 - d(b,k)/2r & \text{if } d(b,k) < 2r \\ 0 & \text{otherwise} \end{cases} \tag{G.4}$$

$$W_g(b,k) = a_g^n \tag{G.5}$$

Here $W_d$ represents the decrease of the strength of the inter-axon interaction with the distance $d(b,k)$ between the locations of branches $b$ and $k$ in the tectum. $W_g$ is used to take account of the effect of scarring at graft boundaries after tectal grafts reduces the strength of the axon-axon interaction. $0 < a_g < 1$ and $n$ (which is an exponent, not a superscript) is the number of such graft edges intersected by a straight line connecting the tectal locations of $b$ and $k$.

Finally, the term $\overrightarrow{E_b}$ takes account of the effect of graft edges as well as the edges of the tectum.

$$\overrightarrow{E_b} = \sum_{q \in Q} W_d(b,q)\, W_g(b,q)\, \overrightarrow{U}(b,q) \tag{G.6}$$

where $Q$ is the set of tectal and graft edges, and the index $q$ therefore refers to such edges, not RGC axons branches. The term $W_d$ is a linear decay terms similar to the one in equation G.4 above (except that $2r$ is replaced with $r$ throughout), and $W_g$ is also the same as defined in expression G.5.

Compared to the original arrow model, this extended form introduces a large number of parameters ($a_1$ through $a_5$) for which biological interpretations are not obvious.

## G.5 The model of Yates et al.

Yates and coworkers (2004) define the term *totalrepellent* $I_n^T(y)$ for RGC axon $n$ projection to location $y$ in the tectum:

$$I_n^T(y) = \frac{1}{1+\exp(-(M_n(y)-\theta)/\phi)} \tag{G.7}$$

$$M_n(y) = R_n^C L^C(y) + I_n^{AC}(y) + R_n L(y) + I_n^A(y) \tag{G.8}$$

$$I_n^A(y) = \omega \sum_z R_n R_z^C S_z^T(y) \tag{G.9}$$

$$I_n^{AC}(y) = \omega \sum_z R_n^C R_z S_z^T(y) \tag{G.10}$$

where the terms are defined as

| | |
|---|---|
| $R_n$ | The retinal Eph receptor concentration for the axon at location $n$. |
| $L(y)$ | The tectal ephrin ligand concentration at the tectal location $y$. |
| $R_n^C$ | The retinal ephrin ligand concentration for the axon at location $n$. |
| $L^C(y)$ | The tectal Eph receptor concentration at the tectal location $y$. |
| $S_n^T(y)$ | The total arbor size of the axon $n$ at the tectal location $y$. |
| $\theta$, $\phi$, $\omega$ | Scaling factors. |

This gives the following interpretation of the four terms making up expression G.8.

| | |
|---|---|
| $R_n^C L^C(y)$ | The activation of RGC ephrin of axon $n$ from the tectal Eph receptor at location $y$. |
| $I_n^{AC}(y)$ | The activation of RGC ephrin of axon $n$ from the Eph receptor carried by all other RGC axons at location $y$, scaled by $\omega$. |
| $R_n L(y)$ | The activation of RGC Eph receptor of axon $n$ from the tectal ephrin at location $y$. |
| $I_n^A(y)$ | The activation of RGC Eph receptor of axon $n$ from the ephrin receptor carried by all other RGC axons at location $y$, scaled by $\omega$. |

The term $M_n(y)$ increases with increasing repellent interactions between the RGC axon $n$ and the environment at tectal location $y$, as mediated by the receptor function of both EphA and ephrinA. The term *total repellent* $I_n^T(y)$ makes the repellent interaction non-linear: As $M_n(y)$ becomes small, $I_n^T(y)$ approaches $1/(1+\exp(\theta/\phi)) \leq 1$, as $M_n(y)$ becomes large, $I_n^T(y)$ approaches 1. Based on this framework, the probability of branch formation is defined as

$$p_n(y) = \eta(1 - I_n^T(y)) \tag{G.11}$$

where $\eta$ is another scaling factor. Branches retract with the probability $(1 - p_n(y))^2$. All other branches remain unchanged. The primary axon retracts with the probability

$$p_n^r(y) = 1 - \rho p_n(y) \tag{G.12}$$

$$p_n^r(y) = \begin{cases} 1 - \rho p_n(y) & for\, 0 \leq \rho p_n(y) \leq 1 \\ 0 & for\, \rho p_n(y) > 1 \end{cases} \tag{G.13}$$

where $\rho$ is yet another scaling parameter. They then introduce a branch density parameter $D_n(y)$:

$$D_n(y) = (1 - A) + A\left(\sum_{w=y-10}^{y+10} \sigma_z S_n^T(w) \middle/ \sum_{w=1}^{100} S_n^T(w)\right) \tag{G.14}$$

$$A = \frac{\alpha}{1 + \exp\left(-\frac{t-\gamma}{\beta}\right)} \tag{G.15}$$

Where $t$ is the simulation iteration count, $\sigma_z$ is a Gaussian filter core with standard deviation of 10, centred at $y$. The term $A$ determines the influence of the branch density $S_n^T(y)$ on the value of $D_n(y)$. The terms $\alpha$, $\beta$ and $\gamma$ are scaling parameters, and $t$ is the simulation time step counter. The influence of $S_n^T(y)$ on $D_n(y)$ increases as the simulation progresses, with the limits of $A = \alpha/(1 + \exp(\gamma/\beta)) \leq \alpha$ when $t = 0$ and $A = \alpha$ as $t \to \infty$. This gives rise to a modified branching probability:

$$p_n(y) \quad = \quad D_n(y)\eta(1 - I_n^T(y)) \tag{G.16}$$

It is a weakness of this model that it has a large number of parameters which for the most part have no direct biological interpretation. However, the model does appear to show resilience to variations in several of these parameters, as demonstrated in the quite dissimilar chick and mouse simulations. It is not clear from the paper whether these two points in parameter space are unique in their performance of the model, or whether large areas of the parameter space gives good results for the model.

## G.6 Conclusion

Duplicating a wide range of reported experimental observations on the retinotectal projection, and well founded in the known biological facts about this projection, our growth cone based model of topographic map formation represents significant progress toward a full understanding of the biological mechanisms forming this projection.

Appendix H

# A neurotrophic model of synaptic learning

Elliott and Shadbolt (1998a,b, 1999, 2002) have developed a model of synaptic learning based on chemical signaling. This learning rule is formulated in terms of the biological mechanism by which hebbian learning might be implemented. In this model, learning takes place in the presynaptic cell. The postsynaptic cell emits a chemical signal (here called *neurotrophic factor* or NT) when it fires, thereby informing synaptic termini of presynaptic cells of this fact. The affinity to NT of the presynaptic cells depend on their activity. Maximal uptake of NT therefore happens in active presynaptic cells as postsynaptic cells fire. The simulated genes which implement the genetic and biochemical processes outlined below are listed in appendix B.2.3.

