A Multiple-Mechanism Developmental Model for Defining Self-Organizing Geometric Structures

Thesis by

Kurt W. Fleischer

In Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

California Institute of Technology
Pasadena, California

1995

(Defended May 22, 1995)

Abstract

This thesis introduces a model of multicellular development. The model combines elements of the chemical, cell lineage, and mechanical models of morphogenesis pioneered by Turing, Lindenmayer, and Odell, respectively. The internal state of each cell in the model is represented by a time-varying state vector that is updated by a differential equation. The differential equation is formulated as a sum of contributions from different sources, describing gene transcription, kinetics, and cell metabolism. Each term in the differential equation is multiplied by a conditional expression that models regulatory processes specific to the process described by that term.

The resulting model has a broader range of fundamental mechanisms than other developmental models. Since gene transcription is included, the model can represent the genetic orchestration of a developmental process involving multiple mechanisms.

We show that a computational implementation of the model represents a wide range of biologically relevant phenomena in two and three dimensions. This is illustrated by a diverse collection of simulation experiments exhibiting phenomena such as lateral inhibition, differentiation, segment formation, size regulation, and regeneration of damaged structures.

We have explored several application areas with the model:

Synthetic biology. We advocate the use of mathematical modeling and simulation for generating intuitions about complex biological systems, in addition to the usual application of mathematical biology to perform analysis on a simplified model. The breadth of our model makes it useful as a tool for exploring biological questions about pattern formation and morphogenesis. We show that simulated experiments to address a particular question can be done quickly and can generate useful biological intuitions. As an example, we document a simulation experiment exploring inhibition via surface chemicals. This experiment suggests that the final pattern depends strongly on the temporal sequence of events. This intuition was obtained quickly using the simulator as an aid to understanding the general behavior of the developmental system.

Artificial evolution of neural networks. Neural networks can be represented using a developmental model. We investigate the use of artificial evolution to select equations and parameters that cause the model to create desired structures. We compare our approach to other work in evolutionary neural networks, and discuss the difficulties involved.

Computer graphics modeling. We extend the model to allow cells to sense the presence of a 3D surface model, and then use the multicellular simulator to grow cells on the surface. This database amplification technique enables the creation of *cellular textures* to represent detailed geometry on a surface (e.g., scales, feathers, thorns).

In the process of writing many developmental programs, we have gained some experience in the construction of self-organizing cellular structures. We identify some critical issues (size regulation and scalability), and suggest biologically-plausible strategies for addressing them.

Availability

Paper copies of the thesis. Paper copies of this thesis will be available after August, 1995 by U.S. mail from:

Computer Science Library
Caltech M/S 256-80
Pasadena, CA 91125

The Computer Science library also maintains an online version on the World Wide Web. The URL of the index of available technical reports is:

file://cs.caltech.edu/tr/INDEX.html

Color images and animations. This thesis contains many color images, which do not reproduce well in black and white. I am looking into the possibilities for distributing color versions of the thesis, but it is difficult due to the expense of color copies.

If you have a color printer, you can download and print the color postscript files from the CS library site (above) or my Web page (below). If you have access to a color monitor and can get to the World Wide Web, you can peruse the images online. I have not yet put all of the color images online, but I intend to. The Web site to examine is:

http://www.gg.caltech.edu/~kurt/avail.html

Animations produced by the simulator may someday be on the Web site. A short suite of animations produced with the simulator are also available as part of the 1994 Siggraph Video Review [Fleischer, 1994].

Code availability. I may distribute the code as binary or source at a later date. If you are interested in using it, send me some email describing what you intend to do with the code, and (briefly) something about your background (biology, computer science, etc). It will also be helpful to me to know how much experience you have with simulators, C++, etc. The code was written as part of a research project; as such, it is not particularly user-friendly. It may require some expertise and patience to use fruitfully. It is possible that a revised version will be written with a clearer user-interface; if so, a notice will appear in my Web page.

E-mail. I can be reached via e-mail at kurt@mailhost.gg.caltech.edu.

Acknowledgements

Thanks to my advisor, Al Barr, for his generosity and encouragement. He has provided a fertile environment for research and exploration, and I have greatly benefitted from our discussions on many topics in mathematics, biology, modeling, and computer graphics.

My fellow students in the Graphics Group have been a pleasure to work with. They have provided a source of intellectual discussions and camaraderie that I have truly enjoyed during my stay at Caltech: Ronen Barzel, Bena Currin, Dan Fain, Jeff Goldsmith, Devendra Kalra, Tim Kay, Dave Kirk, David Laidlaw, Mark Montague, Preston Pfarner, John Platt, John Snyder, Brian Von Herzen, Erik Winfree, Adam Woodbury, and Denis Zorin.

Thanks to the following people who patiently read and commented on early versions of the thesis: Cindy Ball, Ronen Barzel, Bena Currin, Dan Fain, Mark Montague, Dian De Sha, and Erik Winfree.

My appreciation to the staff of the Graphics Group who kept things running smoothly through chaotic times: Matt Avalos, Cindy Ball, Carolyn Collins, Allen Corcorran, Louise Foucher, Alf Mikula, Sandra Reyna, Dian De Sha, and Gail Stowers.

The following biologists kindly took the time to discuss a variety of biological issues with me: Arik Mercer, Ajay Chitnis, Scott Fraser, Bill Trevarrow, Andres Collazo, George Striedler, Katherine Wu, John Shih, and Steve Potter.

I'd like to express my appreciation to the the members of my thesis committee for their guidance and helpful discussions: Alan Barr, Scott Fraser, John Hopfield, Jim Kajiya, and Przemek Prusinkiewicz.

My fond appreciation to Professor Andries van Dam of Brown University who helped get me started in research as an undergraduate and has remained a source of encouragement ever since.

This work was supported in part by grants from Apple, DEC, Hewlett Packard, and IBM. Additional support was provided by NSF (ASC-89-20219) as part of the NSF/ARPA STC for Computer Graphics and Scientific Visualization, by the DOE (DE-FG03-92ER25134) as part of the Center for Research in Computational Biology, the Beckman Foundation, the Parsons Foundation, the National Institute of Health as part of their Training Grants program, and by the National Institute on Drug Abuse and the National Institute of Mental Health as part of the Human Brain Project. All opinions, findings, conclusions, or recommendations expressed in this document are those of the authors and do not necessarily reflect the views of the sponsoring agencies.

Contents

Index of Figures

Index of Equations

5.1 $$dz_{split}/dt \mathrel{+}= (\mathsf{env[surf\_RG]} \mathrel{\tilde{<}} \alpha)\,(1.1\ \mathit{split\_threshold})$$ 64

Split condition for R cells in hierarchical structure simulation.

5.2 $$\begin{aligned} \mathsf{R}_r(r,g) &= \gamma\,(a - r + r^2 g) \\ \mathsf{R}_g(r,g) &= \gamma\,(b - r^2 g) \end{aligned}$$ 67

Reaction functions for the reaction-diffusion patterns.

Ch. 6. Synthetic Biology

6.1 $$\begin{aligned} \tfrac{dz_d}{dt} &\mathrel{+}= -20\tfrac{e_r}{e_n} + 10\tfrac{z_d^2}{1+z_d^2} - 0.5z_d + 13 \\ \tfrac{dz_l}{dt} &\mathrel{+}= 0.95\tfrac{e_r}{e_n} \\ \tfrac{dz_l}{dt} &\mathrel{+}= (z_d \mathrel{\tilde{>}} 3)\,z_d \\ \tfrac{dz_l}{dt} &\mathrel{+}= -z_l \\ \tfrac{dz_r}{dt} &\mathrel{+}= 5 - z_r \\ \tfrac{dz_n}{dt} &\mathrel{+}= 1 - z_n \end{aligned}$$ 86

Cell state equations for experiment with inhibition via surface chemicals.

Ch. 7. Artificial Evolution of Neural Networks

7.1

cell state equation : $\frac{dz_{emit\_b}}{dt} \mathrel{+}= ((z_{ctype} \mathrel{\tilde{>}} 1.5)\ \tilde{\mathsf{or}}\ (z_{ctype} \mathrel{\tilde{<}} 0.5))\,(2.3 - z_{emit\_b})$

LISP expression : (stmt

condition : (or ($>$ ctype 1.5) ($<$ ctype 0.5))

index : emitb

consequent : ($-$2.3 emitb))

102

Example of a cell state equation and its LISP expression-tree equivalent.

Ch. 9. Summary and Discussion

9.1 $$\text{if cond then } dz/dt \mathrel{+}= r(s - z)$$ 133

Rate/setpoint style of cell state equation.

9.2 $$\begin{aligned} &\text{if cond}_1 \text{ then } dz/dt \mathrel{+}= r_1(s - z) \\ &\text{if cond}_2 \text{ then } dz/dt \mathrel{+}= r_2(s - z) \end{aligned}$$ 133

Combining rate/setpoint equations.

9.3 $$\begin{aligned} &\text{if true then } dz/dt \mathrel{+}= -z \\ &\text{if cond}_1 \text{ then } dz/dt \mathrel{+}= s_1 \\ &\text{if cond}_2 \text{ then } dz/dt \mathrel{+}= s_2 \end{aligned}$$ 133

'Separated equations' style of cell state equation.

9.4 $$\begin{aligned} &\text{if cond}_1 \text{ then } dz_{die}/dt \mathrel{+}= \alpha \\ &\text{if cond}_2 \text{ then } dz_{die}/dt \mathrel{+}= -z_{die} \end{aligned}$$ 133

'Accumulation equations' style of cell state equation.

9.5 $$\text{if cond}_3 \text{ then } dz_{die}/dt \mathrel{+}= -\alpha$$ 134

Another accumulation equation.

Chapter 1

Introduction

In this thesis I present a multiple-mechanism developmental model and a simulation of the model that enables a user to define the detailed behaviors of individual cells in a developing multicellular organism. The model has many areas of application, some of which are explored here. Before describing the model in detail, I'd like to retrace the path that led me into developmental modeling. The progression of ideas may prove helpful to others working in this area, and it has influenced my thinking about such developmental models as well as my goals in creating this particular model.

This work began as an exploration into the feasibility of automatically creating artificial neural networks with geometric structure and heterogeneous neural types. The approach uses an automated search algorithm to seek networks that perform better on certain classes of problems, where geometry provides an important part of the neural computation.

Any sort of automated search algorithm requires specification of a *search method* and a *representation scheme* (with parameters to vary). The search method is a method of choosing between candidate solutions, for instance an objective function in optimization, or selection criteria in evolution. Optimization algorithms are typically suitable for problems with continuously varying parameters, whereas networks are more conveniently represented with a combination of continuous *and* discrete parameters (number of neurons, connections that may or may not exist, instantiation of subnetworks, etc.) Because networks structures can vary in a discrete fashion, I chose to focus on artificial evolution to perform the automated search.

A representation scheme is a choice of a set of parameters and their mapping into a candidate solution. This choice is critical for success with genetic algorithms or artificial evolution, just as is true for neural net learning or optimization algorithms. The representation determines the nature of the configuration space. [1]

Hence I devoted a lot of time to devising an appropriate representation scheme for the artificial evolution. After several attempts that were lacking, I began to think that having a developmental process might be helpful for evolution. Many artificial evolution algorithms simply evaluate an objective function on the representation scheme (a *direct encoding scheme* [Yao, 1993]). For example, the network could be encoded by a list of nodes and connections. My conjecture was

[1] In the fields of genetic algorithms and artificial evolution, the representation scheme is often called a 'genome' or 'genotype'. However, the term representation scheme is less likely to cause confusion in the biological portions of this thesis, and follows the usage of [Koza, 1992] and others.

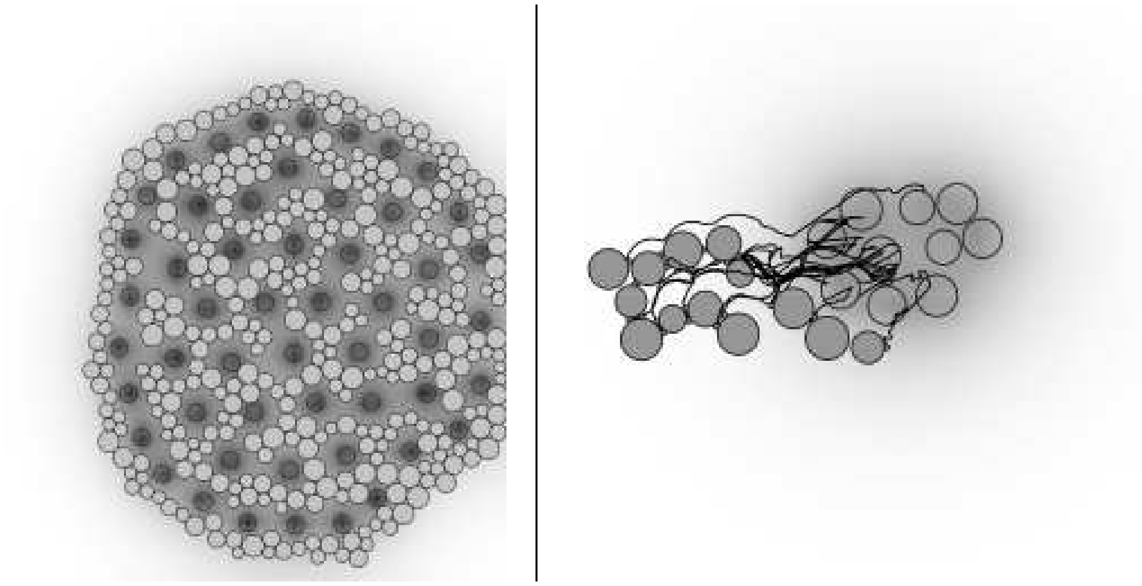


Figure 1.1: These are results from running the multiple-mechanism developmental simulation on two different sets of cell state equations. In the left figure, two cell types interact to form a compartmental structure, with single cells of one type surrounded by many cells of a second type (details in Section 5.2.2).
In the right figure, neurites grow from the cells on the right towards the cells on the left (details in Section 5.3.3).
The cells (depicted as circles) are capable of migration, differentiation and adhesion, and they interact with the continuously diffusing chemicals in the surrounding extracellular environment.
The concentrations of diffusing chemicals are shown here as different gray levels. □

that a developmental process might aid by providing two properties: *robustness* and *developmental gain*. The adaptive nature of development can provide robustness to some deleterious mutations, and still produce viable organisms. I use the term developmental gain to refer to the possibility of a large-scale change in behavior or structure based on a small change to the representation. An example of developmental gain is the ability to create another layer of neurons or duplicate an entire neural module with a relatively small genetic change. A representation with developmental gain is likely to make it easier to explore the large space of possible networks.

Thus I chose to focus on developmental models for creating the networks. The only existing examples of evolving developmental systems are in the living creatures around us, so it seems reasonable to take cues from biological systems wherever possible. A sufficiently detailed biological model could reproduce the desired behaviors, but it is difficult to know how much detail is required and whether it will be computationally feasible.

This brought me to the following question: *What features of a representation scheme and a developmental process are important for describing a wide range of structures, and also provide the robustness and developmental gain properties?* This question has been the driving force behind the creation of the multiple-mechanism developmental model described in this thesis. It is tempered with a concern for the computational feasibility of the model's simulation.

The result of investigating this question has led to the creation of a developmental model and simulator that are capable of representing many biological phenomena, and hence are of use in several applications. The model includes a broad range of developmental mechanisms, sufficient to reproduce developmental behaviors that give rise to complicated structures.

I chose the representation scheme to be a set of differential equations that describe intracellular processes and environmental characteristics. The equations govern intracellular processes such as gene transcription, regulation, translation, cell metabolism, etc. I'll refer to these collectively as the **cell state equations**. The right-hand-side of each equation is an expression tree whose structure and parameters can be modified by an artificial evolution algorithm [Koza, 1992, Sims, 1991a].

The cell state equations define the state changes occurring inside the cells. They are described in a particular form that makes them convenient for representing regulatory elements and combining gene products. I regard this specification as one of the major contributions of the thesis. It is described in detail in Chapter 2.

The simulator is programmable; the cell state equations can be specified by the user or by an automated search algorithm. In order to perform optimization or artificial evolution on the cell state equations, we present different sets of cell state equations to the simulator and evaluate the resulting synthetic organism after the simulation has computed the develompental process. Alternatively, a user can enter the cell state equations and observe what occurs.

A user can investigate characteristics of developmental systems by creating and modifying cell state equations to describe various synthetic organisms (Figure 1.1). Because of the general structure of the system, the simulator has some characteristics not found in other developmental simulators. It is intentionally programmable, allowing the user to enter arbitrary cell state equations, and it is general, including a wide range of developmental mechanisms. The simulator is a flexible tool that enables biological explorations one might not otherwise undertake, a computational complement to gedanken experiments (Synthetic Biology, Chapter 6).

1.1 Overview of the Contributions of the Thesis

This outline lists the contributions of the thesis, and where to find them. The thesis is divided into two major parts. The first describes the developmental model and shows examples of its use. The second part examines three applications.

The Multiple-Mechanism Developmental Model:

- Multiple Developmental Mechanisms in a Unified System (Chapters 2 and 3):
  - ◇ discrete motile cells
  - ◇ cell-cell interactions: chemical, cell lineage, mechanical, etc.
  - ◇ intracellular processes: gene transcription and regulation, kinetics, and metabolism
  - ◇ model of cell surface proteins (homophilic and heterophilic)
  - ◇ cell lineage and cell cleavage plane control mechanisms
  - ◇ three-dimensional implementation (Section 5.4)
- Form of the Intracellular State Equations (Chapter 2):
  - ◇ based on differential equations
  - ◇ superposition of contributions to same state variable
  - ◇ regulatory terms (using continuous conditionals)
- Experiment Gallery Exhibits Emergent Behaviors (Chapters 4 and 5):
  - ◇ biological phenomena
  - ◇ robustness and development gain
  - ◇ heterogeneous asymmetric neural networks
  - ◇ size regulation and scalability
  - ◇ programming in a genome-like language

Applications:

- Synthetic Biology (Chapter 6):
  - ◇ computer aid for generating intuitions about biological systems
  - ◇ limitations of continuous reaction-diffusion models
  - ◇ archival representation for biological information
- Artificial Evolution of Neural Networks (Chapter 7):
  - ◇ Review of related work and comparision with this approach
  - ◇ Status report, discussion of difficulties with the approach
- Cellular Texture Generation (Chapter 8):
  - ◇ computer graphics application
  - ◇ automatic generation of model complexity

Discussion (Chapter 9):
- ◇ issues in developmental modeling
- ◇ experience programming in a genome-like language
- ◇ experience building simulators

1.2 Related Work in Developmental Modeling

[2] Much of the early work in developmental modeling focused on modeling a particular mechanism and then showing the range of patterns that could be achieved using that mechanism in isolation. Reaction-diffusion models, mechanical models, and grammar-based models, all have been shown to be capable of describing many aspects of pattern formation and morphogenesis.

Even to modelers, it has long been evident that the *interactions* of these mechanisms are critical to pattern formation [Turing, 1952]. Recently, the number of mathematical and computational models that incorporate multiple mechanisms is growing (e.g., mechanochemical models). This is partly due to the availability of more powerful computers, and partly due to the advances in experimental science that provide more detailed data about developmental processes, hence provoking more detailed questions.

The multiple-mechanism developmental model described in Chapter 2 contains a broader spectrum of developmental mechanisms than any other I am familiar with. Although some of the mechanisms are modeled fairly simply, it provides a framework into which other models can fit. For example, the current implementation uses a simple mechanical model of spherical cells subject to forces from collision and adhesion, in a viscous environment. More detailed mechanical models have been described by several researchers [Odell et al., 1981, Oster et al., 1990]. One of these models can be incorporated into this framework, enabling the mechanical deformations to occur under genetic control, or via the influence of chemical interactions.

Cell migration is an important aspect of development (especially neural development), so we explicitly model discrete cells that are free to move within the environment. This distinguishes our work from most previous work in reaction-diffusion systems and grammar-based systems that do not have this capability. Other models that do allow cell migration often do not include other mechanisms (chemical, genetic) which are critical to developmental pattern formation.

The references to related work are organized as follows:

- Single-mechanism models
  - ◇ Chemical models (reaction-diffusion) [Turing, 1952]
  - ◇ Mechanical models [Odell et al., 1981]
  - ◇ Grammar-based cell lineage models [Lindenmayer, 1968]
  - ◇ Intracellular chemical kinetics models [Michaelis and Menten, 1913]
  - ◇ Cell surface models
- Multiple-mechanism models
  - ◇ Mechanochemical model [Murray, 1993, Chapter 17]
  - ◇ Hybrid grammar-based models
  - ◇ Connectionist model
  - ◇ Cell Programming Language
- Summary

[2] Related work in artificial evolution is presented in Chapter 7, and, likewise, the related computer graphics work is described in Chapter 8.
There is a large body of work in developmental biology. A comprehensive review of this work is not attempted here. Several textbooks have been quite helpful in building the model[Gilbert, 1991, Alberts et al., 1989, Darnell et al., 1990, Purves and Lichtman, 1985, Brown et al., 1991, Berg, 1983], as well as course notes from Caltech, and many articles.

Single-Mechanism Models

Chemical models. Turing's paper *The Chemical Basis of Morphogenesis* proposed a mathematical theory of cell-cell interaction via chemical substances (*morphogens*) [Turing, 1952]. He showed that these *reaction-diffusion* systems could exhibit stable patterns, and proposed this as a possible mechanism for pattern formation in development. This has formed the basis of much work on developmental modeling using reaction-diffusion equations, such as that by Meinhardt [Meinhardt, 1982, Murray, 1993], the chemotaxis models of *dyctyostelium* slime molds [Lin and Segel, 1974], and creating patterns such as those of the zebra coat [Bard, 1981], butterfly wings [Murray, 1981], etc.

Mechanical models. In *The Mechanical Basis of Morphogenesis*, Odell et al. discuss how a mechanical model can account for gastrulation, neural tube formation, and eversion behaviors such as that observed in Volvox [Odell et al., 1981]. In their two-dimensional system, each cell is modeled as a quadrilateral which is attached to adjacent cells in an epithelium. The cell membrane of each cell is modeled as several springs with variable spring constant and rest length. The spring parameters are modified based on interactions between adjacent cells, and this gives rise to the various behaviors.

Cell adhesion models. Later work with mechanical models uses more detailed mechanical models with more complex cell shapes, and incorporates cell adhesion as well. In [Oster et al., 1990, Weliky and Oster, 1990], each cell has a polygonal shape (in three dimensions), and is subject to forces from osmotic pressure as well as those arising from an elastic membrane model.

A different approach is taken by [Glazier and Graner, 1993]. They use a discretized spatial model (a *lattice*) in which a single cell can cover several adjacent lattice points. This enables cells to assume arbitrary shapes (up to the resolution of the grid). This model elegantly describes behaviors arising from differential adhesion [Steinberg, 1964].

Cell lineage models: L-systems and grammars. Grammar-based techniques such as Lindenmayer-systems (L-systems) [Lindenmayer, 1968] are convenient for describing cell lineage and genetic control of cell division. These systems use rewrite rules to sequentially modify strings which represent organisms, and are capable of creating realistic-looking models of biological structures. L-systems have been particularly successful for modeling plants [Prusinkiewicz and Lindenmayer, 1990, Lindenmayer and Prusinkiewicz, 1989], and have also been applied to modeling cell layers [Prusinkiewicz and Lindenmayer, 1990, de Boer et al., 1992, de Boer, 1989], topology of neural networks [Kitano, 1990], and other systems. The results of these systems underscore the importance of cell lineage and the orientation of the cleavage plane in determining the shape of an organism.

Models of neural development and activity. Electrical activity affects the development of neural structure in many ways. For example, it is thought that correlated firing between neighboring axons can affect their destinations [Fraser and Perkel, 1990], and that synapse formation can be strengthened in some cells when the firing of the input and output cells are correlated [Hebb, 1949].

Fraser and Perkel proposed a developmental model of the neural map between the retina and the tectum in the visual system of lower vertebrates. This model incorporates several different mechanisms, involving modulation of cell and neurite adhesion, competition for space, and activity dependent processes (which depend on the firing of neurons), and can reproduce a variety of observed experimental data.

Multiple-Mechanism Models

The mechanochemical model. The *mechanochemical model* [Murray, 1993, Chapter 17] hinges on two assumptions: cells migrate within an extracellular matrix, and cells can generate large traction forces. A continuum model is created from the conservation of cell population density and extracellular matrix density, together with a force balance law between the cells and the matrix. This model is then combined with a reaction-diffusion model to create a combined mechanical/chemical system.

There are some drawbacks of a continuum model. It cannot conveniently represent behavior at the resolution of individual cells. For example, a pattern with every other cell having a different state cannot be represented easily. Patterns at the level of individual cells are important even in repeating structures, such as the canonical microcircuits of cortical layers.

Hybrid grammar-based systems. Some grammar-based models incorporate environmental influences and cell-cell interactions using context-sensitive languages [Lindenmayer, 1968, Lindenmayer and Prusinkiewicz, 1989]. Other grammar-based models have been enhanced to include computation of physical forces for the modeling of cell layers [de Boer et al., 1992, Prusinkiewicz and Lindenmayer, 1990]. Between cycles of cell division, adjacent cells apply forces to each other, changing shape until equilibrium is achieved. The cell walls are modeled as linear springs (similar to Odell's model [Odell et al., 1981]), and the cells expand and contract due to an approximation of osmotic pressures.

This sort of hybrid system which incorporates both grammatical elements and differential equation elements seems to be an effective way to make computationally feasible models which incorporate physical behaviors [Prusinkiewicz et al., 1993].

The connectionist model. Mjolsness et al., among others, have noted the similarities between gene regulation and standard neural net dynamics [Mjolsness et al., 1991, Kauffman, 1993]. They also use a hybrid method, in which differential equations are used to describe the dynamics of gene interactions at a short time scale, and a grammar-based model describes cell state changes at a longer time scale (e.g., state in the cell cycle: mitosis[3], interphase, post-mitotis). The differential equations are of the standard connectionist form [Hopfield, 1984]: $\tau_a(\mathrm{d}v_i^a/\mathrm{d}t) = g_a(\sum_b T^{ab}v_i^b + h^a) - \lambda_a v_i^a$. The v_i^a are the components of the state vector for each cell i, g_a is a sigmoidal threshold function, T^{ab} are the components of a connection matrix, and h^a are offsets.

The *connectionist model* is similar to ours in many ways, since it has a differential equation model of gene expression, it has a representation for cell differentiation, and it has mechanisms for cell-cell interaction. They, too, propose a general modeling framework which encompasses multiple mechanisms. Their geometric model is currently special-purpose, designed for a particular experiment. In our system, we have more thoroughly explored general geometric mechanisms (cell collision and adhesion, sensing and emitting diffusing chemicals, etc.), which enable a greater range

[3] cell division

of cell-cell and cell-environment interactions. The differences in the intracellular representations are:

	Connectionist Model	Our Model
cell state changes:	grammar rules and ODEs	conditional ODEs (Section 2.5)
type of ODE:	connectionist (fixed)	arbitrary (user-specified, Section 2.5)

The connectionist model's hybrid representation, combining grammar rules and differential equations, is a quite attractive option for representing cell state changes efficiently. The use of a particular differential equation type enables parameter estimation for model-matching. It has been successfully applied to the early developmental genetics of *Drosophila* [Reinitz et al., 1992], by optimizing the parameters of the model to match biological data (the optimized parameters are T^{ab}, h^a, and parameters associated with synthesis, decay, and diffusion rates).

The Cell Programming Language. The *Cell Programming Language* [Agarwal, 1993] combines elements of discrete event simulation, cellular automata, and the lattice models. It makes several simplifying assumptions which enable it to compute simulations with a few thousand cells. Time and space are discretized, and direct interactions between cells is allowed (i.e., each cell can directly react to the internal state of a neighbor, as in a cellular automaton). The discretized space is a lattice similar to that of [Glazier and Graner, 1993]. Each cell's genome consists of a set of states for the phases of the cell (pre-mitotic, post-mitotic, etc.), and associated with each state is a sequential list of instructions. As in a discrete event simulation, time is discrete, and during each time step every cell executes its instruction list. The state of each cell is directly available to neighboring cells, so they can modify or react to the state of their neighbors. This system has been applied to modeling cellular sorting by differential adhesion, aggregation in slime molds, and other phenomena.

The *Cell Programming Language* supports all of the mechanisms of our model, with the exception of our extension to neural structures (Section 2.8). However, our representations differ substantially. The main differences are: (i) their use of an instruction list executed in discrete time steps to represent intracellular state, as opposed to our differential equation based system, and (ii) their discretization of space and time versus our continuous model of the extracellular processes and continuous model of intracellular state over time. The instruction list representation is somewhat less related to biological reality than our differential equations, yet it does provide substantial functionality. Powerful programming constructs such as loop statements can be used within the intracellular computation. Their use of a lattice to allow arbitrary cell shape is more general than our parameterized geometric primitive (circle or sphere). We believe the lattice representation is superior for two-dimensional systems, but for the generalization to three dimensions it is unclear if it can be made computationally feasible. Some other deformable geometric primitive may be the best compromise.

Summary. There is a growing trend toward multiple-mechanism developmental models. Current workstations can compute the activities of hundreds or a few thousands of cells using these models, permitting the investigation of multicellular behaviors.

We are still in the initial, exploratory phase of this research. Although we can compare the current state of each approach as presented in the published papers, it is more difficult to compare the approaches on a more fundamental level to determine which are most likely to be successful in the long run.

The hybrid approach of combining grammar models with differential equations is very promising. The use of conditional terms in the differential equations (Section 2.5) adds an important regulatory ability, and is compatible with the hybrid approach. Shape models such as the lattice model are quite attractive for two dimensions, but do not scale trivially to three dimensions, so further work is required in that area. The future challenges for multiple-mechanism representations are centered around the desire for

- ◇ efficient and accurate cell shape models in three dimensions,
- ◇ dealing with cell behaviors at different time scales, and
- ◇ maintaining biological relevance.

PART I

THE DEVELOPMENTAL MODEL

Description of the formulation of the developmental model.

Chapter 2

A Multiple-Mechanism Developmental Model

In our exploration of morphogenesis and pattern formation, we have chosen to focus on the patterns arising from interactions between cells as they express their genetic programs in a dynamic environment. Chemical, mechanical, geometric and cell contact interactions are included to create a system with the capability to reproduce a wide variety of developmental behaviors.

Our computational goal is to compute simulations of hundreds or thousands of discrete cells in reasonable amounts of time (minutes or hours).[1] To this end, we model cells as the fundamental units of development, and mostly ignore detail below the level of the cell, although we do include a simple model of neurites and growth cones. Generalizations to other subcellular structures are discussed as well.

The mathematical and computational framework is built upon time-varying objects which interact in a common geometric space. The behavior of each object is specified by a set of differential equations which modify its state based on the current state and other geometrically local information (e.g., contact with other objects). The implementation uses these objects to represent cells. The extracellular environment is another time-varying object which computes spatially varying fields such as chemical diffusion, and also mediates interactions between cells (such as collisions and adhesion). This framework is quite general and can be applied to other domains as well.

The contributions described in this chapter [2] and the next are:

- (i) the inclusion of genetic, chemical, mechanical and geometric mechanisms in a unified developmental model of discrete cells (Section 2.3),
- (ii) the general object framework for simulating interacting geometric objects (Section 2.3),

[1] The time estimates given here are informal because actual computation time depends on many simulation parameters. Experience with other simulators has taught us that there is a 'patience' limit which determines the complexity of a simulation which someone will actually bother to run. Depending on how often one wishes to change parameters, the simulation time of “a stretch,” “a cup of coffee,” “a good night's sleep,” may be appropriate. For a system such as this one, where the intent is to experiment with changing parameters, we are happiest when the experiments run on the order of minutes. This is often the case (see Chapter 5 for details).

[2] An early version of this work was published in [Fleischer and Barr, 1994].

- (iii) the form of the cell state equations which allows straightforward description of gene regulation, enzyme kinetics, and many other cell processes (Section 2.5), and
- (iv) the implementation using dynamic compilation and numerical simulation of piecewise continuous differential equations (Ch. 3).

A discussion of the strengths and weaknesses of the model appears in the Section 9.3, along with proposals for some extensions.

Chapter overview. We begin with a discussion of the goals, and the criteria by which we evaluate success or failure of the model (Section 2.1). Section 2.2 then presents the simplifications and idealizations of the real system which we make in creating the model abstraction, and why these are likely to achieve our goals. The high-level modular framework of the model is described in Section 2.3.

In Section 2.4, we show how the user can choose the meanings for a cell's state variables, using them to represent genes, proteins, sequestered chemicals, etc.

Section 2.5 presents the form of the intracellular state equations (the **cell state equations**). These equations define the state changes for both types of state variables (intracellular and membrane). They represent all intracellular processes (gene expression, enzyme activity, metabolism, protein transport to membrane, etc.).

A cell's intracellular state depends on information from the extracellular environment. We provide this information via an array of local environment variables described in (Section 2.6).

The remaining major component of our cell model is cell behavior. Given the current intracellular state and the local extracellular environment, the cell behaves in a certain way. In our system, this is determined by the cell behavior functions (Section 2.7).

For simplicity of presentation, the neurites and growth cones are not discussed in the initial sections describing the model. They are covered in Section 2.9, along with a discussion of possible generalizations to include other subcellular structures.

An implementation of the model using piecewise continuous differential equations is described in more detail in Ch. 3. In Section 1.2, we compare this model with other developmental models, and discuss possible extensions and generalizations.

2.1 Goals and Evaluation Criteria

Any model is a simplified, idealized representation of a real system. Since a model must differ from the real system in some aspects, it is only a valid representation in certain domains. To evaluate a model, we must consider it in the context of the objectives of the problem solving enterprise.

The objectives of this project are to investigate the formation of structure via developmental processes. Pattern formation (spatial ordering of differentiated tissues) and morphogenesis (the creation of shape) are two important facets of the creation of the structure of an organism. It is our goal that the proposed model be capable of representing many motifs of pattern formation and morphogenesis, as well as exhibiting the properties of robustness and developmental gain (Chapter 1). In addition, we require that the simulator be programmable, allowing a user to enter the intracellular state equations and the equations describing interactions of chemicals in the extracellular environment.

Abstraction	Implementation
Discrete cells (capable of migration)	
cell shape (geometry)	2d circles/3d spheres
subcellular structures	
growth cones	modeled as small cells
neurites	path of growth cone and communication link between cell and growth cone
Intracellular Activity	
metabolism, kinetics, etc.	ODEs
gene expression, regulation, transport	ODEs with conditionals
Genetic/Cell Lineage	
genetic control of cell operations	ODEs with conditionals
inherit state from parent cell	yes
control orientation of cell divisions	yes
asymmetric cell division	partially implemented
Extracellular environment	
chemical	2d reaction-diffusion grid
mechanical	mechanical barriers, viscous drag, collisions
Cell-cell interactions	
mechanical	collisions and adhesion between cells
chemical (membrane proteins)	adhesion and contact recognition (homophilic and heterophilic)
electrical (gap junction, synapse)	discrete spiking neuron model
Cell-environment interactions	
chemical	emit, absorb, sense values in grid
mechanical	cell-env collisions and adhesion

Figure 2.1: This table presents the real-world characteristics which we include in the model. The model combines many developmental mechanisms into one system. The current implementation of the model is shown on the right. □

It is our intent to create a system with the large scale characteristics of multicellular development (i.e., with the right "look and feel"). The success or failure of the model will depend on its ability to

◇ reproduce known developmental behaviors (Chapter 5),
◇ reproduce results of other developmental models (Chapter 5),
◇ demonstrate the properties of robustness and developmental gain (Chapters 1 and 4),
◇ provide intuitions about development (Chapter 6), and
◇ create "life-like," plausible animations of developing organisms. See video [Fleischer, 1994].

The novelty of this system lies in its ability to combine behaviors arising from different mechanisms. We have drawn from many previous developmental models, especially [Odell et al., 1981, Turing, 1952, Lindenmayer, 1968] (Section 1.2). To the extent to which we include elements of these models, we are able to reproduce the range of developmental behaviors that they exhibit. In addition, we can explore the possibilities of interactions between the chemical, mechanical, and genetic mechanisms.

2.2 The Abstraction

Following the goals outlined above, we abstract the following features of biological development as being critical to morphogenesis and pattern formation:

1. **boundary:** a cell has a boundary surface with some shape and location,
2. **chemicals:** chemicals exist within the cell, on the cell surface, and in the extracellular space,
3. **sensors:** a cell can sense local information from the extracellular environment (amount/gradient of chemicals, light, etc.), and the proportion of its surface chemicals which are bound to complementary chemicals,
4. **cell lineage:** a cell can control the orientation of its next cleavage, as well as the initial states and cell types of its immediate children (perhaps asymmetrically),
5. **equations of motion and shape change:** a cell is capable of exerting forces to move and change shape, which are mediated by viscous dynamics, collisions with other cells and obstacles, and adhesion due to binding of surface chemicals, and
6. **death:** a cell can cause its own death.

This would be our ideal abstraction. Due to the difficulties of modeling an arbitrary cell shape, and to the arbitrary distributions of chemicals in and on the cell, the current implementation of the model is a restricted version of that abstraction (Figure 2.1). Cell shapes are modeled with circles (2d) or spheres (3d). Each intracellular chemical is represented as a single value (thus it is assumed to be uniformly distributed through the cell). Each surface chemical is also assumed to be uniformly distributed on the boundary of the cell. The effects of these restrictions are discussed in detail in Chapter 9.

Justification. Why should we have any faith that this abstraction will work? We have included, in some form, many of the mechanisms which biologists have recognized as important in development. For instance, each of the processes mentioned in this quote from Gilbert [Gilbert, 1991, page 523] are incorporated into our modeling framework, although in a simplified form (as will be described in the following sections).