## H.1 The learning rule

The learning rule is expressed in equation (2.16) from Elliott and Shadbolt (1999):

$$s_{xi}^{n+1} - s_{xi}^{n} = \varepsilon s_{xi}^{n} \left[ \sum_{y} \Delta_{xy} \left( T_0 + T_1 \frac{\sum_j s_{yj}^n a_j^n}{\sum_j s_{yj}^n} \right) \frac{(a + a_i^n)\, \rho_i^n}{\sum_j s_{xj}^n \left(a + a_j^n\right) \rho_j^n} - 1 \right] \tag{H.1}$$

$s_{xi}^n$ is the synaptic weight between afferent cell $i$ and target cell $x$ at time step $n$. $a_i^n$ is the neural activity level of afferent cell $i$ at time step $n$, in the range from zero to one. $T_0$ and $T_1$ are related to the release rate of NT by target cells, $a$ is related to the uptake of NT by afferent cells, these terms are more fully explained below. $\Delta_{xy}$ is the diffusion of NT between target cells $x$ and $y$. $\rho_i^n$ has two alternative formulations

$$\rho_i^n = \begin{cases} \lambda & \text{Case 1 (cell death)} \\ \lambda \bar{a}_i^n / \sum_x s_{xi}^n & \text{Case 2 (no cell death)} \end{cases} \tag{H.2}$$

where $\lambda$ is an arbitrary constant and $\bar{a}_i^n$ is the time average activity level of afferent cell $i$ at time step $n$. The terms *cell death* and *no cell death* derives from the fact that under the first case, the total synaptic weight from any afferent cells may drop to zero, representing the death of that afferent cell. Under the second case, total synaptic weight will always remain greater than zero, since cells with small total synaptic weight are given an advantage in the competition with other cells with higher total synaptic weight. In the NeuroGene simulation for refinement and occular dominance, the second form will be used. The implementation of the time average $\bar{a}_i^n$ is outlined below.

The equations above are derived in great detail in Elliott and Shadbolt (1998a). I will not repeat this derivation here. Instead, the expressions will be rationalized in terms of the biological process they represent. For the purposes of this rationalization, the equation H.1 is rearranged so as to clearly separate the presynaptic and postsynaptic components of the learning process

$$s_{xi}^{n+1} - s_{xi}^n = \varepsilon \left[ \frac{s_{xi}^n (a + a_i^n) \rho_i^n}{\sum_j s_{xj}^n \left(a + a_j^n\right) \rho_j^n} \sum_y \Delta_{xy} \left( T_0 + T_1 \frac{\sum_j s_{yj}^n a_j^n}{\sum_j s_{yj}^n} \right) - s_{xi}^n \right] \tag{H.3}$$

### H.1.1 Postsynpatic mechanism

Consider first the second half of the large term in the above expression. It describes the NT production and diffusion:

$\sum_j s_{yj}^n$ is the total weight of all the synapses connecting to the target cell $y$.

$\sum_j s_{yj}^n a_j^n$ is the total activity coming into the target cell $y$ from all afferent cells $j$.

$\sum_j s_{yj}^n a_j^n / \sum_j s_{yj}^n$ is then the ratio of the current activity level to the maximal possible activity level of target cell $y$, since $a_j^n$ has an upper bound of one for all $j$.

$T_0 + T_1 \left( \sum_j s_{yj}^n a_j^n / \sum_j s_{yj}^n \right)$ is the rate of production of NT from target cell $y$. Since $a_j^n$ is in the range $[0, 1]$, so is the fraction $\sum_j s_{yj}^n a_j^n / \sum_j s_{yj}^n$, meaning the rate is in the range

$[T_0, T_0+T_1]$. $T_0$ gives the activity-independent component, and $T_1$the component that depends on the amount of activity coming into the cell. The gene NT in the gene script follows this expression pattern, see lines 48–65 in the listing of the gene script in appendix B.2.3. We use values 0 and 0.02 for $T_0$ for $T_1$ respectively.

$\Delta_{xy}\left(T_0+T_1\left(\sum_j s^n_{yj}a^n_j / \sum_j s^n_{yj}\right)\right)$ is the amount of NT produced at cell $y$ that reaches cell $x$. $\Delta_{xy}$ represents the diffusion of NT from cell $y$ to cell $x$. $\Delta_{xy}$ is normalized so that $\sum_y \Delta_{xy} = 1$ for all values of $x$. Summed over all cells $y$, this gives the total amount of NT present in the vicinity of cell $x$. In the NeuroGene simulation, the *local* receptor-ligand relationship (line 93) and the local expression of NT (line 62) means that diffusion does not occur, so that $\Delta_{xy} = 1$ for $x = y$ and $\Delta_{xy} = 0$ for all $x \neq y$. For the significance of local expression, see appendix B.3.

### H.1.2 Presynaptic mechanism

Now consider the first half of the large term in (H.3), it describes the competition for NT uptake among afferent cells:

$(a + a^n_i)$ describes the affinity of afferent cell $i$ to the NT protein. This affinity has an activity-independent part $a$ and also depends on the current activity level of the cell, $a^n_i$, which is in the range $[0, 1]$ as before.

$s^n_{xi}(a + a^n_i)\rho^n_i$ describes the ***ability*** of afferent cell $i$ to take up NT from the vicinity of target cell $x$. This ability depends on the strength (or weight) $s^n_{xi}$ of the synaptic contact between afferent $i$ and target cell $x$, and on $\rho^n_i$. In the NeuroGene simulation, this ability is determined by the surface concentration of the NT receptor protein. The expression of the NT receptor gene fulfills this requirement (using the no-cell-death form of $\rho^n_i$ as defined in equation H.2), see lines 66–88 in appendix B.3. We have used $a = 1$ and $a^n_i = 1$ (or 0) when the retinal ganglion cell is firing (or not firing). These give the constants 2 and 1 in the gene on lines 79 and 84. Finally we use the paramter value $\lambda = 10^6$ to ensure that the absolute level of receptor is much greater than that of the ligand.

$\sum_j s^n_{xj}\left(a + a^n_j\right)\rho^n_j$ is the sum of the NT uptake abilities near target cell $x$ of all the afferent cells. The higher this value, the more competition there is near cell $x$ for the available amount of NT.

$s^n_{xi}\,(a+a^n_i)\,\rho^n_i / \sum_j s^n_{xj}\left(a+a^n_j\right)\rho^n_j$ is then the NT uptake ability of afferent cell $i$ as fraction of the total NT uptake ability of all the afferent cells that project to target cell $x$. It describes the fraction of the total amount of NT that is taken up by the synaptic terminal of afferent cell $i$ in competition with all the other afferent cells that project to the target cell $x$. This competitive relationship is modeled using the NeuroGene receptor-ligand functionality using a binding constant indicating strong binding of the ligand to the receptor. See line 93 of the genome in appendix B.2.3.

## H.2 Ligand-receptor binding

The receptor-ligand binding is governed by a chemical equilibrium between the free receptor ($R$), the free ligand ($L$) and the receptor-ligand complex ($C$). The equilibrium is described by a dissociation constant $K_d$:

$$C \rightleftharpoons R+L \qquad K_d = \frac{[R][L]}{[C]} \tag{H.4}$$

where the square brackets signify "the concentration of". If the initial, non-equilibrium concentrations of receptor, ligand and complex are designated by $[R]_0$, $[L]_0$ and $[C]_0$, then the equilibrium concentrations are

$$\begin{aligned} [C] &= ([C]_0 + \Delta) && \text{(H.5)}\\ [R] &= ([R]_0 - \Delta) && \text{(H.6)}\\ [L] &= ([L]_0 - \Delta)\,, && \text{(H.7)} \end{aligned}$$

the rationale being that each time a ligand molecule binds to a receptor molecule, one molecule each of $L$ and $R$ is lost, and one molecule of $C$ is gained. $\Delta$, which may be positive or negative, is computed by substituting (H.5)–(H.7) into equation H.4. Of the two solutions to the resulting quadratic equation, the one is selected which give non-negative values for $[R]$ and $[L]$, see section 2.4.4.