> *"Indeed, it is generally thought that morphogenesis is brought about through a limited repertoire of cellular processes:*
>
> 1. *the direction and number of cell divisions;*
> 2. *cell shape changes;*
> 3. *cell migration;*
> 4. *cell growth;*
> 5. *cell death; and*
> 6. *changes in the composition of the cell membrane and extracellular matrix."*

We have also concentrated on including each of the four basic means by which multicellular organisms can develop. Once we have agreed to assume the cell doctrine, then combinations of the following processes are the only means for generating multicellular structures:

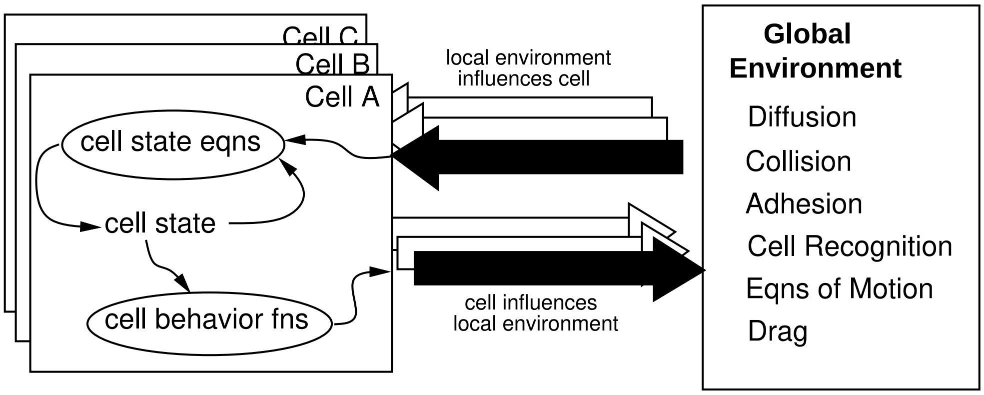


Figure 2.2: **Conceptual Organization of the Model**: Each cell's state is modified by the *cell state equations* (the genome, metabolism, and other intracellular processes). These equations may depend on parameters drawn from the cell's local environment (e.g., concentration of a diffusing extracellular chemical). The local environment information is computed from the processes in the global environment evaluated at the cell's location. The cell's state determines the cell's behavior via the *cell behavior functions*. The behavior can then affect the environment (e.g., by releasing a diffusable chemical, or by exerting a motile force and causing a collision with a neighboring cell).
The cell state equations and behavior functions are defined by the user (shown in the ovals). □

- ◇ an initial state that is inherited from a cell's parent (e.g., mRNA),
- ◇ the effect of intracellular state changes,
- ◇ the effect of cell-cell interactions, and
- ◇ the effect of cell-environment interactions.

Each of these is present in our model.

Is cell fate determined by cell lineage or induction? This is a question that biologists often consider when examining a developmental system. Cells may be assigned a particular function by their lineage, or they may be influenced by the activities of nearby cells to take on a particular function (induction). There is clear evidence that either or both of these occur in various developmental systems [Gilbert, 1991]. In our abstraction, there are many cell-cell interaction mechanisms that can support induction, as well as a mechanism for a cell's initial state to be specified by its parent (cell lineage).

2.3 Conceptual Organization of the Model

From the abstraction, we now proceed to a concrete representation, as depicted in Figure 2.2. We will introduce several terms that are defined in both Section 2.3.1 and the Glossary.

Figure 2.2 shows the major components of the model and how they interact. Each cell has an array of state variables that represents its intracellular and surface chemicals. State changes are effected using differential equations (the **cell state equations**). Input from the cell's sensors are parameters to the cell state equations. The cell also has behaviors. The particular behaviors it exhibits are determined by the **cell behavior functions**.

This framework, the result of some thought and experimentation, has some attractive features in addition to its simplicity:

◇ The simulated cell only has access to information which a real cell would have (it does *not* know its own location, for instance, since that is computed by the equations of motion).
◇ Cell behaviors are effected via the environment.
◇ The modularity makes adding new environmental features straightforward.
◇ Cell behavior functions can be viewed as a model of protein function.

Perhaps this last item needs a bit of explanation. The cell behavior functions can be seen as 'structure-to-function maps' describing the function of a protein. In real biochemical systems, a protein or group of proteins has a specific function in the cell, which is a consequence of the molecular structure of the proteins involved. We are not modeling structural interactions of three dimensional proteins! Instead, we use a mathematical expression (the *cell behavior function*) to describe how a protein or group of proteins acts to perform some function, e.g., produce a locomotor force, release a diffusable chemical, adhere to another molecule, etc. Thus these functions serve as a mapping from the concentration of some protein to a behavior in the cell.

Cells exhibit both continuous and discontinuous behaviors (events such as dividing, emitting a growth cone, and dying). The timing of each event is determined by a cell behavior function; when the function crosses zero, the specified event occurs (cell divides, emits a growth cone, or dies).

Formulation as a system of PODEs. The simulation gathers the state variables from each cell and the state variables from the global environment into one large state vector. Likewise, the differential equations from every cell and from the global environment are gathered to form a large coupled system of differential equations. Since there are both continuous and discontinuous behaviors in the cells, it is actually a system of piecewise continuous ordinary differential equations (PODEs) [Barzel, 1992, Appendix C],[Shampine et al., 1991]. The partial differential equations arising from chemical diffusion in the extracellular environment are spatially discretized to form an ODE approximation, and thus also fit into this form (Section 3.2.6).

PODEs are solved similarly to ordinary differential equations (ODEs), except that when an event occurs (as signalled by an event function crossing zero), the solver is briefly halted and structures are created or destroyed (e.g., during cell division or cell death). Thus the solver must also find roots on the event functions while it is integrating the ODEs forward in time. PODEs are described in a bit more detail in Section 3.2.1.

Forms of cell interaction. Cells are only able to access local information such as the local concentration of chemicals (see Local Environment Variables, Section 2.6). They cannot directly access their absolute position and orientation in the world. Nor can they directly change their position, but do so only by applying forces which may be counteracted by other forces (e.g., collision with a wall).

Some state variables can be designated as *membrane* state variables which represent concentrations of membrane-bound molecules. The simulator computes interactions between these membrane-bound surface factors and those on neighboring cells which are in contact. The interactions between these molecules are specified by a table indicating the adhesive binding force between any pair of molecule types (Section 3.2.4). This allows for definition of homophilic or heterophilic adhesion molecules.

The inclusion of multiple mechanisms enables many forms of interaction between cells. Interactions between cells can occur *directly* from one cell to another or *indirectly* via some intermediate process. Examples of direct interaction are collision (applying a force) and contact recognition (changing the amount of a cell's artificial membrane protein which is in a bound state). Indirect cell interactions occur when a cell changes its local environment, which in turn is sensed by another cell. For example, one cell can emit a chemical into the environment which will then diffuse spatially. Another cell some distance away can sense and respond to that emitted chemical, thus reacting to the actions of the first cell in an indirect manner.

2.3.1 Definitions

cell A cell is modeled as a geometric shape (currently a circle or sphere, with optional neurites) with a given response to applied forces, as well as an array of cell state variables governed by cell state equations.

cell behavior functions (protein function model) The cell's current state determines what it is trying to do: the forces it is applying, events such as cell division, etc. The behavior functions compute all of the cell's forces and discontinuous events such as cell division from the cell state variables and the local environment variables (Section 2.7).

cell state equations (genome, metabolism, etc.) The state equations modify the cell's internal state variables based on the local environment and the cell's current state (Section 2.5).

cell state variables (`state[]`) An array of variables which loosely represent the amounts of proteins within the cell (other interpretations are discussed in Sections 2.4 and 2.5). The values of these variables affect the cell's movements, the timing of events, and the cell's interaction with the environment.

membrane (or surface) state variables These are state variables which represent concentrations of membrane-bound molecules (referring to the cell membrane). We use 'surface' and 'membrane' interchangeably in the text (Section 2.6 and Section 3.2.4).

equations of motion Given the cell's behavior functions which describe what the cell is *trying* to do, these equations compute the end results using viscous dynamics. For example, the cell may be applying forces to move to the right, but the collision-adhesion forces may counteract that movement, producing a net movement to the left or right (Section 3.2.3).

environment All of the simulated cells interact within a single global environment. The environment contains diffusing, reacting chemicals, as well as physical barriers. Within the simulation, cells access information about their environment locally through an array of *local environment variables*.

local environment variables (`env[]`) An array of variables which represent the local extracellular environment of a cell. The values available to the cell as a function of time. Since each cell is in a different location, in general the local extracellular environments of two cells will differ. These variations can then lead to different behavior for the cells, even though their genomes may be identical (Section 2.6).

Clarification on inputs/outputs for the cell state equations. From a cell's point of view, the cell state equations have read-only access to the local environment variables. They have read-write access to the cell state variables, including the membrane state variables. The cell behavior functions are write-only.

2.4 Associating Meanings with State Variables

One way to think of the state variables is as representing intracellular and surface chemicals. In fact, when first devising this model, we initially thought of them only as proteins, and the cell state equations as genes (the differential equations represent the rate of creation of the protein).

But we can take advantage of the powerful underlying differential equation formulation and consider the state variables to be much more general.

The state variables can be used to represent whatever quantities are appropriate for a given simulation experiment. We can use state variable to represent the concentration of a protein. Or we can use one to represent the stored ATP in a cell, or even a more general notion such as the stage in the cell cycle, the time in a cellular clock, or some less well-defined quantity like 'vitality.'

A limited concept of location within a cell. With the current abstraction, there is no geometry associated with the state variables within a cell. That is, a single state variable cannot represent a graded concentration of some chemical across the cell. However, it is possible to use several state variables to represent concentrations of chemicals in different conceptual locations or conditions, such as assigning one to "DNA for X," another to "nuclear RNA for X," and another to "cytoplasmic mRNA for X." But these descriptions exist only in the user's interpretation of the variables, and are not handled by the simulator as different spatial locations – each state variable is just a scalar value in the cell.

Notation. The set of cell state variables in a cell is represented as an array, and we assign appropriate names to individual state variables. In the implementation, this is done by having appropriately named integer variables as indices into the state array. For example, the state variable controlling the rate of emission of a diffusable chemical r is referred to as state[r], or sometimes z_r for brevity. The pair of state variables determining the motive force in two dimensions are referred to as (state[fx], state[fy]), or sometimes as the vector $\mathbf{z}_f$ for brevity.

Pre-defined state variables. Some state variables are defined by the system to have particular meanings. They are used for controlling cell behaviors, as described below in Section 2.7. For example, we define state[fsize] to control the rate of growth of the cell.

2.5 The Cell State Equations: Gene Expression, Metabolism, Etc.

We represent intracellular processes such as gene regulation, gene expression, metabolism, etc., as differential equations of a particular form, which we call the *cell state equations*. Differential equations are a natural choice for modeling natural time-varying phenomena. As Murray notes [Murray, 1993, page 124], differential equations enable us to represent processes for which we have qualitative knowledge even if we do not know the detailed biochemical reactions.

We use the cell state equations of the following form to model intracellular processes:

$$\text{(2.1)} \quad \textbf{if} \quad \mathsf{Condition_i}(state, env) \quad \textbf{then} \quad \frac{dstate[i]}{dt} \mathrel{+}= \mathsf{Consequent_i}(state, env)$$

Equation 2.1: The basic form of a cell state equation for a cell with state vector *state* and local environment characterized by the vector *env*. This formula determines a contribution to the state variable denoted *state*[*i*]. □

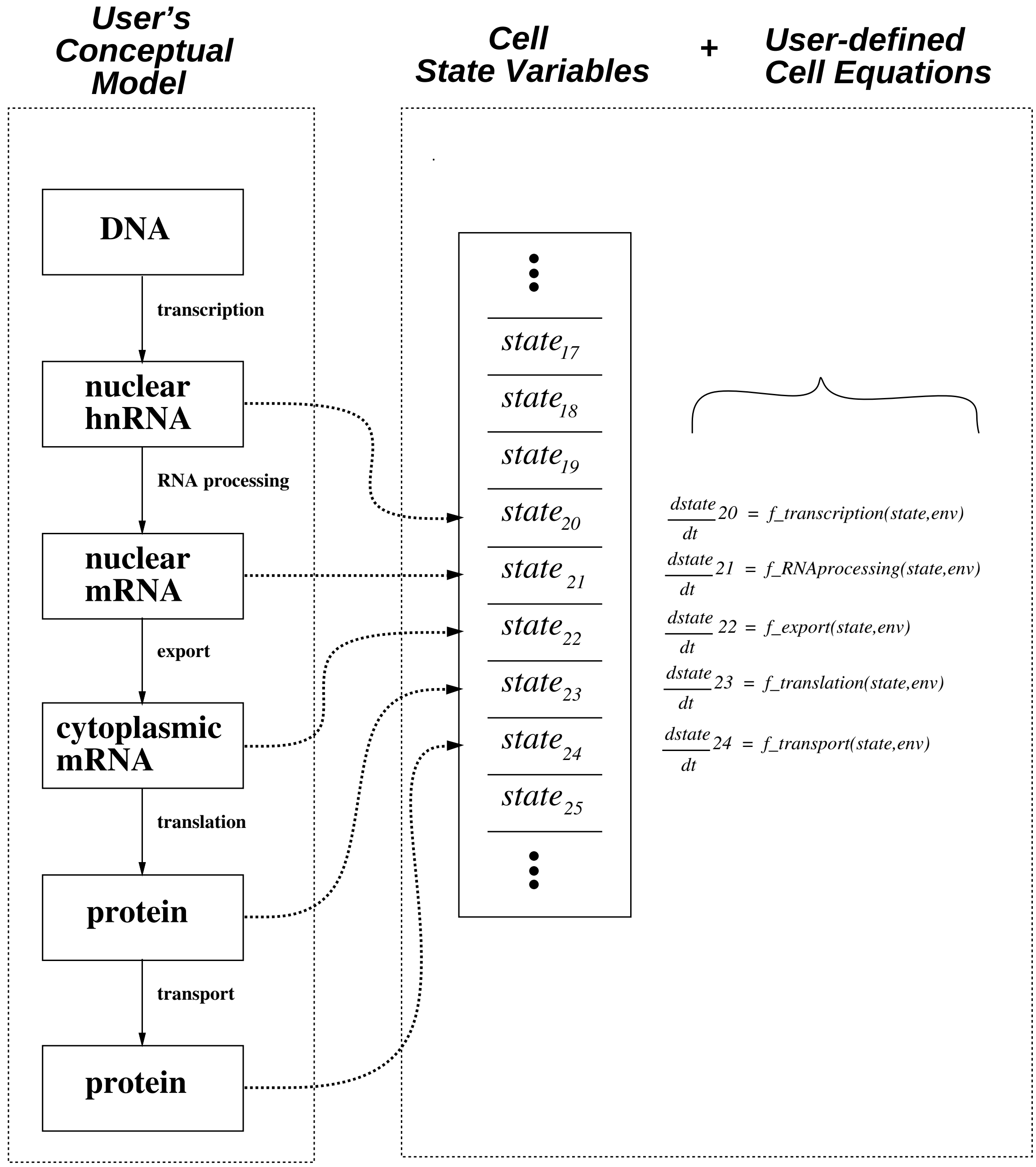


Figure 2.3: **Open Interpretation of State Variables.** This figure shows an example of using state variables to represent intracellular processes. On the left is a schematic description of the gene expression for some particular gene. On the right, we show its implementation as a set of state variables (one for each conceptual state in the schematic on the left). Processing stages such as mRNA translation are implemented as differential equations which modify state variables.
The user defines the particular differential equations to implement the processing stages (shown as *f_transcription*, *f_mRNAprocessing*, etc., in the figure.) The user also 'interprets' the state variables as having particular meanings. The simulator does not know that state variable $state_{22}$ is being used to represent the concentration of cytoplasmic mRNA. The simulator just knows that there is a variable which is modified by a given differential equation. The interpretation of the variables is left to the user. □

Gene regulation. For the moment, let us think of this equation as describing the activity of a single gene (in Section 2.4 we generalize). The concentration of the gene product is state[i]. The Condition contains regulatory terms for this gene and describes whether the gene is being expressed or not. It depends on the state of the cell, and may contain models for promoters, enhancers, chromatin, etc. Anything which can affect gene transcription will be modeled as a term in this expression. The Consequent then describes the rate (and perhaps also the setpoint) of gene expression.

The Condition can be used to turn on and off a single gene, or the same condition can be used to modulate a group of genes, giving homeobox-like behavior. [3]

Note that both the Condition and Consequent depend on the cell's state and its local environment. In real cells, the local environment generally doesn't directly affect gene transcription. Depending on the user's desires, the local environment terms may all be zero for the gene transcription, and may only be incorporated in the equations for other cell state variables which represent cellular messengers. More detail on the contents of the local environment array is in Section 2.6.

What is the +=? The += is simply a notational convention to indicate that the products of genes which code for the same protein combine additively, thus each += expression is a term in a sum over all expressions for state variable $state[i]$ in the cell. [4]. There may be multiple genes which code for the same product, but are expressed in different situations and at different rates. This notation is a convenient way to capture such phenomena.

Implications of += for artificial evolution. The += notation has some particularly nice properties for evolutionary algorithms, since this means that duplicating a gene simply doubles the rate of gene expression, and doesn't necessarily cause major disruptions in the developmental process (as opposed to some other gene representations, see Chapter 7). Then one of the copies can undergo severe mutation or recombination while the other copy continues to function as before, perhaps enabling the organism to survive, and hence permitting the evolutionary algorithm to explore more territory or tunnel through local minima.

Implications of += for constraints. Another feature of the += notation is that it captures the concept of superposition of energy constraints. Energy constraints are functions that go to zero when the constraint is met. They are also combined additively. Soft (penalty) or hard (Lagrange) constraints can be expressed in this manner. This has primarily been of use in the construction of computer graphics models, as described in Chapter 8. However, it leads one to wonder if there is a naturally occurring system of reactions that implements Lagrange multipliers, perhaps involved in homeostasis.

Continuous conditionals. The conditional part of the cell state equations need not be a binary function. In the real world, reactions do not generally turn on and off instantaneously, but instead ramp up to their full rate. In the computational world, discontinuities caused by binary functions lead to numerical problems, since solvers need to determine the location of the discontinuity and

[3]The piecewise ODEs can thus manage state changes, performing a similar function to the grammar rules in the *connectionist model* [Mjolsness et al., 1991].

[4]This += expression is adopted from the C programming language [Kernighan and Ritchie, 1978]

deal with it. For these reasons, we model continuous conditionals as sigmoids, which have a similar conceptual function as binary conditionals, but they ramp up with a finite slope (user-settable).

Thus we actually represent Eq. 2.1 as the product of this ContinuousCondition which lies in the interval $[0, 1]$ and the Consequent. When the condition is close to one, behavior is as if it were true.

$$\frac{dstate[i]}{dt} \mathrel{+}= (\mathsf{ContinuousCondition}_i(state, env))(\mathsf{Consequent}_i(state, env)) \tag{2.2}$$

Equation 2.2: Continuous condition form of the cell state equations. □

Conditional operators are implemented with continuous counterparts, which return values on the interval $[0, 1]$. We have done this for the functions $>$, $<$, AND and OR, and we call the resulting continuous versions $\tilde{>}$, $\tilde{<}$, $\widetilde{\mathsf{and}}$ and $\tilde{\mathsf{or}}$.

$$\begin{aligned} (a \tilde{>} b) &\equiv (tanh(steepness * (b - a)) + 1)/2 \\ (a \tilde{<} b) &\equiv (tanh(steepness * (a - b)) + 1)/2 \end{aligned} \tag{2.3}$$

Equation 2.3: Continuous Conditionals $<$ and $>$ defined over $a, b \in \Re$ □

$$\begin{aligned} (A \ \widetilde{\mathsf{and}} \ B) &\equiv min(A, B) \\ (A \ \tilde{\mathsf{or}} \ B) &\equiv max(A, B) \end{aligned} \tag{2.4}$$

Equation 2.4: Continuous AND and OR defined over $A, B \in [0, 1]$ □

Other choices are possible for these functions, such as $(A \ \widetilde{\mathsf{and}} \ B) \equiv AB$ and $(A \ \tilde{\mathsf{or}} \ B) \equiv max(A + B, 1.0)$.

Combining all of the equations for one state variable. Combining the continuous conditionals with the $+=$ notation gives us the following equation which combines all contributions S to the state variable $state[i]$:

$$\frac{dstate[i]}{dt} = \sum_S (\mathsf{ContinuousCondition}_i^S(state, env))(\mathsf{Consequent}_i^S(state, env)) \tag{2.5}$$

Equation 2.5: Summation of all terms S for $state[i]$ in a cell. □

Can state variables be negative? If we use a cell state variable to encode the concentration of a chemical in the cell, clearly it should be constrained to be greater than or equal to zero. However, if we apply a different interpretation to the variable, negative values may be appropriate. For example, a single signed variable can be used to represent the difference of two concentrations. This suggested a computational speedup by allowing our state variables to be non-negative where appropriate (at the user's option), reducing the number of variables as well as omitting the calculation of the non-negative constraint.

2.6 Local Environment Variables

The inputs to the cell state equations are provided in the local environment variables, which contain information which a cell can sense nearby in its external environment. These include:

- ◇ cell size
- ◇ concentration and gradient of diffusable chemicals at a local sensor (perhaps with noise)
- ◇ amount of cell surface chemicals in a bound state (for detecting contact with other cells – see Section 3.2.4)
- ◇ direction of fields such as light, gravity

Cell size is included as an environmental variable since the equations of motion and growth of cells are actually computed in a global process, and then propagated back to the cells. In this model, a cell does not have access to its absolute location, but it does know its size. All of the values are provided in nondimensionalized form.

In the simulations shown, the local environment variables were defined to contain the values and gradients of the chemicals around each cell. For each diffusable chemical in the environment ($i \in 0, \ldots, m$), there are three variables: one for the value ($chem_i$), and two for the components of the two-dimensional spatial gradient ($\partial chem_i/\partial x$, $\partial chem_i/\partial y$). This gradient is rotated into the cell's local coordinate frame (Section 3.2.2).

For each membrane protein ($j \in 0, \ldots, n$), there is a value (mem_j) which approximates the amount of that protein which is bound to a matching protein on an adjacent cell (in our model, the cells must be in contact for their membrane proteins to bind – see Section 3.2.4 below for details).

Thus the entire array of local environment variables has the form:

$$\begin{aligned}\mathsf{env[\ ]} \;=\; & (size,\ chem_0,\ \frac{\partial chem_0}{\partial x},\ \frac{\partial chem_0}{\partial y},\\ & chem_1,\ \frac{\partial chem_1}{\partial x},\ \frac{\partial chem_1}{\partial y}, \cdots, chem_{m-1},\ \frac{\partial chem_{m-1}}{\partial x},\ \frac{\partial chem_{m-1}}{\partial y},\\ & mem_0,\ mem_1,\ \cdots,\ mem_{n-1})\end{aligned}$$

Other values such as the direction of gravity or light are easily added to this framework. For the computer graphics application of Chapter 8, the implicit surfaces are implemented using this framework. The value and gradient of a function defining the implicit surface are simply inserted as local environment variables, as if they were another chemical.

Notation. In the actual experiment files (and in this thesis), the env[ ] array is accessed using mnemonic index names. For example, env[surf_a] refers to the environment variable which reports the amount of surface membrane protein a which is bound, and (env[rdx], env[rdy]) refers to the gradient of a diffusing chemical r.

User control of reaction-diffusion in the environment. The user can enter arbitrary reaction-diffusion equations to compute the interactions of extracellular chemicals. A *reaction* function is entered that describes the modification of a particular chemical based on the concentrations of other chemicals. This function is called at each node in the discretized grid. An example of the use of user-defined reaction function is shown in Section 5.2.6. Section 3.2.6 describes this in more detail; the reaction function is called $\mathsf{R}_a(\mathcal{A})$ in Eq. 3.3.

2.7 Cell Behavior Functions

The behavior functions compute a cell's behaviors based on its current state. Cells have both continuous and discontinuous behaviors. Continuous behaviors are simply continuous functions of the state variables. Discontinuous behaviors are events triggered when a behavior function crosses a specified threshold. These discontinuous events contribute to the event function $g()$ mentioned in Section 3.2.1.

continuous cell behaviors Cells exhibit several continuous behaviors, determined by the cell behavior functions:

- ◇ attempt to move in some direction (may be limited by collisions, adhesion, or drag)
- ◇ attempt to grow in size
- ◇ emit or absorb chemicals from the environment
- ◇ change concentrations of particular proteins in the membrane (e.g., cell adhesion proteins, which mediate how much this cell will adhere to another cell)

These are the behavior functions used to compute most of the two-dimensional simulations shown in Chapter 5. The three-dimensional versions are a straightforward extension. We use quaternions to represent orientation in three dimensions.

$$\begin{array}{lcl} \mathsf{MotiveForce}(\mathsf{state}, \mathsf{env}) & \equiv & (\mathsf{state}[\mathsf{fx}], \mathsf{state}[\mathsf{fy}]) \\ \mathsf{GrowthForce}(\mathsf{state}, \mathsf{env}) & \equiv & \mathsf{state}[\mathsf{fsize}] \\ \mathsf{CleavageAxis}(\mathsf{state}, \mathsf{env}) & \equiv & \mathsf{state}[\mathsf{fangle}] \\ \mathsf{EmitOrAbsorb}_a(\mathsf{state}, \mathsf{env}) & \equiv & \mathsf{state}[i_a] \end{array} \tag{2.6}$$

Equation 2.6: Continuous Cell Behaviors. □

$$\begin{array}{lcl} \mathbf{f}_{motive}(z, env) & \equiv & (z_{fx}, z_{fy}) \\ f_{angle}(z, env) & \equiv & z_{fangle} \\ f_{size}(z, env) & \equiv & z_{fsize} \\ f_{emit\_a}(z, env) & \equiv & z_{emit\_a} \end{array} \tag{2.7}$$

Equation 2.7: Continuous Cell Behaviors (short notation form). □

As mentioned above in Section 2.4, state variables fx, fy, fsize, etc., are pre-defined by the system, and are used to determine the behaviors via the cell behavior functions.

MotiveForce is a vector which is the locomotive force the cell is currently exerting. This is expressed in coordinate frame local to the cell (which keeps the cell from having access to the global coordinate frame, Section 3.2.2). When state[fx] is positive, the cell will exert a force in the positive x direction in its local coordinate frame.

For our current cell shape model (circle or sphere), the GrowthForce is a scalar representing the force the cell is exerting to change its radius. For other shape models, more parameters are needed, and GrowthForce would be a vector. CleavageAxis determines the rotation rate of the cleavage axis (axis of the next cell division).

For each chemical $a \in 0, 1, \ldots, nchems$, $\mathsf{EmitOrAbsorb}_a$ defines the rate at which a diffusable chemical a is being emitted into or absorbed from the environment by a cell. In the simulations shown, the rate at which chemical a is emitted by the cell is determined a single state variable $\mathsf{state}[i_a]$, where i_a is an index into the state array.

discontinuous cell behaviors (events) The cell provides functions which determine the timing of the following events. An event is a discontinuity in the solution, which stops the solver and may create or destroy data structures. The event occurs when the corresponding cell behavior function crosses zero.

- ◇ split (cell division)
- ◇ die
- ◇ emit neurite with growth cone

$$
\begin{aligned}
&\mathsf{TimeToSplit(state, env)} &\equiv\ & ((\mathsf{env[radius]} \mathrel{\tilde{>}} \mathrm{r}_0)\ \widetilde{\mathsf{and}} \\
& & & (\mathsf{state[split]} \mathrel{\tilde{>}} \mathit{split\_threshold})) \\
&\mathsf{TimeToDie(state, env)} &\equiv\ & (\mathsf{state[die]} \mathrel{\tilde{>}} \mathit{die\_threshold}) \\
&\mathsf{TimeToEmitNeurite(state, env)} &\equiv\ & (\mathsf{state[emitgc]} \mathrel{\tilde{>}} \mathit{emitgc\_threshold})
\end{aligned} \tag{2.8}
$$

Equation 2.8: Events: Discontinuous Cell Behaviors Occur at $\mathsf{TimeToX} = 0.5$. The PODE solver looks for zero-crossings, so this is actually computed using $\mathsf{TimeToX} - 0.5$. □

The definition of TimeToSplit() is a continuous version of the condition

$$((\mathsf{env[radius]} > r_0)\ \mathsf{and}\ (\mathsf{state[split]} > \mathit{split\_threshold})).$$

Thus a cell splits when it is large enough (bigger than r_0) and has accumulated enough of the state[split] protein (more than *split_threshold*).

As mentioned before, this is an example of the equations used in the simulations shown; alternative functions of the state and environment variables can easily be defined as necesary. For instance, to implement cell division which occurs during early development (when a large oocyte divides several times without changing size), the TimeToSplit function could be redefined to be independendent of cell size.

Specifying initial state of progeny cells (cell lineage specification). The extensive modeling capabilities shown by Lindenmayer-systems [Lindenmayer and Prusinkiewicz, 1989, Prusinkiewicz and Lindenmayer, 1990] indicate the value including cell lineage mechanisms in our model. We include a cell behavior function that describes the setting of state variables in the daughter cells upon cell division. This behavior is dependent on the current state of the cell, so it can divide into cells of various types dependent on its own state and its local environment. This mechanism can implement grammar rules such as those used in L-systems. [5]

User control of cell behavior functions. The cell behavior functions mentioned above are used for all of the experiments shown. It is straightforward to allow them to be modified by the user, a modification which is underway.

[5]The current implementation is somewhat restricted and only allows setting one state variable, but that is enough to specify cell type in many of our experiments. The generalization to allow setting all state variables is straightforward.

Direct specification of behaviors. The cell behavior functions shown above depend on state variables, e.g., $\mathbf{f}_{motive}$ depends on z_{fx}. The rate of change of z_{fx} is determined by some cell state equations. Thus the state equations determine the rate of change of the force rather than the force itself. Sometimes it is convenient to set $\mathbf{f}_{motive}$ directly to the output of the cell state equations (i.e., to dz_{fx}/dt). We provide this as another option to the user.

Why would we want to do this? One reason is to avoid creating a second-order system for the motion of the cells (however, see Section 10.4.3). Our viscous dynamics ($F = mv$) are first-order. But we specify motion by describing dz_{fx}/dt, so there is another derivative, and we have a second-order system. It really depends on what characteristics we wish to model; this option provides more flexibility.

Extensions to three dimensions or to other shapes. In three dimensions, we use a quaternion to represent the orientation of the next cell cleavage plane, and then the CleavageAxis becomes a four-dimensional object. The motive force is a three-dimensional vector. If we use a different shape representation that requires more parameters (e.g., an ellipse or a spline curve), then the GrowthForce needs to be changed accordingly.

2.8 Neurites and Growth Cones

We have implemented a simple abstraction of neural growth using growth cones which are modeled as small cells with a few restrictions. The growth cones are connected to the parent cell by a neurite. We model them as having most of the capabilities of cells [Purves and Lichtman, 1985, page 103], except that they die if the parent cell dies, and they cannot emit growth cones of their own. Branching neurites can be implemented as splitting of growth cones, analogous to cell division (but we have not yet implemented branching). Growth cones and cells communicate via a set of state variables which are held in the neurite. The growth cones and cells can modify and sense the levels of these state variables in the neurite.

Although the growth cone computes collisions with the cells, collisions between neurites and other objects are not computed. The geometry of the neurites is simply a record of the path of the growth cone. In biological systems, adhesion and mechanical interactions between neurites are known to be factors in neural development. However, implementing the neurite-cell collisions in two dimensions seems too restrictive on the cells' movements. Therefore, we have opted for computational expedience and do not compute collisions between neurites and other objects.

Activity within neurites is modeled using additional 'neurite state variables'. A cell and growth cone can communicate during development via the neurite by modifying and examining these state variables.

The electrical model – neuron firing Using Lloyd Watt's "Spike" simulator [Watts, 1993], we can simulate spiking neurons. Connections have an associated signal delay relative to the length of the neurite (see details in Chapter 4).

2.9 Other Generalizations, in Particular Hierarchy

The underlying mathematical and computational model for the simulator is that there is a set of objects interacting in a geometric space. The behaviors of the objects are determined by piecewise continuous differential equations.

This general computational model can be used for a variety of applications:

- model subcellular assemblies, organelles (by adding a hierachy of object containing other objects),
- economics: customers and providers,
- predator-prey,
- locations of buildings, roads, towns, post offices based on local rules about topography and population density.

The computational objects currently used to represent cells could be used to represent various organelles, and they could interact inside a cell in much the same way our current simulation allows cells to interact in their external environment. This would require creating a hierarchical scheme in which the subcellular components interacted with each other and with the containing cell, but not with entities outside the cell. This is similar to the relationship between the cells and the global environment in the current scenario.

Chapter 3

Implementation

This chapter contains more details about the equations used in the model, the numerical solution of those equations, and the software architecture.

- ◇ Software Architecture
- ◇ Equations
  - ◇ PODE framework (Piecewise-continuous Ordinary Differential Equation),
  - ◇ Coordinate Systems
  - ◇ Equations of Motion
  - ◇ Collision and Adhesion
  - ◇ Conversion between Concentration and Amount
  - ◇ Reaction-Diffusion Related Equations (of Extracellular Chemicals)
- ◇ Numerical Computation
  - ◇ Random Numbers
  - ◇ Stability of the Diffusion Computation

The material in this chapter is not essential for understanding the simulator operation at a high level, but will be useful to people who wish to implement a similar system. There are some discussions of modeling choices that may be of interest to the general reader:

- Alternatives for computing reaction-diffusion in the presence of moving sources. (Section 3.2.6)

- Uses of random numbers. (Section 3.3.2)

3.1 Software Architecture: Object-Oriented Design

Object-oriented programming is a method of decomposing problems into component modules (objects) and interfaces between the objects (protocols) [Booch, 1991]. We adopt this approach with our model to make it flexible and extensible. The program is written entirely in `C++`. Other alternatives such as LISP or Simula should be considered for future implementations. They are generally less efficient, but provide greater flexibility.

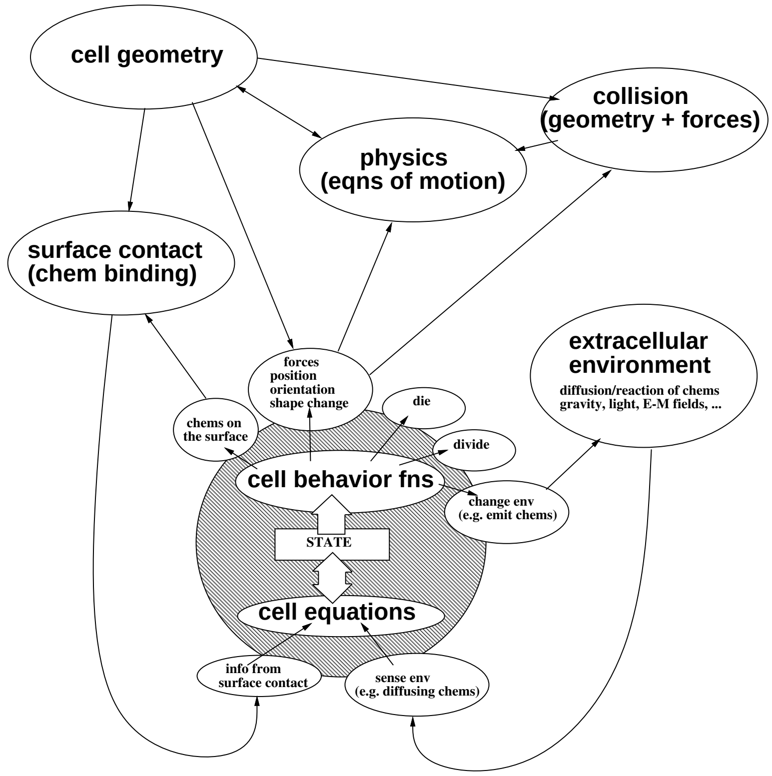


Figure 3.1: **Object Framework:** The conceptual objects in the model. We define an abstract interface between each of these objects to facilitate the maintenance of the simulation program as well as to enable us to modify one object without making extensive changes to the others. This sort of object design is standard practice in Object-Oriented Programming. □

Code modifications. An advantage of object-oriented design is that it makes some code modifications very easy. For example, adding a new feature such as gravity to the environment is straightforward because it follows the established protocols.

However, as with any object-oriented design, this only works if the desired modifications fit into the framework. If we later wanted to modify the model to do something else, it requires more work.

For example, the current cells cannot sense the amount of force acting on them. Adding this would require adding a new protocol (and we would draw a corresponding arrow from 'physics' to an input on the cell equations in Figure 3.1). This is slightly more work than adding something that fits the existing protocols.

A more difficult change is something that changes the choices of objects. For instance, if we were to compute the diffusion of chemicals within the cell boundary, and the diffusion or transport through channels in the cell membrane to the extracellular environment, then we would have to couple the environment and cell computations in a different way. This would require rethinking the existing object framework.

3.2 Detailed Equations of the Model

3.2.1 Combining the Equations into a PODE Solver Framework

The equations of the model are combined into a system of piecewise continuous differential equations (PODEs). This procedure is described in [Barzel, 1992, Appendix C]. We outline it here briefly. The differential equations and state variables arising from various sources are gathered into a system of ODEs:

$$\mathbf{y}'(t) = f(\mathbf{y}(t), t) \tag{3.1}$$

Equation 3.1: Differential equation framework. □

where $y(t)$ is the time-varying state of the combined system. The function f is *piecewise continuous*, that is, "the function f is a continuous and has bounded derivative except at isolated values of t." [Barzel, 1992, Appendix C.] Two auxiliary functions are constructed from the equations in Section 3.2 to describe the location in t of events, and the new initial condition from which to continue. The location in t of the discontinuities is given by an *event* function $g(\mathbf{y}, t)$ which is nonnegative with $\mathbf{y}(t)$ in a legal state. When g crosses zero, a discontinuity occurs. At a discontinuity, the *transition* function $\mathbf{h}(\mathbf{y}, t)$ describes the initial state for the next continuous segment.

At the discontinuity, structure can be added or deleted, corresponding to changing the number of variables in the system (and the dimensionality of $\mathbf{y}$). This feature is essential for modeling processes such as cell division or cell death.