The underlying assumption about the receptor-ligand binding expressed in equation H.3 is that very close to all the available ligand binds to the receptor, i.e. $[L] \simeq 0$. For this condition to be satisfied, it is required that:

- The receptor must have high affinity for the ligand ($K_d \ll 1$).
- The receptor concentrations is much greater than the ligand concentration ($[R] \gg [L]$), since each molecule of the receptor can only bind one molecule[1] of the ligand.

This requirement is met by ensuring that the expression rate of the receptor is much greater than that of the ligand, and the dissociation constant is small, representing strong binding of the ligand to the receptor. In equation H.3 the absolute magnitude of the receptor expression rate ($a + a_i^n$) cancels out. This means that the receptor expression rate can be freely scaled to meet the condition $[R] \gg [L]$. For the expression of receptor, see lines 79 and 84 with $\lambda = 10^6$ (line 37)[2]. For that of the ligand, see lines 58–62 with $T_0 = 0$ and $T_1 = 0.02$ (lines 35 and 36). The binding constant is equal to $10^{-12}$M as defined on line 93.

## H.3 From concentrations to learning

Our goal is now to restate the learning rule (H.3) in terms of concentrations of proteins, using (H.4) and the relationships outlined in sections H.1.1–H.1.2. Since virtually all the available ligand binds to receptors (i.e. $[L] \simeq 0$) we get from (H.7) that $x = -[L]_0$. If we consider an initial situation in which no receptor-ligand complex exists ($[C]_0 = 0$), we have from (H.5) that $[C] = -x$, which gives

$$[C] = [L]_0. \tag{H.8}$$

The amount $[C]_{ix}$ of ligand bound to afferent cell $i$ close to target cell $x$ is

$$[C]_{ix} = \frac{[R]_{0ix}}{[R]_{0x}}[C]_x \tag{H.9}$$

Here the subscript $x$ has been introduced to signify the concentrations in the vicinity of the target cell $x$. Keep in mind that the afferent cell $i$ may extend over a large area of dissimilar chemical environments, and that it may form contacts with a number of different target cells

[1]The same holds true if the receptor can bind a constant number of ligand molecules. However, this would require modifications to equation H.4.

[2]Note that the expression rates are scaled by the cell component surface area. The small surface area of synaptic terminals (set to equal 1% of that of the soma, see section 2.2.3) means that the difference between the expression rates of receptor and ligand is not quite as large as it seems, since the receptor is expressed by synaptic termini and the ligand by somas.

$x$. The concentration $[C]_{ix}$ therefore refers to the portion of cell $i$ that is close to a particular target cell $x$. If we consider that only the synapses of the afferent cells carry the receptor, then $[R]_{0ix}$ is the total amount of receptor (ligand-bound and free) carried by the synapse of afferent cell $i$ making contact with the target cell $x$. $[R]_{0x}$ is the total amount of receptor carried by *all* the synapses making contact with the target cell $x$ and $[C]_{ix}$ is the amount or receptor-ligand complex on the synapse connecting cell $i$ to cell $x$.

By substituting (H.8) into (H.9) we get

$$[C]_{ix} = \frac{[R]_{0ix}}{[R]_{0x}}[L]_{0x} \tag{H.10}$$

where $[L]_{0x}$ is the total ligand concentration close to cell $x$ (both free and bound to receptor). This relationship gives the (unknown) equilibrium concentration of the complex in terms of the (known) initial concentrations of ligand and receptor. From the above rationalization of equation H.3 we have that the affinity of cell $i$ as fraction of the affinity for all cells $j$ close to target cell $x$ is given by (from section H.1.2)

$$\frac{[R]_{0ix}}{[R]_{0x}} = \frac{s_{xi}^n\,(a + a_i^n)\,\rho_i^n}{\sum_j s_{xj}^n \left(a + a_j^n\right) \rho_j^n} \tag{H.11}$$

and that the amount of NT at the target cell $x$ is given by (from section H.1.1)

$$[L]_{0x} = \sum_y \Delta_{xy} \left( T_0 + T_1 \frac{\sum_j s_{yj}^n a_j^n}{\sum_j s_{yj}^n} \right)$$

From this we can rewrite equation H.3, using H.10, as

$$s_{xi}^{n+1} - s_{xi}^n = \varepsilon \left( \frac{[R]_{0ix}}{[R]_{0x}} [L]_{0x} - s_{xi}^n \right) = \varepsilon \left( [C]_{xi} - s_{xi}^n \right) \tag{H.12}$$

The change in synaptic weight is thus proportional to the deviation of the ligand-receptor complex concentration from the current synapse weight, with $\varepsilon$ as the learning rate. The more NT that binds to receptors on the synapse between cells $i$ and $x$, the stronger the synapse becomes. But high level of binding of NT comes about when the postsynaptic cell fires (producing NT) in synchrony with the firing of the presynaptic cell (producing NT receptor). We see that equation H.3 in fact embodies a Hebbian learning rule. This part of the learning rule is implemented in the gene *learn*, see lines 94–108.

## H.4 Time-average activity level

As mentioned in section H.1.2, the expression rate of the receptor depends $\rho_i^n$. In the no-cell-death case we have that

$$\rho_i^n = \frac{\lambda \bar{a}_i^n}{\sum_x s_{xi}^n}$$

(see equation H.2), i.e., the expression rate of the receptor depends on the recent time-average activity level $\bar{a}_i^n$ of the presynaptic cell, as well as the instantaneous activity level $a_i^n$. From equation H.11 we see that the constant $\lambda$ cancels out from the expression and need not concern us here. $\bar{a}_i^n$ is defined as an exponentially decaying average with time constant $\tau$. With continuous time, this gives

$$\bar{a}_i(t) = \frac{1}{\tau}\int_{-\infty}^{t} a(t')\exp\left(-\frac{t-t'}{\tau}\right)dt'$$

With descrete time this becomes

$$\begin{aligned}\bar{a}_i^{n+1} &= (1-\exp(-1/\tau))\sum_{m=-\infty}^{n} a_i^m \exp\left(-\frac{n-m}{\tau}\right)\\ &= a_i^{n+1}(1-\exp(-1/\tau)) + \bar{a}_i^n\exp(-1/\tau)\\ &= a_i^{n+1}(1-\phi)+\bar{a}_i^n\phi \end{aligned} \tag{H.13}$$

with $\phi = \exp(-1/\tau)$. It is shown in Elliott and Shadbolt (1998a) that $\phi$ is in fact related to the learning rate $\varepsilon$ through the following simple relation

$$\varepsilon = 1-\phi$$

Using this relation, the averaging scheme is restated as

$$\bar{a}_i^{n+1} = a_i^{n+1}\varepsilon + \bar{a}_i^n(1-\varepsilon) \tag{H.14}$$

The gene called "timeAverageActivity", which modeles $\bar{a}_i^{n+1}$, has the following properties:

- The decay rate of the protein expressed by the gene is $k = -\ln(1-\varepsilon)$. This means that each time step, the concentration of the protein carried over from the previous time step is scaled by an amount $\exp(-k) = 1-\varepsilon$.
- When the cell is firing, i.e. $a_i^{n+1} = 1$, the expression rate is $\varepsilon$.

- When the cell is not firing, i.e. $a_i^{n+1} = 0$, the expression rate is zero.

Together these properties mean that the concentration of the protein expressed by the gene at all times reflect the time-average activity level of the cell, suject to the averaging scheme outlined above. The gene is defined in lines 38–47.

# Index of language primitives

# Bibliography

Agarwal, Pankaj. 1995. The cell programming language. *Artificial Life* 2(1):37–77.

Astor, J. C., and C. Adami. 2000. A developmental model for the evolution of artificial neural networks. *Artificial Life* 6(3):189–218.

Atkins, Peter W., and Julio de Paula. 2002. *Physical chemistry.* 7th ed. New York: W. H. Freeman.

Bachy, Isabelle, Philippe Vernier, and Sylvie Rétaux. 2001. The LIM-homeodomain gene family in the developing *Xenopus* brain: Conservation and divergences with the mouse related to the evolution of the forebrain. *J. Neurosci.* 21:7620–7629.

Baier, H., and F. Bonhoeffer. 1992. Axon guidance by gradients of a target-derived component. *Science* 255:472–475.

Barrow, Jeffrey R., H. Scott Stadler, and Mario R. Capecchi. 2000. Roles of *Hoxa1* and *Hoxa2* in patterning the early hindbrain of the mouse. *Development* 127:933–944.

Bi, Guo-qiang, and Mu-ming Poo. 1999. Distributed synaptic modification in neural networks induced by patterned stimulation. *Nature* 401:792–796.

Bielsky, Isadora F., and Larry J. Young. 2004. Ocytocin, vasopressin, and social recognition in mammals. *Peptides* 25:1565–1574.

Bishop, Kathie M., John L. R. Rubenstein, and Dennis D. M. O'Leary. 2002. Distinct actions of *Emx1*, *Emx2*, and *Pax6* in regulating the specification of areas in the developing neocortex. *J. Neurosci.* 22(17):7627–7638.

Bonhoeffer, Tobias. 1996. Neurotrophins and activity-dependent development of the neocortex. *Curr. Opin. Neurobiol.* 6:119–126.

Bower, James M., and David Beeman. 1995. *The book of Genesis - exploring realistic neural models with the GENESIS.* Springer-Verlag.

Brickley, Stephen G., Elizabeth A. Dawes, Michael J. Keating, and Simon Grand. 1998. Synchronizing retinal activity in both eyes disrupts binocular map development in the optic tectum. *J. Neurosci.* 18(4):1491–1504.

Brook, William. 1998. Genetic control of segmentation in drosophila: The maternal legacy. Available: `http://www.ucalgary.ca/UofC/eduweb/virtualembryo/D_m_segment_I.html` Accessed: [June 29-2004].

Brose, N. 1999. Synaptic cell adhesion proteins and synaptogenesis in the mammalian central nervous system. *Naturwissenschaften* 86(11):516–524.

Browder, Leon W., Carol A. Ericson, and William R. Jeffery. 1991. *Developmental biology.* 3rd ed. Saunders College Publishing.

Brown, Arthur, Paul A. Yates, Patric Burrola, Dan Ortuno, Vaidya Ashish, Thomas M. Jesselt, Samuel L. Pfaff, Dennis D. M. O'Leary, and Greg Lemke. 2000. Topographic mapping from the retina to the midbrain is controlled by relative but not absolute levels of EphA receptor signaling. *Cell* 102:77–88.

Buck, Kenneth B., and James Q. Zheng. 2002. Growth cone turning induced by direct local modification of microtubule dynamics. *J. Neurosci.* 22(21):9358–9367.

Buettner, H. M., R. N. Pittman, and J. K. Ivins. 1994. A model of neurite extension across regions of nonpermissive substrate: simulations based on experimental measurements of growth cone motility and filopodial dynamics. *Dev. biol.* 163:407–422.

Butts, Daniel A., and Daniel S. Rokhsar. 2001. The information content of spontaneous retinal waves. *J. Neurosci.* 21(3):961–973.

Cangelosi, Angelo, Domenico Parisi, and Stefano Nolfi. 1994. Cell division and migration in a 'genotype' for neural networks. *Network: Comp. in Neur. Syst.* 5:497–515.

Carmichael, S. Thomas, and Marie-Françoise Chesselet. 2002. Synchronous neuronal activity is a signal for axonal sprouting after cortical lesions in the adult. *J. Neurosci.* 22(14): 6062–6070.

Chen, Chinfei, and Wade G. Regerh. 2000. Developmental remodeling of the retinogeniculate synapse. *Neuron* 28:955–966.

Cline, Hollis T. 2001. Dendritic arbor development and synaptogenesis. *Curr. Opin. Neurobiol.* 11:118–126.

Connor, Robert J., Patricia Menzel, and Elena B. Pasquale. 1998. Expression and tyrosine phosphorylation of eph receptors suggest multiple mechanisms in patterning of the visual system. *Dev. Biol.* 193(1):21–35.

Cook, Jeremy E. 1979. Interactions between optic fibres controlling the locations of their terminals in the goldfish optic tectum. *J. Embryol. Exp. Morph.* 52:89–103.

Crair, Michael C. 1999. Neuronal activity during development: permissive or instructive? *Curr. Opin. Neurobiol.* 9:88–93.

Cruise, Robert. 2001. Numerical solution of the diffusion equation. Available: `http://www.iu.edu/~rac/hpc/mpi_tutorial/10-0.html` [Accessed: June 11-2004].

von Dassov, George, Eli Meir, Edwin M. Munro, and Garrett M. Odell. 2000. The segment polarity network is a robust developmental module. *Nature* 406:188–192.

Davidson, Eric H. 2001. *Genomic regulatory systems — development and evolution.* Academic Press.

Davidson, Eric H., Jonathan P. Rast, Paola Oliveri, Andrew Ransick, Cristina Calestani, Chiou-Hwa Yuh, Takuya Minokawa, Gabriele Amore, Veronica Hinman, Cesar Arenas-Mena, Ochan Otim, C. Titus Brown, Carolina B. Livi, Pei Yun Lee, Roger Revilla, Alistair G. Rust, Zheng jun Pan, Maria J. Schilstra, Peter J. C. Clarke, Maria I. Arnone, Lee Rowen, R. Andrew Cameron, David R. McClay, Leroy Hood, and Hamid Bolouri. 2002. A genomic regulatory network for development. *Science* 295:1669–1678.

Davy, Alice, Nicholas W. Gale, Elizabeth W. Murray, Richard A. Klinghoffer, Philippe Soriano, Claude Feuerstein, and Stephen M. Robbins. 1999. Compartmentalized signaling by GPI-anchored ephrin-A5 requires the Fyn tyrosine kinase to regulate cellular adhesion. *Genes devel.* 13:3125–35.

Debski, Elizabeth A., and Hollis T. Cline. 2002. Activity-dependent mapping in the retinotectal projection. *Curr. Opin. Neurobiol.* 12:93–99.

DeVaul, Richard W., and Bruce H. McCormick. 1996. Neuron developmental modeling and structural representation: An introduction to the N++ language, and open stochastic L-system. Tech. Rep., Scientific Visualization Lab., Dept. Comp. Sci., Texas A&M University.