3.2.2 Beware of Coordinate System Dependencies

What coordinate system is appropriate for the cell's MotiveForce (Section 2.7)? And for the chemical gradient information (Section 2.6)? Although it is convenient to have everything in the same (global) coordinates, there are some reasons to prefer having a local coordinate system associated with each cell.

When using the model for artificial evolution, we want to be very careful to make sure that we can't end up with expressions which can see the global coordinate system. Bunches of cells which sweep back and forth along the x axis are undesirable. One way to avoid this is to have all of the cell's geometrical information expressed in a local coordinate frame. This is the current implementation. For convenience, we choose the coordinate frame of the next cell cleavage orientation.

When using the model for developmental explorations, the use of a local coordinate frame constrains the user to create cell state equations that do not depend on the global coordinates. A real cell has no concept of a global coordinate system, and neither should the simulated cells. This helps the user to 'think like a cell,' and prevents the user from introducing certain non-physical behaviors.

For example, cells can move in a particular direction because either:

- the direction is related in some way to a direction sensed locally (e.g., via a chemical gradient, a velocity with respect to the surface, or some other means), or

- the cell is performing some kind of dead-reckoning and keeps track of its own heading internally.

To implement local coordinates for cells, we transform all sensor information from the local environment (Section 2.6) into local coordinates before handing it to the cell state equations. As the state equations are solved forwards in time, the vector quantities, such as the motive force, are transformed back to global coordinates for computing the equations of motion of the cell. The equations of motion also incorporate forces from other sources, such as collisions between cells and other objects.

The transformation between local and global coordinates is specified by an orientation stored in the cell's state array. In two dimensions, this is a single value representing an angle from the horizontal. In three dimensions, we have two implementations, one with a quaternion (four values) and one with Euler angles (two values).

In a future implementation, we would use tensor notation. This is described briefly in Section 10.4.1.

3.2.3 Equations of Motion for the Cells

The motion and growth of the cells is determined by the forces they generate, and the forces applied from collisions and other extracellular effects. In the low Reynolds number domain of small objects in viscous fluids, we determine the velocity v from balancing the drag force $F = k_{drag} v$ with the applied forces. The drag in a fluid of viscosity η for the disc and spherical cell shapes is [Berg, 1983, Page 57]:

$$
\begin{aligned}
\text{disc moving edge on}: \quad k_{drag} &= \tfrac{32}{3}\eta\, r \\
\text{sphere}: \quad k_{drag} &= 6\pi\eta\, r
\end{aligned}
\tag{3.2}
$$

Equation 3.2: Frictional drag coefficients for the disc and sphere cell models of radius r. □

The CollisionAdhesionForce is computed as the sum of all collision and adhesion forces acting on a cell (from its neighbors), as described below (Section 3.2.4).

$$
\begin{aligned}
\sum_{\mathsf{forces}} &= \mathsf{CollisionAdhesionForce}_c + \mathsf{MotiveForce}_c - k_{drag} v \\
&= 0
\end{aligned}
$$

Unlike the drag force, the collision forces do not depend on velocity (Section 3.2.4). Adding a dependence on velocity is straightforward, and leads to a set of simultaneous equations of motion which can be solved to find cell velocities.

3.2.4 Model of Contact Recognition and Cell Adhesion

Real cells have many proteins in their membranes which perform a variety of functions. We propose a simple model of membrane proteins (which we refer to as surface factors). Our model captures a few of the major functions:

- ◇ cell recognition (cells recognize that they are in contact by activation of membrane-bound molecules)
- ◇ cell adhesion (cells physically bind together via membrane-bound molecules)

In both cases, our model allows homophilic (like binds to like) and heterophilic (a complementary pair binds together) surface factors.

In real cells, surface factors may also bind via an extracellular linker chemical [Alberts et al., 1989, page 825]. We do not currently support this capability, although it would be a straightforward addition to the current system since the cells have the ability to sense and emit diffusing chemicals.

A single state variable directly controls the concentration of each surface chemical in a cell's membrane. The concentration of surface chemical which is bound to its complement on an adjacent cell is reported in the mem_j variables in the local environment array (Section 2.6). This suffices for cell contact recognition; cells can determine that they are in contact with other cells that express a certain surface factor. For cell adhesion, we compute a force depending on the amounts of all bound adhesive surface factors in the contacting regions of adjacent cells. This computation is described in the next section (Section 3.2.4).

Note that this model does not allow for asymmetric expression or recognition along the membrane. Cells cannot express a surface factor on just one side, nor can they tell which side has been contacted. This limitation may be important, and we are considering various models to enable specifying a spatial distribution of surface factors without adding too much computational burden. See Section 9.3.2 for further discussion.

All of the functions of the membrane protein model depend on the amount of a surface factor which is bound. This is computed from the contact area between two cells, and the concentrations of the complementary factors in their membranes. Let $\mathsf{Conc}_c(a)$ return the concentration of a on cell c. $\mathsf{ContactArea}(c,b)$ estimates the contact area between two cells. For a particular surface factor a which binds to a', we have:

$$
\begin{aligned}
\mathsf{AmountBound}_c(a) &= \sum_{b \in \text{cells}} \mathsf{Binding}(\mathsf{ContactArea}(c,b), \mathsf{Conc}_c(a), \mathsf{Conc}_b(a')) \\
&= \sum_{b \in \text{cells}} \mathsf{ContactArea}(c,b) * min(\mathsf{Conc}_c(a), \mathsf{Conc}_b(a'))
\end{aligned}
$$

The function Binding computes the fraction of chemicals which actually get bound. We currently use a very simple function, but more accurate models could be substituted here. For cell shapes defined as 2d circles, this is the chord length of the overlap between the circles (see next section). The recognition factors report AmountBound in the appropriate local environment variable (the mem_i mentioned in Section 2.6).

The adhesion between two cells b and c is computed as follows:

$$
\begin{aligned}
\mathsf{AdhesionForce}_{bc} = \sum_{a \in \text{surfchems}} \sum_{a' \in \text{surfchems}} & (\mathsf{ContactArea}(c,b) \times \mathsf{Conc}_c(a)) \times \\
& (\mathsf{ContactArea}(c,b) \times \mathsf{Conc}_b(a')) \times p_{bind} \times \mathsf{adhesion\_per\_bond},
\end{aligned}
$$

Biological cells also use membrane proteins as sensors to detect diffusable chemicals in the environment, and as channels for emitting diffusable chemicals. We could model these effects similarly, but instead we have chosen to allow cells to directly sense or emit the diffusable chemicals in the local environment (as discussed above in Sections 2.6 and 3.2.6). This is done for efficiency, but it does remove one level of indirection which might be useful as another regulatory switch for cell function. This is a feature which could easily be added to the existing implementation.

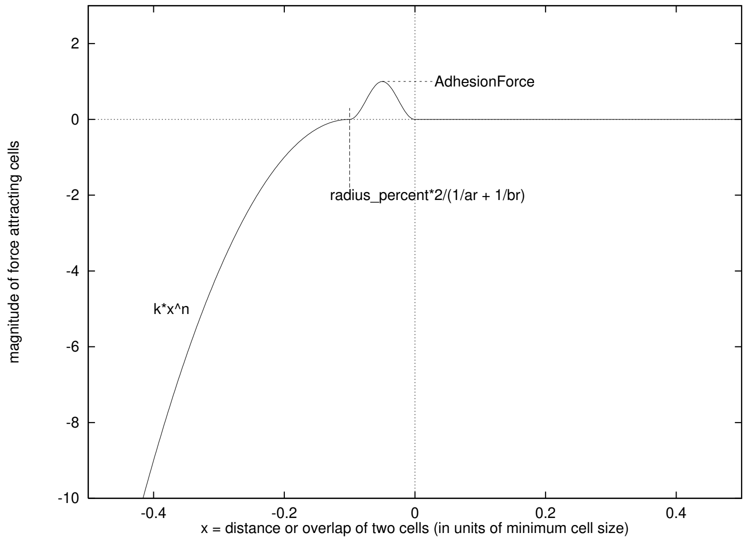


Figure 3.2: Collision-adhesion forces. 'ar' and 'br' are the radii of the two contacting cells. 'radius_percent' specifies the overlap for which the force is zero between two adhering cells as a function of the radius of the cells. In this figure, ar=1, br=1, radius_percent=0.1, AdhesionForce=1, k=1, and n=2. □

Forces due to collision and adhesion

The shape model in the current implementation is simply a circle (2d) or sphere (3d). Since we are modeling deformable objects (cells) we use a polynomial penalty method for generating collision forces. [Platt, 1989] This allows substantial overlap between adjacent cells, which gives us a crude approximation of cell deformation. The overlap can then be used to determine the area of contact between two cells. This method is similar to cell models which use moving points for cells and then compute the displayed shapes from a Voronoi algorithm [Honda, 1978, Honda, 1983].

When the distance between two objects is greater than zero, there is no contact. For overlapping objects, we define use a negative number to represent the maximum overlap of the two objects. If this value is less than zero, the surface chemicals begin to interact, and forces due to collision and/or adhesion begin to be exerted (Figure 3.2).

Note that using this paradigm, together with a desire that the collision-adhesion force function be 1-to-1 and C^2 continuous, implies that the adhesion forces will actually pull cells together slightly when they touch. This effect is not entirely non-physical, since adhering cells tend to enlarge their contacts [Gilbert, 1991]. Also, it eases the implemention considerably to avoid dealing with hysteresis for adhesion forces. Thus 'contact' is achieved and removed at overlap = 0 for all three effects (contact recognition, collision, and adhesion).

The adhesion 'bump' in the force function (Figure 3.2) asymptotes to zero before entering the polynomial collision region. This was done to make the numerical solution more stable, since cells which are sticking together will tend to be operating in that region of the force function.

The interpenetration of cells at the adhesion-collision force equilibrium can be ameliorated for display by introducing a small boundary layer around each cell (drawing them as smaller circles), or by displaying cells using a modification of the Voronoi algorithm [Honda, 1978, Honda, 1983].

Collisions are computed using a penalty term [Platt, 1989], which is introduced into the equations of motion for each cell as a force. For every colliding pair of objects, the collision manager computes

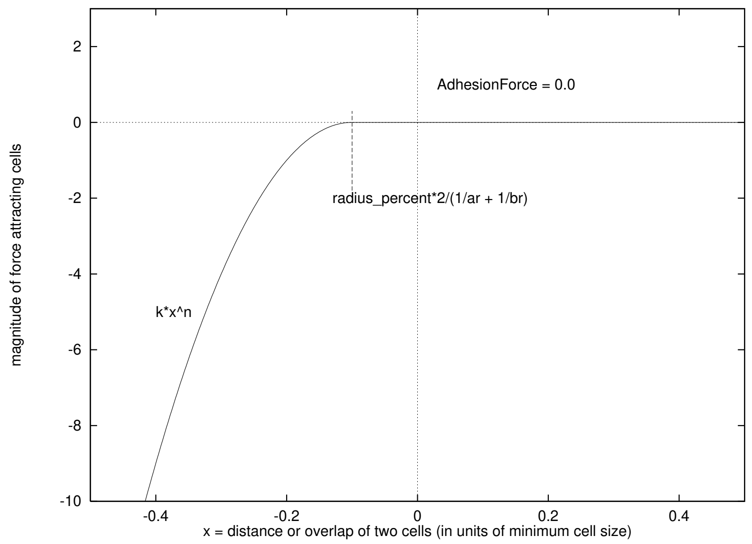


Figure 3.3: Collision forces in the absence of adhesion. In this figure, ar=1, br=1, radius_percent=0.1, AdhesionForce=0, k=1, and n=2. □

equal and opposite forces. Note that the objects for the collision computation are not restricted to being cells. For instance, collisions are also computed with the boundaries of the environment, and other objects can be introduced as well (to model bone, fibers or other substances in the environment).

For every pair of objects b, c, we denote the function computing their maximum overlap as $\mathsf{d}(b, c)$. $\mathbf{p}$ is a unit vector in the direction of the maximum overlap.

Equation for collision between cells a and b given the parameters radiuspercent, k, and n as defined in Figures 3.2 and 3.3.

$$
\begin{aligned}
\mathsf{bump}(\mathsf{x}, \mathsf{width}, \mathsf{height}) &\equiv 0.5\, height \left(1 - \cos(\frac{2\,\pi\,x}{width})\right) \\
\mathsf{repulse}(\mathsf{x}, \mathsf{k}, \mathsf{n}) &\equiv (k\,x)^n \\
\mathsf{CollisionAdhesionForce}(\mathsf{x}, \mathsf{AdhesionForce}, & \\
\mathsf{ar}, \mathsf{br}, \mathsf{radiuspercent}, \mathsf{k}, \mathsf{n}) &\equiv \mathrm{If}(x > 0., 0., \\
&\quad \mathrm{If}(x < -pr, \\
&\quad\quad -repulse(-\,(x + radiuspercent)\,, k, n), \\
&\quad\quad bump(x + pr, pr, \mathsf{AdhesionForce})))
\end{aligned}
$$

Note: in the 'If' construct used here, the first argument is the condition. If the condition is true, the second argument is computed, else the third is computed.

Combined Forces on a Cell

The combination of all collision and adhesion forces acting on a cell can cause a net motion. We also compute forces to compress/expand the cell, although they are generally not used (see discussion in Section 9.3.1). We compute the force components using the following method.

```
Compute compression/expansion and net forces on a cell.

1. Keep two lists of 'inwards' and 'outwards' forces.
 (inwards due to collision, outwards due to adhesion)
2. Compute total force. (sumf = sum f_inwards + sum f_outwards)
3. Separate components of each inwards force into
   component along sumf (f_parallel) and orthogonal
   to sumf (f_orthog.
   (if sumf close to 0, any vector will do)
4. Orthogonal compression = sum |f_orthog|/2
5. Parallel   compression = (|sumf| - sum |f_parall|) / 2
6. Set compression_force =  ortho + parallel
7. Repeat steps 3-6 for outwards (expansive) forces.
8. Results:
     Total compression/expansion force =
           expansion_force - compression_force
     Total moving force = sum f\_inwards + sum f\_outwards
     Note: positive is outwards, (expansive)
```

3.2.5 Conversion between Concentration and Amount

If we embrace the abstraction that the state variables represent concentrations of intracellular proteins, and that the cell state equations are analogous to genes, then we need to convert between the state variables as concentrations and the output of the genes which are amounts of new proteins (per time). For chemicals which are diffusing freely through the cytoplasm (such as ions), this conversion will be affected by changes in the volume of the cell.

For other interpretations of state variables, such as a concentration within an enclosed region of the cell (e.g., endoplasmic reticulum), a change in the volume of the cell is unimportant, since the volume within which the chemical exists can remain constant.

Thus the code to scale by the volume may or may not be required, depending upon the user's interpretation for the state variables.

In practice, the code discussed in this section is not used in many of the later simulations shown in this thesis, primarily because it makes writing cell state equations more difficult (since they now depend rather sensitively on changes to cell size).

It is interesting to consider whether this is a difficulty for real cells, and even what it means for something to be 'difficult' for a real cell. One definition that seems reasonable is that process A is more difficult than process B if it requires a finer tuning of a parameter, or if it depends on more parameters.

Real cells are sensitive to volume changes caused by a change in the salinity of their environment. Even a slight swelling or shrinking caused by changes in extracellular ionic concentrations will cause cells to die.

Derivation of amount-concentration conversion. Recall that the *state* arrays in the cells contain putative concentrations of chemicals. *dstate* is computed by adding or subtracting an incremental *amount* of some chemical to the cell. The cells are modeled as thin disks lying on a plane, with radius r and height k.

Term	Definition
a_i	amount of substance in cell
z_i	concentration of substance in cell
r	radius of cell in 2d
k	height of cell disk
v	volume of cell as a disc in 3d

Given the change in amount $\frac{da_i}{dt}$, we compute the change in concentration (the state variables).

$$\begin{aligned} z_i &\equiv \frac{a_i}{v} \\ &= \frac{1}{v}\frac{da_i}{dt} + \frac{a_i}{v^2}\frac{dv}{dt} \\ &= \frac{1}{v}\frac{da_i}{dt} + \frac{c_i}{v}\frac{dv}{dt} \\ &= \frac{1}{v}(\frac{da_i}{dt} + c_i\frac{dv}{dt}) \end{aligned}$$

Since the radius of the cell may also be changing as a function of time, we compute the derivative of volume:

$$\begin{aligned} v &= \pi k r^2 \\ \frac{dv}{dt} &= 2\pi k r\frac{dr}{dt} \end{aligned}$$

Substituting, we find:

$$\begin{aligned} \frac{dz_i}{dt} &= \frac{1}{v}(\frac{da_i}{dt} + c_i 2\pi k r\frac{dr}{dt}) \\ &= \frac{1}{\pi k r^2}(\frac{da_i}{dt} + 2k\pi c_i r\frac{dr}{dt}) \end{aligned}$$

3.2.6 Diffusion and Interaction of Chemicals in the Environment

The diffusion of each chemical is governed by a partial differential equation for f_a, the concentration of chemical a. The $\mathsf{R}_a(\mathcal{A})$ function computes reactions occurring naturally between chemical a and all other chemicals $\mathcal{A}$ as they mix in the extracellular matrix. An example of a user-defined $\mathsf{R}()$ is shown in Section 5.2.6. For each chemical a, at a particular location:

$$\frac{\partial f_a(x,y)}{dt} = -\bigtriangledown^2 f_a(x,y) - \text{dissipation}_a + \mathsf{R}_a(\mathcal{A}) + \mathsf{SourcesAndSinks}_a(x,y,t) \tag{3.3}$$

Equation 3.3: Diffusion and reaction of extracellular chemicals. □

Each cell can emit or absorb chemicals locally, and thus contributes to $\mathsf{SourcesAndSinks}(x,y,t)$. We have experimented with two models for the location of the sources/sinks on the cells:

- ◇ several locations along the cell perimeter
- ◇ a single location at the center of the cell

We primarily use the single location since it gives qualitatively similar results and is computationally more efficient. For this case, the function for chemical a can be specified as:

$$\mathsf{SourcesAndSinks}_a(x, y, t) = \sum_{c \in \text{cells}} \mathcal{P}(x, y, c_x, c_y)\ \mathsf{EmitOrAbsorb}_{c,a}(state_c, env_c)$$

where the location of cell c is (c_x, c_y), and $\mathcal{P}()$ describes the locations at which the chemical is released on the cell.

Approaches to solving a reaction-diffusion equation with moving sources.

There are many ways one might approach this problem. Our current implementation method is to spatially discretize the equations. We considered the following options, which are elaborated below:

- ◇ Discretization
- ◇ Closed Form
- ◇ Bumps
- ◇ Hybrid (Closed Form and Discretization)
- ◇ Particles

Discretize. We discretize the reaction-diffusion equations in spatial coordinates to create a set of ODEs. These ODEs are then combined in our general PODE framework with the rest of the differential equations arising from other computations (Section 3.2.1).

This approach has the advantage of scaling well with the number of cells at a given scale. It also works for an arbitrary chemical reaction function R. However, the grid size limits the solver stepsize (Section 3.3.1), and smaller cells require smaller grid sizes. For some experiments where accuracy in the diffusion values are important, the use of a fine grid causes the diffusion to dominate the computation time. In many biological situations, we suspect that this accuracy is unnecessary since the diffusing chemicals are primarily used for broad effects (e.g., with hormones).