Diamond, Mathew E., Rasmus S. Petersen, Justin A. Harris, and Stefano Panzeri. 2003. Investigations into the organization of information in sensory cortex. *J. Physiol. Paris* 97(4-6):529–536.

Díaz, Elva, Yee Hwa Yang, Todd Ferreira, Kenneth C. Loh, Yasushi Okazaki, Yoshihide Hayashizaki, Marc Tessier-Lavigne, Terence P. Speed, and John Ngai. 2003. Analysis of gene expression in the developing mouse retina. *Proc. Nat. Acad. Sci. U.S.A.* 100(9):5491–5496.

DuCharme, Bob. 1999. *XML: The annotated specification.* Prentice Hall PTR.

Dutilh, Bas E., and P. Hogeweg. 1999. Gene networks from microarray data. Available: `http://www-binf.bio.uu.nl/~dutilh/research/gene-networks/` [Accessed: August 5-2004].

Eggenberger, Peter. 1997a. Creation of neural networks based on developmental and evolutionary princliples. In *Proceedings of the international conference on artificial neural networks (ICANN'97).*

———. 1997b. Evolving morphologies of simulated 3d organisms based on differential gene expression. In *Proceedings of the 4th european conference on artificial life (ECAL97)*, ed. Phil Husbands and Inman Harvey. MIT Press.

Elliott, Terry, and Nigel R. Shadbolt. 1998a. Competition for neurotrophic factors: Mathematical analysis. *Neur. Comput.* 10:1939–1981.

———. 1998b. Competition for neurotrophic factors: Ocular dominance columns. *J. Neurosci.* 18(15):5850–5858.

———. 1999. A neurotrophic model of the development of the retinogeniculocortical pathway induced by spontaneous retinal waves. *J. Neurosci.* 19(18):7951–7970.

———. 2002. Multiplicatice synaptic normalization and a nonlinear Hebb rule underlie a neurotrophic model of competitive synaptic plasticity. *Neur. Comput.* 14:1311–1322.

Elowitz, Michael B., and Stanislas Leibler. 2000. A synthetic oscillatory network of transcriptional regulators. *Nature* 403:335–338.

Eph nomenclature committee. 1997. Unified nomenclature for Eph family receptors and their ligands. *Cell* 90:403–404.

Feldheim, David A., Young-In Kim, Andrew D. Bergemann, Jonas Frisén, Mariano Berbacid, and John G. Flanagan. 2000. Genetic analysis of ephrin-A2 and ephrin-A5 shows their requirement in multiple aspects of retinocollicular mapping. *Neuron* 25(3): 563–574.

Feller, Maria B., Daniel A. Butts, Holly L. Aaron, Daniel S. Rokhsar, and Carla J. Shatz. 1997. Dynamic processes shape spatiotemporal properties of retinal waves. *Neuron* 19:293–306.

Fiala, John C., Marcia Feinberg, Viktor Popov, and Kristen H. Harris. 1998. Synaptogenesis via dendritic filopodia in developing hippocampal area CA1. *J. Neurosci.* 18(21): 8900–8911.

Finlay, B. L., S. E. Schneps, and G. E. Schneider. 1979. Orderly compression of the retinotectal projection following partial tectal ablation in the newborn hamster. *Nature* 280:153–155.

Flanagan, John G., and Pierre Vanderhaegen. 1998. The ephrins and Eph receptors in neural development. *Annu. Rev. Neurosci.* 21:309–345.

Fleischer, Kurt, and Alan H. Barr. 1993. Simulation testbed for multicellular development: The multiple mechanisms of morphogenesis. In *Artificial life iii*. Addison-Wesley.

Fraser, Scott E., and Donald H. Perkel. 1990. Competitive and positional cues in the patterning of nerve connections. *J. Neurobiol.* 21(1):51–72.

Fu, Yu-Xi, Kaj Djupsund, Hongfeng Gao, Benjamin Hayden, Kai Shen, and Yang Dan. 2002. Temporal specificity in the cortical plasticity of visual space representation. *Science* 296:1999–2003.

Fukuchi-Shimogori, Tomomi, and Elizabeth A. Grove. 2001. Neocortex patterning by the secreted signaling molecule fgf8. *Science* 294:1071–1074.

Furrer, Marie-Pierre, Susan Kim, Brian Wolf, and Akira Chiba. 2003. Robo and frazzled/DCC mediate dendritic guidance at the CNS midline. *Nature Neurosci.* 6(3): 223–230.

Gaze, R. M., M. Jacobson, and G. Székely. 1963. The retino-tectal projection in *Xenopus* with compund eyes. *J. Physiol.* 165:484–499.

Gehring, W. J. 1998. *Master control genes in development and evolution: the homeobox story.* The Terry lectures, Yale University Press.

GeneCards. 2004. GeneCard for gene PAX6 GC11M031775. Available: `http://bioinformatics.weizmann.ac.il/cards-bin/carddisp?PAX6` [Accessed: Aug 16-2004].

Gierer, A. 1983. Model for the retino-tectal projection. *Proc. R. Soc. Lond. B* 218(1210): 77–93.

Gnuegge, Lara, Susanne Schmid, and Stephan C. F. Neuhauss. 2001. Analysis of the activity-depeprived zerbrafish mutant *macho* reveals an essential requirement of neuronal activity for the development of a fine-grained visuotopic map. *J. Neurosci.* 21(10): 3542–3548.

Gomez, Timothy M., and Nicholas C. Spitzer. 2000. Regulation of growth cone behaviour by calcium: new dynamics to earlier perspectives. *J. Neurobiol.* 44:174–183.

Goodhill, Geoffrey J. 1998. Mathematical guidance for axons. *Trends. Neurosci.* 21:226–231.

———. 2003. A theoretical model of axon guidance by the Robo code. *Neur. Comput.* 15: 549–564.

Goodhill, Geoffrey J., and Siegrid Löwel. 1996. Theory meets experiment: correlated neural activity helps determine ocular dominance column periodicity. *Trends Neurosci.* 18: 437–439.

Goodhill, Geoffrey J., and Linda J. Richards. 1999. Retinotectal maps: molecules, models and misplaced data. *Trends in Neurosci.* 22(12):529–534.

Goodhill, Geoffrey J., and Jeffrey S. Urbach. 1999. Theoretical analysis of gradient detection by growth cones. *J. Neurobiol.* 41:230–241.

Gordon-Weeks, Phillip R. 2000. *Neuronal growth cones.* Cambridge University Press.

Grand, Mark. 1998. *Patterns in Java: a catalog of reusable design patterns illustrated with UML.* Wiley computer publishing.

Hadley, Robert F. 1999. Connectionism and novel combinations of skills: Implications for cognitive architecture. *Minds and Machines* 9(2):197–221.

———. 2003. A defence of functional modularity. *Connection Sci.* 15(2-3):95–116.

Halder, G., P. Callaerts, S. Flister, U. Walldorf, U. Kloter, and W. J. Gehring. 1998. Eyeless initiates the expression of both, sine oculis and eyes absent during Drosophila compound eye development. *Development* 125:2181–2191.

Halder, G., P. Callaerts, and W. J. Gehring. 1995. Induction of ectopic eyes by target expression of the eyeless gene in Drosophila. *Science* 267:1788–1792.

Halloran, Mary C., and Katherine Kalil. 1994. Dynamic behaviors of growth cones extending in the corpus callosum of living cortical brain slices observed with video microscopy. *J. Neurosci.* 14(4):2161–2177.

Harding, Stephen E., and Babur Z. Chowdhry. 2001. *Protein-ligand interactions*, vol. 243 of *The Practical Approach Series.* Oxfort University Press.

Hatada, Yohko, Fang Wu, Rachel Silverman, Samuel Schacher, and Daniel J. Goldberg. 1999. En passant synaptic varicosities form directly from growth cones by transient cessation of growth cone advance but not of actin-based motility. *J. Neurobiol.* 41: 242–251.

Hebb, Donald O. 1949. *The organization of behavior; a neuropsychological theory.* Wiley.

Held, Lewis I., Jr. 2002. *Imaginal discs: The genetic and cellular logic of pattern formation.* Cambridge University Press.

Hely, T. A., and D. J Willshaw. 1998. Short term interactions between microtubules and actin filaments underlie long term behaviour in neuronal growth cones. *Proc. R. Soc., Biol. Sci.* 265(1407):1801–1807.

Hely, Tim A. 1998. Computational models of developing neural systems. Ph.D. thesis, University of Edinburgh.

Hentschel, H. G. E., and A. van Ooyen. 1999. Models of axon guidance and bundling during development. *Proc. R. Soc. Lond. B. Biol. Sci.* 266:2231–2238.

Hindges, Robert, Todd McLaughlin, Nicolas Genoud, Mark Henkenmeyer, and Dennis D. M. O'Leary. 2002. EphB forward signaling controls directional branch extension and arborization required for dorsal-ventral retinotopic mapping. *Neuron* 35(3):475–487.

Hines, Michael L., and N. Ted Carnevale. 2003. The NEURON simulation environment. In *The handbook of brain theory and neural networks*, ed. Arbib M. A., 2nd ed., 769–773. MIT Press.

Hope, R. A., B. J. Hammon, and Gaze R. M. 1976. The arrow model: retinotectal specificity and map formation in the goldfish visual system. *Proc. R. Soc. Lond. B.* 194:447–466.

Horder, T. J. 1971. Retention, by fish optic nerve fibres regenerating to new terminal sites in the tectum, of 'chemospecific' affinity for their original sites. *J. Physiol.* 216: P53–P55.

Hornberger, Martin R., Dieter Dütting, Thomas Ciossek, Tomoko Yamada, Claudia Handwerker, Susanne Lang, Franco Weth, Julita Huf, Ralf Weßel, Cairine Logan, Hideaki

Tanaka, and Uwe Drescher. 1999. Modulation of EphA receptor function by coexpressed ephrinA ligands on retinal ganglion cell axons. *Neuron* 22:731–742.

Hudson, Scott, Frank Flannery, and C. Scott Ananian. 2003. CUP parser generator for Java. Available: `http://www.cs.princeton.edu/~appel/modern/java/CUP/` [Accessed: June 10-2003].

Hummel, John E., and Irving Biederman. 1992. Dynamic binding in a neural network for shape recognition. *Psychological Review* 99(3):480–517.

Hunt, R. K., and M. Jacobson. 1973. Development of neuronal locus specificity in *Xenopus* retinal ganlion cells after surgical eye transection or after fusion of whole eyes. *Dev. Biol.* 40:1–15.

Innocenti, G.M. 1995. Exuberant development of connections and its possible permissive role in cortical evolution. *Trends Neurosci.* 18:397–402.

de Jong, Hidde. 2002. Modeling and simulation of genetic regulatory systems: A literature review. *J. Comput. Biol.* 9(1):67–103.

Judd, Ellen M., Michael T. Laub, and Harley H. McAdams. 2000. Toggles and oscillators: new genetic circuit designs. *BioEssays* 22(6):507–509.

Kalil, Katherine, Gyorgyi Szebenyi, and Eric W. Dent. 2000. Common mechanisms underlying growth cone guidance and axon branching. *J. Neurobiol.* 44(2):145–158.

Kandel, Eric R., James H. Schwartz, and Thomas M. Jessel. 2000. *Principles of neural science.* McGraw-Hill.

Knöll, Bernd, and Uwe Drescher. 2002. Ephrin-As as receptors in topographic projections. *Trends Neurosci.* 25(3):145–149.

Knöll, Bernd, Hannes Schmidt, William Andrews, Sarah Guthrie, Adrian Pini, Vasi Sundaresan, and Uwe Drescher. 2003. On the topographic targeting of basal vomeronasal axons through Slit-mediated chemorepulsion. *Development* 130:5073–82.

Knöll, Bernd, Konstantinos Zarbalis, Wolfgang Wurst, and Uwe Drescher. 2001. A role of the EphA family in the topographic targeting of vomeronasal axons. *Development* 128:895–906.

Kolls, Bradley J., and Ronald L. Meyer. 2002. Spontaneous retinal activity is tonic and does not drive tectal activity during activity-dependent refinement in regeneration. *J. Neurosci.* 22(7):2626–2636.

Kreyszig, Erwin. 1988. *Advanced engineering mathematics.* John Wiley & Sons.

Krubitzer, Leah, and Kelly J. Huffman. 2000. Arealization of the neocortex in mammals: Genetic and epigenetic contributions to the phenotype. *Brain, Behavious and Evolution* 55:322–225.

Kummer, H. 1971. *Primate societies.* Arlington Heights.

Lai, Cecillia S. L., Simon E. Fisher, Jane A. Hurst, Faraneh Vargha-Khadem, and Anthony P. Monaco. 2001. A forkhead-domain gene is mutated in a severe speech and language disorder. *Nature* 413:519–523.

Levesque, Hector, Fiora Pirri, and Ray Reiter. 1998. Foundations for the situation calculus. *Linkooping electronic Articles in Computer and Information Science* 3.

Li, Guo-Hua, Cheng-De Qin, and Mao-Hui Li. 1994. On the mechanisms of growth cone locomotion: Modeling and computer simulation. *J. theor. biol.* 169:355–362.

Liberty, Jesse. 1998. *Beginning object-oriented analysis and design.* Wrox Press Ltd.

Lindenmayer, Aristide. 1968. Mathematical models for cellular interaction in development, Parts I and II. *J. Theor. Biol.* 18:280–315.

Liu, Zheng, Barry J. Richmond, Elisabeth A. Murray, Richard C. Saunders, Sara Steenrod, Barbara K. Stubblefield, Deidra M. Montague, and Edward I. Ginns. 2004. DNA targeting of rhinal cortex D2 receptor protein reversibly blocks learnig of cues that predict reward. *Proc. Natl. Acad. Sci. U.S.A.* 101(33):12336–12341.

Lodish, Harvey, Arnold Berk, S. Lawrence Zipursky, Paul Matsudaira, David Baltimore, and James Darnell. 2000. *Molecular cell biology.* 4th ed. W. H. Freeman.

Logo Foundation. 2004. Logo foundation web-site. Available: `http://el.media.mit.edu/logo-foundation/` [Accessed: August 10-2004].

Louden, Kenneth C. 1993. *Programming languages: principles and practice.* PWS Publishing Company.

Lövel, Sigrid. 1994. Occular dominance column development: strabismus changes the spacing of adjacent columns in cat visual cortex. *J. Neurosci.* 14(12):7451–7468.