The discretization approach for solving diffusion equations is quite popular in biological modeling, despite its computational expense and the large storage required. Kaandorp has used a hierarchical spatial discretization to simulate the growth of sponges in two and three dimensions[Kaandorp, 1994]. His approach reduces the storage requirements substantially, but the diffusion computation is still a major computational burden.

Closed form. Another approach we have considered is a closed-form solution which is an integral over time. With assumptions about how far back in time one needs to look, and how far out spatially, this approach might yield a reasonable algorithm. However, it may not scale well with the number of cells, since each will have to compute the integral independently and each integral depends on the moving positions of all the other cells (if they are emitting chemicals). Also, using this closed form solution would put more restrictions on the extracellular reaction function R().

Bumps. Similar to the closed-form approach, we have considered representing the diffusion of chemicals from a cell as a simple bump function centered at the cell. This representation would not compute any time-varying diffusion field, and is incapable of representing the reaction function R(). However, it scales well (if the bump function has local support), is fast to compute, and does enable cells to find each other by following gradients.

Hybrid. Using a low resolution discretized grid in conjunction with the closed-form or bump approach might yield the benefits of both. The closed-form part can be used for accuracy in the largest terms, but some locality in time and space can be achieved by computing the smaller terms in the discretization. This one might be a real pain to implement, though.

Particles. Diffusion can also be simulated by representing things at finer scale, and computing an approximation to Brownian motion for many particles. Although it is clearly prohibitive at this time to simulate the numbers of molecules present in real cells, perhaps a small representative sample would have the appropriate behavior. This approach would scale well with number of cells, but poorly with amounts of diffusing chemicals in the environment.

Each of these methods has different scaling properties, and they also impact what sort of solver can be used. They also vary in the how difficult they would be to combine with the other equations in our system. Considering all these factors, we have opted for the simple and general approach of discretization, despite the possible performance penalty.

The discretization of the reaction-diffusion equation. The function $f_a(x, y, t)$ is discretized on an $n \times n$ grid in two dimensions to give n^2 ODES. The notation f^{ij} indicates the value of the discretized variable at node (i, j) in the 2d grid.

$$\begin{aligned} \frac{df_a^{ij}}{dt} &= -(f_a^{i+1,j} + f_a^{i-1,j} + f_a^{i,j+1} + f_a^{i,j-1}) - 4f_a^{ij} \\ &\quad -\text{dissipation}_a + \mathsf{R}_a(f_a, f_b, f_c, \ldots) + \mathsf{SourcesAndSinks}_a^{ij}(t) \end{aligned}$$

The discrete version of the SourcesAndSinks() function is computed by partitioning the components of a cell's emission/absorption between the adjacent grid points using bilinear extrapolation.

Discrete cells and continuous diffusion. There are several nontrivial issues involving the interactions between discrete cells and diffusing chemicals in their environment. These issues must be addressed when designing a system such as ours:

- ◇ sensors (type, location),
- ◇ spewers (how cells emit or absorb diffusing chemicals), and
- ◇ presence of noise in the environment or the sensors.

Sensing values in the diffusion grid. In addition to modifying the information in the diffusion grid (via the SourcesAndSinks() function), a cell can sense the values and gradient of a chemical locally via sensors. We have implemented several sensor strategies to date:

- ◇ multiple sensors on the cell's periphery (sensing value only, gradient is computed by differences),
- ◇ a single sensor at the cell's center (sensing value and gradient directly), and
- ◇ a single sensor at a random location on the cell (sensing value and gradient).

We have found both the randomly moving sensor and the center sensor to be the effective. The randomly moving sensor computes an approximation to the value at the center of the cell. It is more efficient than using multiple sensors, and it avoids some problems which occur when using a stationary center sensor and source which are both located at the center of the cell. Note, however, that using pseudo-random numbers in this manner affects the computation and numerics, as discussed in Section 3.3.2.

An alternative strategy for implementing the sensors is to represent the amount of sensor proteins in the cell membrane. The strength of the sensor signal then depends on this amount, similar to the contact recognition computation discussed below in Section 3.2.4.

3.2.7 Non-Dimensionalizing the Diffusion-Related Equations

The following equation represents reaction-diffusion on a continuous substrate, with moving point sources/sinks:

$$m^*_{,t}(x^*, t^*) = \alpha_k m^*_{,ii}(x^*, t^*) + c^*(x^*, t^*)$$

Where:

$$\begin{aligned} c^*(x^*, t^*) &= \left\{\begin{matrix}\text{point}\\ \text{source}\end{matrix}\right\} + \left\{\text{dissipation}\right\} + \left\{\text{reaction}\right\} + \left\{\text{noise}\right\} \\ &= s^*(t)\delta^*(x^* - x^*_c(t^*)) - (1/H_k)m^*(x^*, t^*) + f^*(.) + n^*(x^*, t^*) \end{aligned}$$

Units for functions, independent variables, parameters are listed below. The symbol # is number of molecules, measured in moles. L represents unit length, T is unit time.

$$\begin{array}{rcll} t^* & \rightarrow & L & \text{time} \\ x^* & \rightarrow & L & \text{position} \\ x^*_c(t^*) & \rightarrow & L & \text{position of cell} \\ m^*(x^*, t^*) & \rightarrow & \#/L^2 & \text{concentration} \\ f^*(x^*, t^*) & \rightarrow & \#/L^2T & \text{reaction with other chemicals} \\ s^*(t^*) & \rightarrow & \#/L^2T & \text{creation/destruction at cell} \\ n^*(x^*, t^*) & \rightarrow & \#/L^2T & \text{noise} \\ \theta & \rightarrow & \#/L^2 & \text{“noise” concentration threshold} \\ \alpha & \rightarrow & L^2/T & \text{diffusion rate (canonical)} \\ \alpha_k & \rightarrow & L^2/T & \text{diffusion rate for chemical } k \\ H & \rightarrow & T & \text{dissipation “half life” (canonical)} \\ H_k & \rightarrow & T & \text{dissipation “half life” for chemical } k \\ r & \rightarrow & L & \text{min cell size} \end{array}$$

Find nondimensional parameter combinations following procedure in Lin and Segel [Lin and Segel, 1974]:

$$\begin{aligned} \alpha^p\ \theta^q\ H^s\ r^u &= 1 \\ \left(\frac{L^2}{T}\right)^p \left(\frac{\#}{L^2}\right)^q T^s L^u &= 1 \end{aligned}$$

We choose a free parameter $s = 1$ to get our nondimensional parameter:

$$\eta = \left(\frac{\alpha H}{r^2}\right)^s = \frac{\alpha H}{r^2}$$

Nondimensionalized variables and functions:

$$
\begin{aligned}
x &= x^*/r \\
t &= t^*/H \\
m(x,t) &= \frac{1}{\theta} m^*(rx, Ht) \\
c(x,t) &= \frac{H}{\theta} c^*(rx, Ht) \\
s(t) &= \frac{H}{\theta} s^*(Ht) \\
x_c(t) &= \frac{1}{r} x_c^*(Ht) \\
n(x,t) &= \frac{H}{\theta} n^*(rx, Ht)
\end{aligned}
$$

We take derivatives to get our entire equation in nondimensional form:

$$
\begin{aligned}
\frac{\partial}{\partial t^*} m^*(\frac{x^*}{r}, \frac{t^*}{H}) &= \theta \frac{\partial}{\partial t^*} m(\frac{x^*}{r}, \frac{t^*}{H}) = \frac{\theta}{H} \frac{\partial}{\partial t} m(x,t) \\
\frac{\partial^2}{\partial x^{*2}} m^*(\frac{x^*}{r}, \frac{t^*}{H}) &= \theta \frac{\partial^2}{\partial x^{*2}} m(\frac{x^*}{r}, \frac{t^*}{H}) = \frac{\theta}{r^2} \frac{\partial^2}{\partial x^2} m(x,t)
\end{aligned}
$$

Substituting these into the equation, we get:

$$
\begin{aligned}
\frac{\theta}{H} \frac{\partial}{\partial t} m(x,t) &= \alpha_k \frac{\theta}{r^2} \frac{\partial^2}{\partial x^2} m(x,t) + \frac{\theta}{H} c(x,t) \\
\frac{\partial}{\partial t} m(x,t) &= \left(\frac{\alpha_k H}{r^2} \right) \frac{\partial^2}{\partial x^2} m(x,t) + c(x,t) \\
m_{,t}(x,t) &= \left(\frac{\alpha_k}{\alpha} \right) \eta m_{,ii}(x,t) + c(x,t) \\
m_{,t}(x,t) &= \eta_k m_{,ii}(x,t) + c(x,t)
\end{aligned}
$$

where

$$
\eta_k = \left(\frac{\alpha_k}{\alpha} \right) \eta = \frac{\alpha_k H}{r^2}
$$

.

3.3 Numerical Computation

We have combined the equations arising from chemical, mechanical, and electrical sources into one large system of ordinary differential equations. Since discontinuous events occur in the ODE system (representing cell division, collisions between cells, etc.), the numerical implementation is based on a piecewise-continuous ordinary differential equation (PODE) solver [Barzel, 1992]. The PODE solver allows the addition/deletion of variables at discontinuities, which occur when cells split or die during the course of a simulation run.

There is a tradeoff regarding the choice of numerical method for solving the simulation equations and the solution's computation time, stability, and accuracy. The solver needs to work well in the context of the three dominating effects encoded in the differential equations: diffusion of chemicals on a grid, forces which cause the cells to migrate, and forces induced by cell-to-cell contacts.

We have chosen one of the simplest numerical solvers that produces stable and qualitatively accurate solutions. After experimenting with several ODE solution techniques (variable-order variable-step Adams method, Runge-Kutta, ...) we have settled on a type of adaptive Euler solver which greedily increases its stepsize, but is limited by a function which looks for signs of instability and other undesirable behaviors. We were able to eliminate the more advanced but computationally more expensive ODE modules, without compromising qualitative accuracy.

For example, the solver rejects differential equation steps that cause a cell to move more than a specified distance (typically 5% of a cell diameter). This prevents cells from tunneling through one another and prevents gross instabilities when there are many cells in close contact with one another, with very large forces acting on them. If an ODE step is rejected, the solver tries again with a smaller step-size.

With these restrictions, the simple Euler solver is sufficiently accurate and efficient, particularly when dissipation effects (such as from chemical diffusion and from viscous drag on the cells) dominate the equations. The main advantage of this approach is that gross instabilities are eliminated despite using a very simple ODE solution method. The solutions produced by the simpler method approaches the numerical solution from the more advanced solvers as the step size is reduced. Our modular framework enables us to retain the ability to use the more sophisticated solvers, in case more accurate computation is desired.

Other reasons for using the adaptive Euler solver:

- It copes better with discontinuities such as pseudo-random noise added to the system (Sec. 3.3.2). Some other solvers depend on smoothness of the function to compute higher order derivatives, and thus do not handle this as well.
- It eliminates the 'state-less' requirement. Many solvers assume that they can evaluate $\mathbf{y}'(t)$ at any t. Among other things, this implies that the system must be deterministic (since solvers don't guarantee they will only move forward in t).
- It enables implementation of diffusion by dropping out particles (Section 3.2.6). Since it never retracts more than one step, this is feasible. We do not currently use this feature, but it is being considered as a future direction (Section 3.2.6).

3.3.1 Stability of Diffusion Equation

Diffusion of a chemical concentration u with diffusion coefficient c and dissipation k is governed by the following equation (with $c > 0$ and $k \geq 0$):

$$u_t = c\nabla^2 u - ku$$

Following [Strang, 1986, pages 571-2] we find the stability condition on the Euler stepsize for the two-dimensional diffusion calculation with a dissipation term to be:

$$\Delta t_{max} < \frac{h^2}{4c + k}$$

Let $u^n_{i,j}$ denote the value at grid location (i, j) at timestep n. The stepsize is Δt and grid size is h, c is the diffusion coefficient, and k is the dissipation coefficient. Derivation:

$$\begin{aligned}
\frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t} &= \frac{c(u_{i+1,j}^{n} + u_{i,j+1}^{n} + u_{i-1,j}^{n} + u_{i,j-1}^{n}) - (4c+k)u_{i,j}^{n}}{h^2} \\
u_{i,j}^{n+1} &= (c\frac{\Delta t}{h^2})(u_{i+1,j}^{n} + u_{i,j+1}^{n} + u_{i-1,j}^{n} + u_{i,j-1}^{n}) + (1 - \frac{(4c+k)\Delta t}{h^2})u_{i,j}^{n}
\end{aligned}$$

Positive weights guarantee stability; the condition mentioned above ensures this.

We allow moving sources and sinks, which may haven negative contributions. We constrain their contributions to avoid causing negative concentrations. In addition, there are reaction terms. Although particular choices for sources/sinks and reaction terms may also cause oscillations in the diffusion grid, we do not currently have a mechanism to detect this. In practice, this has not been a problem.

3.3.2 Random Numbers

Important things to consider when using pseudo-random numbers in simulations are:

- Does the basic pseudo-random number generate white noise?
- We must choose noise frequency spectrum (white, $1/f$, etc.).
- The noise function should be a continuous function of time (makes solvers happier).
- The noise should be independent of solver stepsize. One way to do this is to bandwidth limit the noise.
- Is the noise function repeatable? This is helpful for checking simulation results.

We would like to have many independent random numbers as functions of time, $r_i(t)$. This is a bandlimited function with wavelength δt_{rand}. We implement an approximation to this using by creating an array of pseudo-random numbers, which are then interpolated to make a continuous bandlimited noise function. The multiple functions use the same array, but traverse it differently.

Some of the ways pseudo-random numbers are used in the simulator:

- random-sensor-location – to choose a position for a cell's chemical sensor (randomly placed at a specified distance from the center of the cell).
- random-spew-location – position of chemical spewer on cell's boundary
- random-split-angle – angle for dividing a cell
- gc-angle – position on cell where a growthcone is emitted

The user can choose other options for these determining variables; they need not be random. In practice, we often use random-sensor-location, random-spew-location, and random-gc-angle for the simulations shown in Chapter 5. Split angle was often determined by the parent cell based on local enviroment information (as noted in the individual experiments).

Chapter 4

An Example Experiment: the Development of a Neural Pattern Generator

This chapter demonstrates the system in action, so that the reader can get a feel for what it is like to create developmental experiments. The experiments provide examples of the style used when writing programs in this developmental language, which may give us some insights into how real multicellular systems are orchestrated by a genome. This idea of learning about 'genome programming' is discussed further in Section 9.4.

The particular example presented here is a developmental program which creates a small neural circuit. In the end of the chapter we show a version of the circuit which is capable of generating an oscillatory spiking pattern which is characteristic of a "central pattern generator" (CPG) [Getting, 1989, Grillner and Wallen, 1985]. CPGs are neural circuits which control rhythmic behavior in animals, such as locomotion or feeding behavior. These circuits are the subject of intense study in neurobiology.

We begin with the simplest case of two neurons and one connection growing between them (Figure 4.1), and examine the cell state equations of this example thoroughly. Then we generalize the example in various ways, by adding additional neurons and growth cones, and by making slight modifications to the cell state equations.

At the end of the chapter, we show an example of using a spiking neural network simulator [Watts, 1993] to compute the electrical behavior of a fully connected three neuron network.

Contributions. In addition to being an introduction to the use of the simulator, we learn two things from this experiment:

- ◇ we get a feel for how modifications to the cell state equations or initial state bring about the developmental behavior and final structure, and
- ◇ we see that the model is capable of generating networks with potentially useful behaviors.

4.1 Using the System

The simulator has an interactive command line interface [Kay, 1988] and a window-based graphical display. The specific commands to describe each experiment are generally kept in a single file which then serves as a record of the experiment. See Appendix A for descriptions of the CLI commands and some actual experiment files.

The simulator supports a broad range of experiments, in which an artificial organism is allowed to undergo development according to its artificial genome and the given environmental conditions. An experiment is described in a file containing the cell state equations (artificial genome), parameter settings, boundary conditions, and initial state. [1]

As the simulator runs, images of the current state of the simulation are shown on the screen. The user has control over the graphical presentation of cell state variables and environmental variables using various combinations of text, graphical primitives, icons and colors.

4.2 One Neural Connection

This example consists of two neurons. The source neuron sends out a growth cone to connect to the target neuron (Figure 4.1). The growth cone finds the target neuron by climbing the gradient of a diffusing chemical g (shown in green). When it arrives at the target neuron, the growth cone adheres due to the action of an adhesive chemical ($surf\_a$) on the surface of the target neuron. The complementary adhesive chemical, $surf\_a'$, is expressed on the surface of the growth cone.

4.2.1 Neurons: Emit Chemo-attractants, Create Growth Cones, and Be Adhesive

There are two neurons in this experiment which behave differently. One of them creates a growth cone and neurite, but the other does not. Both of these neurons share the same cell state equations; they have identical genomes and intracellular processes. Their behaviors are different because they are different cell types, determined by a regulatory element based on the states of the individual cells.

In this experiment, we define the cell type to simply depend on a state variable z_{ctype} which is set to one for the target neuron and zero for the source neuron. Regulatory elements are then implemented as a condition on this state variable. More complicated and realistic regulatory terms can be defined in a similar way.

The neurons in this experiment have three behaviors:

1. emit a diffusing chemo-attractant which the growth cones can sense,
2. express an adhesive surface chemical so that incoming growth cones adhere and form a synapse, and
3. create a growth cone to make a connection to another neuron.

We consider each of these behaviors separately, creating a cell state equation for each.

[1] After the experiment file is read in, the cell state equations are written into a C file, compiled, and dynamically linked into the running executable. This is much faster than interpreting the mathematical functions in the state equations on the fly.