Luger, George F., and William A. Stubblefield. 1998. *Artificial intelligence — structures and strategies for complex problem solving.* 3rd ed. Addison Welsey Longman, Inc.

Maletic-Savatic, Mirjana, and Roberto Malinow. 1998a. Calcium-evoked dendritic exocytosis in cultured hippocampal neurons. Part I: Trans-golgi network-derived organelles undergo regulated exocytosis. *J. Neurosci.* 18(17):6803–6813.

———. 1998b. Calcium-evoked dendritic exocytosis in cultured hippocampal neurons. Part II: Mediation by calcium/calmodulin-dependent protein kinase II. *J. Neurosci.* 18(17):6814–6821.

Mann, Fanny, Samiran Ray, William A. Harris, and Christine E. Holt. 2002. Topographic mapping in dorsoventral axis of the *Xenopus* retinotectal system depends on signaling through ephrin-B ligands. *Neuron* 35(3):461–473.

Marcus, Gary F. 2001. *The algebraic mind.* MIT press.

———. 2004. *The birth of the mind — how a tiny number of genes creates the complexities of human thought.* Basic Books.

Marín, Oscar, María José Blanco, and M. Angela Nieto. 2001. Differential expression of Eph receptors and ephrins correlates with the formation of topographic projections in primary and secondary visual circuits of the empryonic chick forebrain. *Devel. Biol.* 234:289–303.

McLaughlin, Todd, Robert Hindges, and Dennis D. M. O'Leary. 2003a. Regulation of axial patterning of the retina and its topographic mapping in the brain. *Curr. Opin. Neurobiol.* 13:57–69.

McLaughlin, Todd, Christine L. Toborg, Marla B. Feller, and Dennis D. M. O'Leary. 2003b. Retinotopic map refinement requires spontaneous retinal waves during a brief critical period of development. *Neuron* 40:1147–1160.

Meier, Nils. 2002. InstantJ. Available: `http://sourceforge.net/projects/instantj/` [Accessed April 12-2004].

Meinhardt, Hans. 1999. Orientation of chemotactic cells and growth cones: models and mechanisms. *J. Cell. Sci.* 112:2867–2874.

Meir, Eli, George von Dassow, Edwin M. Munro, and Garrett M. Odell. 2002. Robustness, flexibility, and the role of lateral inhibition in the neurogenic network. *Curr. Biol.* 12:778–786.

von Melchner, L., S. L. Pallas, and M. Sur. 2000. Visual behaviour mediated by retinal projections directed to the auditory pathway. *Nature* 404:871–876.

Meyer, R. 1979. Retinotectal projection in goldfish to an inappropriate region with a reversal in polarity. *Science* 205:819–821.

Meyer-Franke, Anke, George A. Wilkinson, Alex Kruttgen, Minjie Hu, Elizabeth Munro, Martin G. Hanson, Jr., Louis F. Reichardt, and Ben A. Barres. 1998. Depolarization and cAMP elevation rapidly recruit TrkB to the plasma membrane of CNS neurons. *Neuron* 21(4):681–693.

Ming, Guo-li, Hong-jun Song, Benedikt Berninger, Christine E. Holt, Marc Tessier-Lavigne, and Poo Mu-ming. 1997. cAMP-dependent growth cone guidance by netrin-1. *Neuron* 19(6):1225–1235.

Muller, Werner A. 1997. *Developmental biology.* Springer.

Mutton, Paul. 2004. Java EPS Graphics2D package. Available: `http://www.jibble.org/epsgraphics/` [Accessed: April 12-2004].

Neet, K. E., and R. B. Campenot. 2001. Receptor binding, internalization, and retrograde transport of neurotrophic factors. *Cell Mol. Life Sci* 58(8):1021–1035.

O'Connor, Timothy P., Janet S. Duerr, and David Bentley. 1990. Pioneer growth cone steering decisions mediated by single filopoial contact in situ. *J. Neurosci.* 10(12): 3935–3946.

Oda, Yoichi, Keisuke Kawasaki, Masahiro Morita, Henri Korn, and Haruko Matsui. 1998. Inhibitory long-term potentiation underlies auditory conditioning of goldfish escape behaviour. *Nature* 394:182–185.

van Ooyen, Arjen, and Richard R. Ribchester. 2003. *Modeling neural development*, chap. Competition in the Development of Nerve Connections, 183–211. Developmental Cognitive Neuroscience, The MIT Press.

van Ooyen, Arjen, and David J. Willshaw. 1999. Competition for neurotrophic factor in the development of nerve connections. *Proc. R. Soc. Lond. B. Biol. Sci.* 266:883–892.

Overton, K., and M. Arbib. 1982. *Competition and cooperation in neural nets*, chap. Systems matching and topographic maps: the branch-arrow model, 202–225. Berlin: Springer.

Palka, J., K. E. Whitlock, and M. A. Murray. 1992. Guidepost cells. *Curr. Opin. Neurobiol.* 2:48–54.

Palzkill, Timothy. 2002. *Proteomics.* Kluwer Academic Publishers.

Pinker, Steven. 2001. Talk of genetics and vice versa [commentary on Lai et. al., 2001]. *Nature* 413:465–466.

PNG development group. 2004. Portable network graphics. Available: `http://www.libpng.org/pub/png/png.html` [Accessed: April 12-2004].

Press, William H., Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. 1986. *Numerical recipes: The art of scientific computing.* Cambridge University Press.

Prusinkiewicz, Przemyslaw, Mark Hammel, and Radomír Měch. 1995. The artificlal life of plants. *SIGGRAPH Course Notes* 7:1-1 - 1–38.

Rasmussen, Carl E., and David J. Willshaw. 1993. Competition in models for the development of neuromuscular connections. *Biol. Cybern.* 68:409–419.

Reiter, Raymond. 1991. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In *Artificial intelligence and mathematical theory of computation (Papers in honor of John McCarthy)*, ed. Vladimir Lifschitz, 359–380. Academic press.

———. 1996. Natural actions, concurrency and continuous time in the situation calculus. In *Principles of knowledge representation and reasoning: Proceedings of the fifth international conference (KR'96)*, ed. L. C. Aiello, J. Doyle, and S. C. Shapiro, 2–13. Morgan Kaufmann Publishers, San Fransisco, CA.

Rodriguez, A., J. Whitson, and R. Granger. 2004. Derivation and analysis of basic computational operations of thalamocortical circuits. *J. Cogn. Neurosci.* 16(5):856–877.

Rust, Alistair G., Rod Adams, and Hamid Bolouri. 1999. Towards computational neural systems through developmental evolution. *International Workshop on Emergent Neural Computation Based on Neuroscience* .

———. 2000. Evolutionary neural topiary: Growing and sculpting artificial neurons to order. *Proceedings of the 7th International Conference on Simulation and Synthesis of Living Systems (ALife VII)* .

Schmidt, J., C. Cicerone, and S. Easter. 1978. Expansion of the half retina projection to the tectum in goldfish: an electrophysiological and anatomical study. *J. Comp. Neurol.* 177:257–278.

Scholpp, Steffen, Claudia Lohs, and Michael Brand. 2003. *Engrailed* and *Fgf8* act synergistically to maintain the boundary between diencephalon and mesencephalon. *Development* 130:4881–4893.

Schulte, Dorotea, and Constance L. Cepko. 2000. Two homeobox genes define the domain of *EphA3* expression in the developing chick retina. *Development* 127:5033–5045.

Shatz, C. J., S. Lindstrom, and T. N. Wiesel. 1977. The distribution of afferents representing the right and left eyes in the cat's visual cortex. *Brain Res.* 131:103–116.

Shirazi, Jack. 2000. *Java performance tuning.* The Java Series, O"Reilly & Associates, Inc.

Smolen, P., D. A. Baxter, and J. H. Byrne. 2000a. Modeling transcriptional control in gene networks: Methods, recent results, and future directions. *Bull. Math. Biol.* 62: 247–292.

Smolen, Paul., Doublas. A. Baxter, and John. H. Byrne. 2000b. Mathematical modeling of gene networks. *Neuron* 26:567–580.

Song, Hong-jun, Guo-li Ming, and Mu-ming Poo. 1997. cAMP-induced switching in turning direction of nerve growth cones. *Nature* 388(17):275–279.

Song, Sen, Kenneth D. Miller, and L. F. Abbott. 2000. Competitive Hebbian learning through spike-timing dependent synaptic plasticity. *Nature Neurosci.* 3:919–926.

Sperry, Roger W. 1963. Chemoaffinity in the orderly growth of nerve fiber patterns and connections. *Proc. Natl. Acad. Sci. U.S.A.* 50:703–710.

St. John, James A., Heidi J. Clarris, and Brian Key. 2002. Multiple axon guidance cues establish the olfactory topographic map: How do these cues interact? *Int. J. Dev. Biol.* 46:639–47.

Stanley, Kenneth O., and Risto Miikkulainen. 2003. A taxonomy of artificial embryogeny. *Art. Life* 9:93–130.

Stern, Claudio D. 2001. Initial patterning of the central nervous system: How many organizers? *Nature Rev. Neurosci.* 2:92–98.

Sun Microsystems. 2003a. GZIPInputStream class documentation. Available: `http://java.sun.com/j2se/1.4.2/docs/api/java/util/zip/-GZIPInputStream.html` [Accessed April 12-2004].

———. 2003b. GZIPOutputStream class documentation. Available: `http://java.sun.com/j2se/1.4.2/docs/api/java/util/zip/-GZIPOutputStream.html` [Accessed April 12-2004].

———. 2003c. JavaCC. Available: `https://javacc.dev.java.net/` [Accessed April 12-2004].

———. 2004. Java$^{tm}$ 2 platform, standard edition, v 1.4.2. Available: `http://java.sun.com/j2se/1.4.2/docs/api/overview-summary.html` [Accessed April 12-2004].

Sur, M., S.L. Pallas, and A.W. Roe. 1990. Cross-modal plasticity in cortical development: differentiation and specification of sensory neocortex. *Trends Neurosci.* 13:227–233.

Swindale, Nicholas V. 1996. The development of topography in the visual cortex: a review of models. *Network: Computation in Neural Systems* 7:161–247.

Tai, Hsin-Chien, and Helen M. Buettner. 1998. Neurite outgrowth and growth cone morphology on micropatterned surfaces. *Biotechnol. Prog.* 14(3):364–370.

Takacs, J., P. Saillour, M. Imbert, M. Bogner, and J. Hamori. 1992. Effect of dark rearing on the volume of visual cortex (areas 17 and 18) and number of visual cortical cells in young kittens. *J. Neurosci. Res.* 32:449–459.

Takahashi, Hiroo, Takafumi Shintani, Hiraki Sakuta, and Masaharu Noda. 2003. CBF1 controls the retinotectal topographical map along the anteroposterior axis through multiple mechanisms. *Development* 130(21):5203-5215.

Taylor, John David. 2001. BeanPeeler. Available: `http://uk.geocities.com/johndavid_taylor/projects/beanpeeler/` [Accessed November 11-2004].

Thor, Stefan, Siv G. E. Andersson, Andrew Tomlinson, and John B. Thomas. 1999. A LIM-homeodomain combinatorial code of motor neuron pathway selection. *Nature* 397: 76–80.

Turin, Allan. 1952. The chemical basis of morphogenesis. *Phil. Trans. B.* 237:37–72.

Verhage, Matthijs, Ascanio S. Maia, Jaap J. Plomp, Arjen B. Brussaard, Joost H. Heeroma, Hendrika Vermeer, Ruud F. Toonen, Robert E. Hammer, Timo K. van den Berg, Markis Missler, Hans J. Geuze, and Thomas C. Südhof. 2000. Synaptic assembly of the brain in the absence of neurotransmitter secretion. *Science* 287:864–869.

Vicari, S., A. Albertoni, A. M. Chilosi, P. Cipriani, G. Cioni, and E. Bates. 2000. Plasticity and reorganization during language development in children with early brain injury. *Cortex* 36:31–46.

de Waal, Frans B. M. 2001. *The ape and the sushi master: cultural reflections of a primatologist.* Basic Books.

Weber, Cornelius, Helge Ritter, Jack Cowan, and Klaus Obermayer. 1997. Development and regeneration of the retinotectal map in goldfish: a computational study. *Phil. Trans. R. Soc. Lond. B.* 352(1361):1603–1623.

West, Anne E., Eric C. Griffith, and Michael E. Greenberg. 2002. Regulation of transcription factors by neuronal activity. *Nature Rev. Neurosci.* 3(12):921–31.

Whitelaw, V. A., and J. D. Cowan. 1981. Specificity and plasticity of retinotectal connections: A computational model. *J. Neurosci.* 1(12):1369–1387.

Willshaw, D. J., and C. von der Malsburg. 1976. How patterned neural connections can be set up by self-organization. *Proc. R. Soc. Lond. B* 194:431–445.

Wong, Rachel O. L. 1999. Retinal waves and visual system development. *Annu. Rev. Neurosci.* 22:29–47.

Yates, Paul A., Alex D. Holub, Todd McLaughlin, Terrence J. Sejnowski, and Dennis D. M. O'Leary. 2004. Computational modeling of retinotopic map development to define contributions of EphA-ephrinA gradients, axon-axon interactions, and patterned activity. *J. Neurobiol.* 59:95–113.

Yates, Paul A., Adina L. Roskies, Todd McLaughlin, and Dennis D. M. O'Leary. 2001. Topographic-specific axon branching controlled by ephrin-As is the critical event in retinotectal map development. *J. Neurosci.* 21(21):8548–8563.

Yuan, Li-Lian, J. Paige Adams, Michael Swank, J. David Sweatt, and Daniel Johnston. 2002. Protein kinase modulation of dendritic $K^+$ channels in hippocampus involves a mitogen-activated protein kinase pathway. *J. Neurosci.* 22(12):4860–4868.

Zhang, Li I., W. Tao, Huizhong, Christine E. Holt, William A. Harris, and Mu-ming Poo. 1998. A critical window for cooperation and competition among developing retinotectal synapses. *Nature* 395:37–44.

Zheng, James Q. 2000. Turning of nerve growth cones induced by localized increases in intracellular calcium ions. *Nature* 403:89–93.

Zheng, James Q., Ji-jun Wan, and Mu-ming Poo. 1996. Essential role of filopodia in chemotropic turning of nerve growth cone induced by a glutamate gradient. *J. Neurosci.* 16:1140–1149.