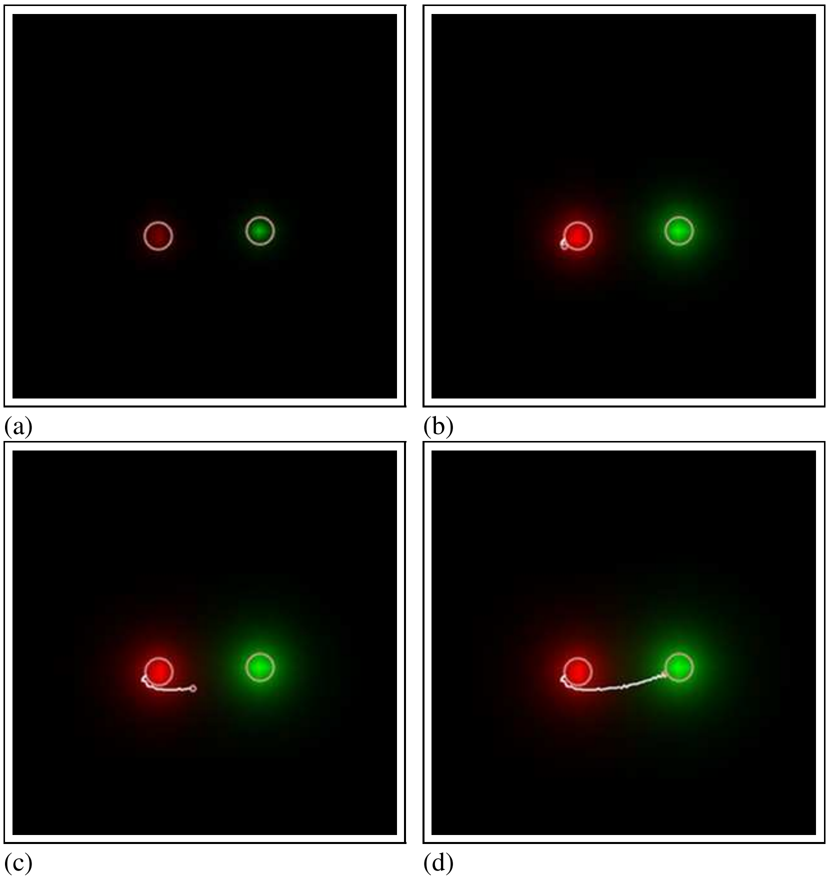


Figure 4.1: Snapshots in time of a neurite growing from a source neuron to a target neuron. The target neuron is emitting a diffusing chemical g which is displayed in green, and the source neuron is emitting a diffusing chemical r which is displayed in red. These neurons share the same genome (cell state equations), since they are part of the same organism. Regulatory elements determine their differences based on their state. □

Express adhesive surface chemical. Let's have both neurons express the adhesive surface chemical $surf_a$ at a fixed concentration (say, 1.0 in nondimensional units). The concentration of surface chemicals is directly controlled by state variables, so we simply designate a state variable which is to represent the surface chemical (z_{surf_a}), and then define a differential equation to bring to the concentration 1.0:

$$\frac{dz_{surf\_a}}{dt} \mathrel{+}= 1.0 - z_{surf\_a}.$$

Since this is the only term for (z_{surf_a}), the += expression behaves as = (there is only one addend). This ODE is stationary and stable at $z_{surf\_a} = 1.0$.

Emit diffusing chemical. Let's say we want our cell to emit a diffusing chemical g at a constant rate G. Recall that the cell behavior function $\mathsf{EmitOrAbsorb}_g$ (Eq. 2.6) sets the rate of emission based on a state variable z_g (Section 2.7). Then we simply need to create a cell state equation to cause the state variable z_g to approach the value G:

$$\frac{dz_{emit\_g}}{dt} \mathrel{+}= G - z_{emit\_g}.$$

This will cause the cell behavior function to emit the chemical at the desired rate.

However, the source neuron should not emit chemical g, since that would cause the growth cone to stay nearby. We add a condition to the z_{emit_g} equation so that the source neuron does not emit g. The source neuron has a cell type determined by $z_{ctype} = 0$, and we can distinguish it from the other cell type $z_{ctype} = 1$ using $(z_{ctype} \tilde{>} 0.5)$. This term will be close to one for the target neuron, so the target neuron will emit chemical g. Thus we have:

$$\frac{dz_{emit\_g}}{dt} \mathrel{+}= (z_{ctype} \tilde{>} 0.5)\,(G - z_{emit\_g}).$$

We can define a second diffusing chemical r being emitted from the source neuron at rate R in an analogous way. Note that the condition is now reversed, so that the target neuron does not emit chemical r.

$$\frac{dz_{emit\_r}}{dt} \mathrel{+}= (z_{ctype} \tilde{<} 0.5)\,(R - z_{emit\_r}).$$

The chemicals r and g are displayed as red and green, respectively, in Figure 4.1.

Creating a growth cone. The final behavior of the neurons in this example is the ability to create a growth cone which will wander off and attempt to make a connection to another neuron. Again, recall that the cell behavior function TimeToEmitNeurite (Eq. 2.8) determines when a growth cone is created by the timing of the zero crossing of the state variable z_{emitgc} (Section 2.7). So we create a differential equation for z_{emitgc} which causes it to cross the threshold *emitgc_threshold*:

$$\frac{dz_{emitgc}}{dt} \mathrel{+}= (z_{clock} \tilde{<} 10.0)\,(1.1\ emitgc\_threshold)$$

$$\frac{dz_{emitgc}}{dt} \mathrel{+}= \; - z_{emitgc}$$

In this case, we take advantage of the += notation to split the equation into two terms. While the condition $(z_{clock} \tilde{<} 10.0)$ is true, both terms are included in the sum. When $(z_{clock} \tilde{<} 10.0)$ becomes false, only the second term contributes, so the concentration of z_{emitgc} drops exponentially to zero.

In the preceding equations, we used another state variable to help determine the time of growth cone emission, z_{clock}. The equation for this state variable implements a cellular clock. Remember that this is at the discretion of the user – this is a simple one, but more complex models can be easily inserted if we they can be written as a differential equation.

$$\frac{dz_{clock}}{dt} \mathrel{+}= 1.0$$

We can add another condition so that only the source neuron emits the growth cone, as we did above for emitting the diffusing chemical. Alternatively, we can determine this by changing the initial value of z_{clock} to be greater than 10.0 for the source neuron. We will do this, just to exhibit a different way of achieving the behavior.

In summary, we have the following equations for the neurons:

$$\begin{aligned}
\frac{dz_{emitgc}}{dt} &\mathrel{+}= (z_{clock} \mathrel{\tilde{<}} 10.0)\,(1.1\ emitgc_threshold) \\
\frac{dz_{emitgc}}{dt} &\mathrel{+}= -z_{emitgc} \\
\frac{dz_{clock}}{dt} &\mathrel{+}= 1.0 \\
\frac{dz_{emit_g}}{dt} &\mathrel{+}= (z_{ctype} \mathrel{\tilde{>}} 0.5)\,(G - z_{emit_g}) \\
\frac{dz_{emit_r}}{dt} &\mathrel{+}= (z_{ctype} \mathrel{\tilde{<}} 0.5)\,(R - z_{emit_r}) \\
\frac{dz_{surf_a}}{dt} &\mathrel{+}= 1.0 - z_{surf_a}
\end{aligned} \tag{4.1}$$

Equation 4.1: Cell state equations for neurons in the two neuron experiment. □

Together with the initial conditions

$$\begin{aligned}
\text{Target :} \quad & z_{clock} = 11.0 \\
& z_{ctype} = 1.0 \\
\text{Source :} \quad & z_{clock} = 0.0 \\
& z_{ctype} = 0.0
\end{aligned} \tag{4.2}$$

Equation 4.2: Initial states for neurons in the two neuron experiment. All other state values are initialized to zero. □

This completes the cell state equations and initial state for the two neurons of Figure 4.1. Now we proceed to the definitions of the cell state equations for the growth cone.

4.2.2 Growth Cones: Chemotaxis and Adhesion

In this experiment, the growth cones find their targets via chemotaxis – they climb a concentration gradient to find a target neuron, and then adhere to it via an adhesive surface chemical. We give the growth cones three behaviors:

1. move up the concentration gradient of a chemical with a neuron emits,
2. express an adhesive surface chemical, so it adheres to the target neuron, and
3. die (retract) if a target is not found.

Since our growth cones are modeled similarly to small cells (with a few restrictions), the cell state equations used to determine their motion and behavior are of the same form as for cells (Section 2.8).

Chemotaxis of growth cones. The following cell state equations specify chemotaxis by setting the time-varying behavior of the state variables z_{fx} and z_{fy}. We will combine these into a vector $\mathbf{z}_f = (z_{fx}, z_{fy})$ for the purposes of this example. Recall that the behavior function MotiveForce uses $\mathbf{z}_f$ to determine the force a cell exerts in an attempt to move (see Section 2.7 and Eq. 2.6).

The concentration gradients of extracellular chemicals are available as inputs to the cell state equations, via the local environment variables (where env[i] denotes the *i*th environment variable, as described in Section 2.6). In the interest of brevity, the following equations use ∇g to represent the concentration gradient (env[gdx], env[gdy]). Gradients of other chemicals are denoted similarly. Note that the concentrations of extracellular chemicals r and g are shown in red and green, respectively, in the figures.

$$\begin{aligned} d\mathbf{z}_f/dt \quad &+= \quad (\mathsf{env[bound\_surf\_a']} \tilde{<} 0.05)\,(a_0\tfrac{\nabla r}{|\nabla r|} + a_1\tfrac{\nabla g}{|\nabla g|} + a_2\tfrac{\nabla b}{|\nabla b|}) \\ d\mathbf{z}_f/dt \quad &+= \quad -\mathbf{z}_f \end{aligned} \tag{4.3}$$

Equation 4.3: Movement eqns for growth cones of three neuron CPG experiment. r, g, and b denote the concentrations of three diffusing chemicals present in the extracellular environment. □

The parameters a_i control the growth cone's movement relative to the gradients of three chemicals. If a_0 is positive, then the growth cone is attracted to r. If a_0 is negative, it is repelled by chemical r. For this first experiment where the growth cone goes from the red-emitting neuron to the green-emitting neuron, we choose $a_0 = -0.1$, $a_1 = 0.3$ and $a_2 = 0.0$. Thus this growth cone is repelled by r, attracted to g, and is not affected by b.

The conditional term (env[bound_surf_a′] $\tilde{<}$ 0.05) is true until the growth cone touches a cell with the complementary adhesion molecule (named surf_a). When the growth cone contacts the target cell, the complementary surface chemicals bind and adhere. The amount of bound chemical is reported in the variable env[bound_surf_a′].

Note that we specify the equation for $d\mathbf{z}_f/dt$ as two += statements. This is because we conceptualize this process as having two components: one which reduces $\mathbf{z}_f$ (the desire to move) and another which increases it (in the direction of the gradient). Once the growth cone is in contact with a cell, the first term turns off, and the second term drives $\mathbf{z}_f$ to zero.

Other growth cone behaviors. Here are the two remaining behaviors for the growth cone. It expresses an adhesive chemical surf_a', the complement of surf_a, to attach to the target neuron. The growth cone also has an equation which causes it to die if it doesn't connect to a target neuron after a certain time.

$$\begin{aligned} \frac{dz_{surf\_a'}}{dt} \quad &+= \quad (\mathsf{env[g]} \tilde{>} 0.015)\,(1.0 - z_{surf\_a'}) \\ \frac{dz_{die}}{dt} \quad &+= \quad -0.03\, z_{die} \\ \frac{dz_{die}}{dt} \quad &+= \quad 0.03\,(\mathsf{env[surf\_a']} \tilde{<} 0.05)\,(1.1\; die\_threshold) \end{aligned} \tag{4.4}$$

Equation 4.4: Cell state equations for death and adhesion of growth cones in the CPG experiment. □

The death equation is again done in two parts. If (env[surf_a′] $\tilde{<}$ 0.05) is true, then the growth cone is not in contact with the cell, and the two terms for z_{die} are both active. Then the differential equation causes z_{die} to approach (and eventually cross) the event trigger threshold *die_threshold*, at which point the growth cone dies. Once the growth cone is in contact with the target cell, the activating term turns off, and the amount of 'death' chemical z_{die} declines to zero. Section 9.4.2 discusses alternate methods of implementing this sort of process, where a chemical accumulates towards a threshold, influenced by combinations of conditions.

4.2.3 Putting It All Together

The cell state equations given above for neurons and growth cones define the developmental behavior shown in Figure 4.1. There are also several parameters which need to be defined, such as initial location and size of the neurons, diffusion coefficients, colors for display, etc., but the basic functionality is described in the cell state equations.

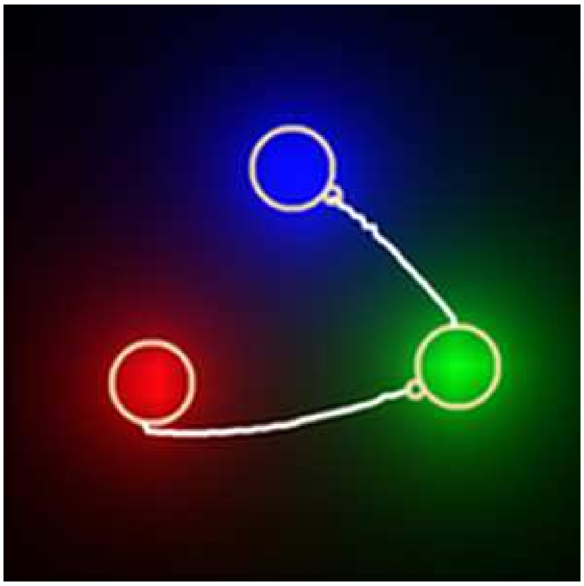


Figure 4.2: Three neurons and two neurites. □

4.3 Three Neurons and Two Connections

Figure 4.2 shows an extension to the previous cell state equations. Another cell type is defined, z_{ctype} = 2, which emits a chemical b (drawn in blue). The combined state equations are below, with the modified or new equations indicated:

$$\begin{array}{rlll}
 & \frac{dz_{emitgc}}{dt} & += & (z_{clock} \stackrel{\sim}{<} 10.0)\,(1.1\ emitgc_threshold) \\
 & \frac{dz_{emitgc}}{dt} & += & -z_{emitgc} \\
 & \frac{dz_{clock}}{dt} & += & 1.0 \\
new: & \frac{dz_{emit_b}}{dt} & += & (z_{ctype} \stackrel{\sim}{>} 1.5)\,(B - z_{emit_b}) \\
modified: & \frac{dz_{emit_g}}{dt} & += & ((z_{ctype} \stackrel{\sim}{>} 0.5)\ \widetilde{\text{and}}\ (z_{ctype} \stackrel{\sim}{<} 1.5))\,(G - z_{emit_g}) \\
 & \frac{dz_{emit_r}}{dt} & += & (z_{ctype} \stackrel{\sim}{<} 0.5)\,(R - z_{emit_r}) \\
 & \frac{dz_{surf_a}}{dt} & += & 1.0 - z_{surf_a}
\end{array} \tag{4.5}$$

Equation 4.5: Cell state equations for neurons in the two neuron experiment. □

The initial conditions will change as well, to set z_{clock} such that the g-emitting neuron also makes a growth cone. Also, we must specify the location and size and initial state of the third, b-emitting neuron.

The growth cone emitted by the g-emitting neuron should approach the chemical b. This is achieved by setting the a_0, a_1 and a_2 parameters from Eq. 4.3 appropriately.

4.4 Three Cyclically-connected Neurons

In Figure 4.3, each neuron attracts growth cones from one of the other neural types. Thus a cyclic pattern of connectivity is assured (from red to green to blue in the figure). This is achieved with

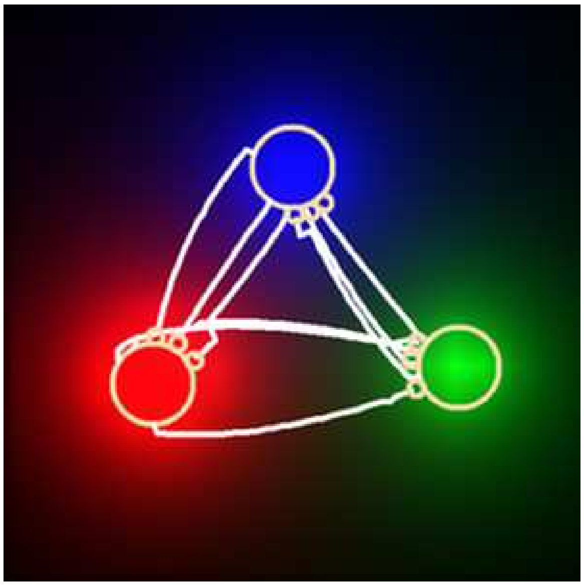


Figure 4.3: Three cyclically-connected neurons. □

no modifications to the existing cell state equations, other than to set the parameters for the growth cones from the b-emitting neuron such that they seek chemical r. The initial state of z_{clock} was set negative so that each of the neurons would emit several growth cones.

To recap, the cell state equations for growth cones from each of the three neural types are:

$$\begin{array}{lcl} r\text{–emitting neuron } (z_{ctype} = 0) & \rightarrow & a_i = [-0.1, 0.3, 0.0] \\ g\text{–emitting neuron } (z_{ctype} = 1) & \rightarrow & a_i = [0.0, -0.1, 0.3] \\ b\text{–emitting neuron } (z_{ctype} = 2) & \rightarrow & a_i = [0.3, 0.0, -0.1] \end{array} \tag{4.6}$$

Equation 4.6: Parameters for growth cone cell state equations Eq. 4.3,

$$d\mathbf{z}_f/dt \mathrel{+}= (\mathrm{env[bound\_surf\_a']} \gtrsim 0.05)\,(a_0\frac{\nabla r}{|\nabla r|} + a_1\frac{\nabla g}{|\nabla g|} + a_2\frac{\nabla b}{|\nabla b|})$$

□

This developmental model consistently makes a cyclically connected three neuron network, even in the presence of some amount of noise or modification to initial conditions. Thus it is robust, in a sense that some other evolving neural network models are not (see further discussion of this in Chapter 7).

What happens if we scale up this example? Clearly this model does not scale easily to create cyclical networks which have more neurons in a single cycle, since it depends on each neuron emitting a characteristic signature for each of the growth cones it wishes to attract. However, there are other simple ways to scale it that are biologically plausible, and lead to networks which are potentially useful. This is the property of **developmental gain**, defined in the introduction (Chapter 1), and discussed further in Chapter 7. An example of this type of scaling is given in the next section.

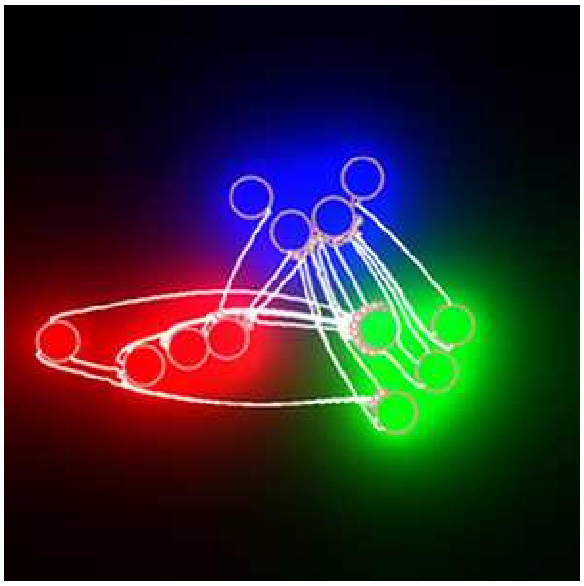


Figure 4.4: Connections between areas (groups of similar neurons). □

4.5 Connections Between Areas, an Example of Developmental Gain

In this experiment, we add more neurons of each type, without adding more types of diffusing chemicals and without changing the cell state equations or growth cones. The neurons act as before, and form connections *between the related groups*, where they used to form connections between single neurons. Thus where we have a single neuron in Figure 4.3, we have a group of neurons in Figure 4.4. Perhaps a process analogous to this is involved in the evolution of area-to-area projections in the brain (where each 'area' is a population of similar neurons). It is this sort of scalability via a developmental process, this developmental gain, which we were aiming to achieve with our model.

Another form of scaling would create many copies of the small circuit shown in Figure 4.3. This could be done by defining a progenitor cell that divided twice to create the three neurons. Then scattering several progenitor cells would create several small circuits.

These two different scaling motifs are analogous to some differences observed in the brain architectures of mammals and birds. Avian brains are generally connected by large parallel projections between areas, while mammalian cortex contains many copies of similar, smaller circuits.

Finding the 'right' neuron within an area. There are still some problems, though. Because these growth cones are simply attracted to the concentrations of diffusing chemicals, they are likely to head for the first neurons they can find, rather than innervating the entire area equally. This can be seen in Figure 4.4, where only two of the g-emitting neurons are innervated, and only a single r-emitting neuron!

Of course, in real neural systems, growth cone guidance is much more sophisticated than this simple chemotactic model. Adhesive factors play a role, as well as mechanical guidance.

We have used adhesive factors to address this multiple-innervation issue in another experiment

Figure 4.5: **Three neuron fully-connected network, with spiking behavior.** The spiking behavior shown in Figure 4.6 was computed from the connections of this network.
A note on the colors: We are using the same color mapping we have been using for the previous figures, with red for chemical r and green for chemical g. Both r and g are emitted by the neuron in the lower left, and the displayed colors combine to form a magenta-colored field around the neuron. Similarly, yellow and cyan appear around the other neurons, which are each emitting two diffusing chemicals. □

shown in Figure 5.20 (Section 5.3.3). In that experiment, neurons which have already been innervated change state to refuse later connections (by losing their adhesiveness).

There are many other combinations of mechanisms one can imagine utilizing to obtain various patterns of innervation. Exploring these is a fruitful avenue for future work.

4.6 Fully Connected Three Neuron Network, with Spiking Computation

In this last variation, we return to the three neuron network discussed above, and modify it in a different way. In Figure 4.5, a minor modification is made to each neural type. Each neural type emits *two* attractive chemicals, so it will probably receive a connection from each of the other neurons. There are no other changes to the cell state equations of the neurons or growth cones.

Note that it is not assured that a neuron will receive connection from both of the other neurons, because now there are two equally attractive targets for the growth cones. There is some chance that all of the growth cones from one neuron will find the same target (due to noise introduced in the sensors, and the initial random location of growth cone creation on the parent neuron).

We show just the modified equations, omitting the ones which do not change. These equations specify that the neuron of $z_{ctype} = 0$ emits r and b, the neuron of $z_{ctype} = 1$ emits r and g, and the neuron of type $z_{ctype} = 2$ emits g and b.

$$
\begin{aligned}
\frac{dz_{emit\_b}}{dt} &\mathrel{+}= ((z_{ctype} \tilde{>} 1.5) \;\tilde{\text{or}}\; (z_{ctype} \tilde{<} 0.5))\,(B - z_{emit\_b}) \\
\frac{dz_{emit\_g}}{dt} &\mathrel{+}= (z_{ctype} \tilde{>} 0.5)\,(G - z_{emit\_g}) \\
\frac{dz_{emit\_r}}{dt} &\mathrel{+}= (z_{ctype} \tilde{<} 1.5)\,(R - z_{emit\_r})
\end{aligned} \tag{4.7}
$$

Equation 4.7: Modifications from Eq. 4.5 for neurons in the fully connected network. □

This approach usually makes a fully connected three neuron network, but it is not robust (it isn't always *fully* connected, as mentioned above). This network's scaling properties are similar to the cyclical three neuron network. It does not scale well to making larger fully connected networks, since it depends on each neuron emitting a characteristic signature for each of the growth cones it wishes to attract. However, it does scale up to having multiply-connected areas.

4.7 A Spiking Neural Model for the Fully-Connected Three-Neuron Network

We extend the developmental model to include spiking neural behavior using **Spike**, a spiking neuron simulator written by Lloyd Watts [Watts, 1993]. From our developmental model we get connectivity information and a value for the delay along neurites (related to length). Feeding these parameters to **Spike** then gives us the continually spiking neural behavior shown in Figure 4.6. This behavior is typical of a pattern generator circuit which might be used to coordinate some rhythmic movement of an organism.

Additional parameters are required for **Spike**, such as refractory period (a particularly critical one). These were not specified by the developmental model as it currently stands. It is straightforward to include more state variables in the developmental model, which can then be mapped to these parameters. This might be of value if the model is being used for artificial evolution.

Heterogeneous Neural Types. Another option is to map the cell types into particular parameters, allowing the creating of networks with heterogeneous neural types. In real neural circuits, neurons differ widely. Neurons of different types, with different electrical characteristics, different morphology, and different connectivity, are often combined in the same circuit. The ability to include this type of structure/behavior in our representation of artificial neural networks was one of our original design goals (Chapters 1 and 7). We have shown that the connectivity of neurons can vary in this representation, and that the electrical properties can also be different.

Another of our original design goals was the creation of asymmetric network connectivity. Clearly, in this developmental representation, asymmetry is the norm, not the exception.

4.8 Summary

This chapter has taken us through a typical experiment with the developmental simulator. Beginning with a simple behavior (e.g., one neural connection), we extend it in various ways and examine the consequences. In situations where the experimental model seems weak, such as the innervation of areas (Section 4.5), we then consider new cell state equations which can compensate.

In the remainder of the thesis, we will not go into such detail with the models, with the exception of the model in Section 6.2. Instead, we will present a high-level description of the cell's equations and behaviors.

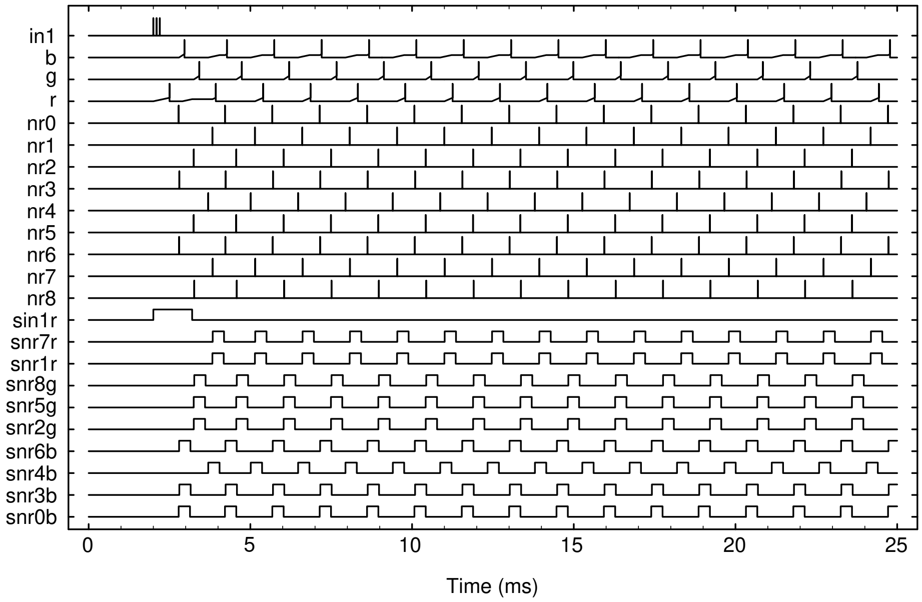


Figure 4.6: Spiking neuron model using the developmentally generated network shown in Figure 4.5.
The labels r, g, and b refer to the three cells emitting chemicals *r*, *g*, and *b*. Labels beginning with nr show activity in neurites. snr shows synaptic activity. An initial pulse to start things off is shown in ini, and after that, the network continues on its own. □

The experience gained from designing many of these simulated experiments may provide some insights into the processes by which cells with only local information self-organize into structured multicellular organisms. Our observations on this appear in Section 9.4.

Developmental gain and robustness in heterogeneous asymmetric neural networks This experiment demonstrates that we have achieved several of our goals. The representation exhibits developmental gain and robustness, and it enables the creation of networks with heterogeneous neural types and asymmetric geometric structure. We believe these properties are an important step towards creating a powerful class of neural networks (discussed further in Chapter 7).

Chapter 5

Exploring Pattern Formation: A Gallery of Experiments

As we mentioned in Chapter 2, the developmental model is capable of representing a wide range of developmental phenomena.

Since each experiment may exhibit multiple phenomena, we organize the chapter as a list of experiments and provide an index of biological behaviors which points at the appropriate experiments in the list. The experiments are loosely ordered so that similar experiments appear in sequence. The final image shown for each experiment is a stable state for that simulated organism, except for the experiment of Section 5.2.7.

In each experiment, we briefly discuss how **size regulation** and **scalability** are exhibited.

Size regulation. Size regulation refers to the processes by which an organism grows to a certain size and shape, and then maintains that shape. Any simulation which includes cell division must include a method of terminating or regulating cell division, otherwise the organism keeps dividing until it fills the 'virtual petri dish' (like a cancer). As in real organisms, cell division must be regulated to form a coherent shape.

Scalability. Scalability refers to the scaling properties of the developmental simulation in terms of the parameters to the state equations. For example, in the CPG experiment (Chapter 4) we examined the scaling properties by changing the initial number of cells and by changing values of parameters. Studying these properties gives us some insight into how complex organisms might be composed from simpler developmental strategies.

A note on the meaning of cell type. In this chapter we often discuss the behaviors of various cell types in an experiment. It is important to remember that cell type is an interpretation that we make as observers.

> "Two cells are differentiated with respect to one another if, while they harbor the same genome, the pattern of proteins they synthesize is different."
> F. Jacob and J. Monod, 1963

Sometimes in our simulation experiments, we explicitly create a state variable z_{ctype}. For certain ranges of values of z_{ctype}, we turn on or off several cell state equations, causing different behaviors. We then refer to the cells in the different behavioral regimes as different cell types, which can be distinguished by the value of z_{ctype} within each cell. Note that the use of z_{ctype} is a convenience. We could interpret cells as being different types using some other criterion, based on the values of other state variables or based on the cell behaviors. There is no need to use just one state variable as the determinant of cell type.

What does it mean for a process to be *difficult* for an organism? It is interesting to consider whether a particular process that is difficult to create in the simulator is a difficulty for real organisms, and even what it means for something to be 'difficult' for a real organism. One definition that seems reasonable is that process A is more difficult than process B if it requires a finer tuning of a parameter, or if it depends on more parameters.[1] This corresponds to difficulty over evolutionary time, for parameters which are set in the genome or development of an organism. We could quantify difficulty of a process like limb formation by how frequently it fails (generates a mutation with, say, six fingers or no thumbs).

Turning this around, we can state it as a goal for a simulation system: things which are difficult in the real system should be difficult in the simulator, and vice versa. Whether this has been achieved is an open question for investigation, which we do not consider further here.

5.1 Biological Behaviors Exhibited by the Model

We show a variety of examples of biological phenomena which can be elicited in the system:

- ◇ chemotaxis: Sec. 5.3.1, Sec. 5.3.3, Sec. 5.2.2
- ◇ lateral inhibition: Sec. 5.2.3, Sec. 5.2.4, Sec. 6.2 and Sec. 5.2.11
- ◇ morphogenetic change by mechanical means: Sec. 5.2.11
- ◇ differentiation: Sec. 5.2.3, Sec. 5.2.2, Sec. 5.2.11
- ◇ size regulation: Sec. 5.2.5, Sec. 5.2.1, Sec. 5.2.9, Sec. 5.2.2
- ◇ differential adhesion: Sec. 6.4
- ◇ reaction-diffusion: Sec. 5.2.3, Sec. 5.2.4, Sec. 6.2
- ◇ formation of segments: Sec. 5.2.11
- ◇ formation of three-dimensional structures: layers Sec. 5.4.2, and spirals Sec. 5.4.3
- ◇ axis formation: Sec. 5.2.11, Sec. 5.4.2

Simulation experiments that combine building blocks used in other examples:

- ◇ Section 5.2.2 combines 'spread out' and 'early chains'.
- ◇ Section 5.2.11 combines '1-2-1 chains' with segmentation and motion.
- ◇ Section 5.3.2 combines simple neural connections Section 4.3 with differentiation.
- ◇ Section 5.3.3 combines the neural growth model from Sec. 5.3.2 with a limitation on cell division.

Simulation experiments that develop into an artificial organism with a behavior:

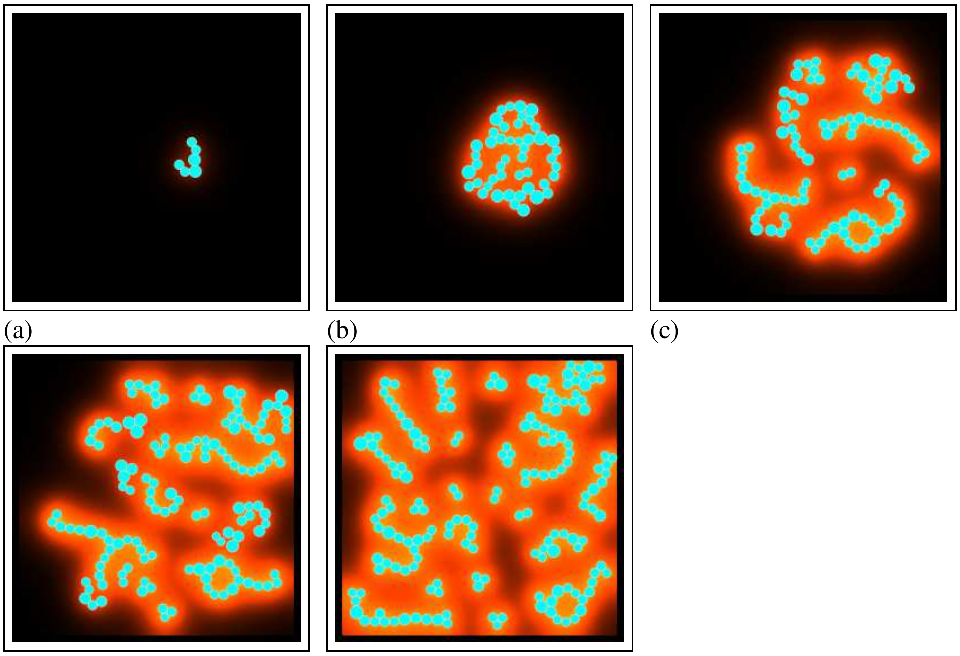
[1] Certainly other definitions of difficulty could be suggested (e.g., energy requirements), but this one fits most closely with the concept of difficulty when creating cell state equations.

- ◇ a worm-like collection of cells which moves (Section 5.2.11),
- ◇ a group of cells which coalesce and disperse rhythmically (Section 5.2.7),
- ◇ a cluster of heterogeneous cells which regenerate if some are killed (Section 5.2.5), and
- ◇ a spiking neural network which computes (Section 4.6).

5.2 Simulation Experiments: Basic Mechanisms

The simulations were computed on Hewlett Packard 9000 series 700 and 800, IBM RS6000, and DEC Alpha computers. The running times range from several seconds to a few hours.

5.2.1 Chains of cells via a combination of adhesive and chemotactic mechanisms



(a) (b) (c)

(d) (e)

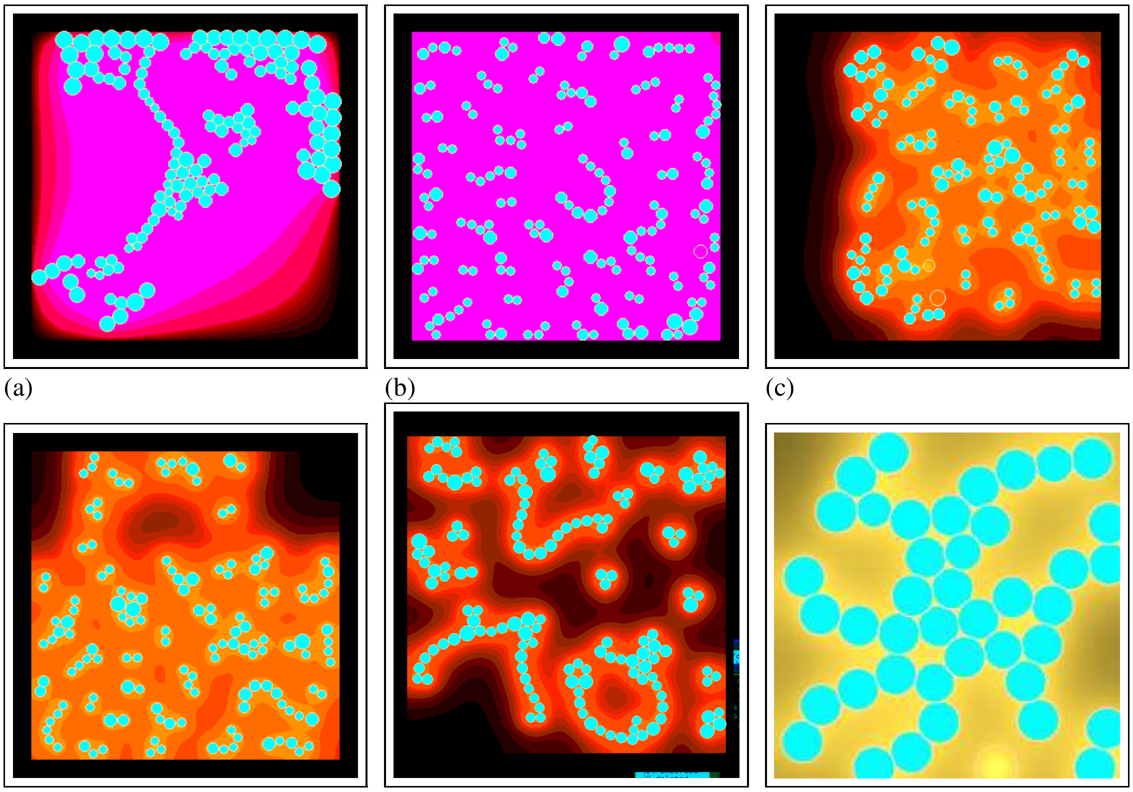
Figure 5.1: Chains of cells formed by the interaction between an adhesive surface chemical and a negative chemotaxis from a repulsive diffusing chemical which the cells themselves emit. The progression in time is shown in (a)-(e). □

This experiment shows how the ratio of forces due to two different mechanisms can be effective in creating a pattern. The two opposing forces are an attractive force due to cell adhesion, and a repulsive force related to a diffusable chemical. The chains of cells are held together by the adhesive force, but they also attempt to move away from each other due to the repulsive diffusing chemical. All of the cells are emitting the chemical, so areas where there are more cells tend to have more of the slowly diffusing chemical. This interaction leads to a pattern of chains of cells, or very small clumps.

The cells divide when the concentration of the chemical is low, and when there is only one other cell in contact (this is determined by a threshold on the amount of a bound surface chemical). Thus the cells at the ends of a chain are the only ones which can divide. Cells in regions of less chemical concentration tend to divide more frequently. This serves to regulate the size of the chains.

Note that a group of three cells clumped together in a triangular formation is stable and stops dividing. In the final panel (e) several triangular groups of cells can be observed.

Size regulation in this example is via limiting processes on the diffusing chemical, as well as the contact constraint. Since the 'virtual petri dish' is of limited size, the concentration of the diffusing chemical eventually reaches a level which halts cell division. If the dish were enlarged, the chains would grow to fill the dish.



(a) (b) (c)

(d) (e) (f)

Figure 5.2: Similar chains of cells, with parameter variations. The hue differences between the different simulations are not meaningful. □

5.2.2 Compartments

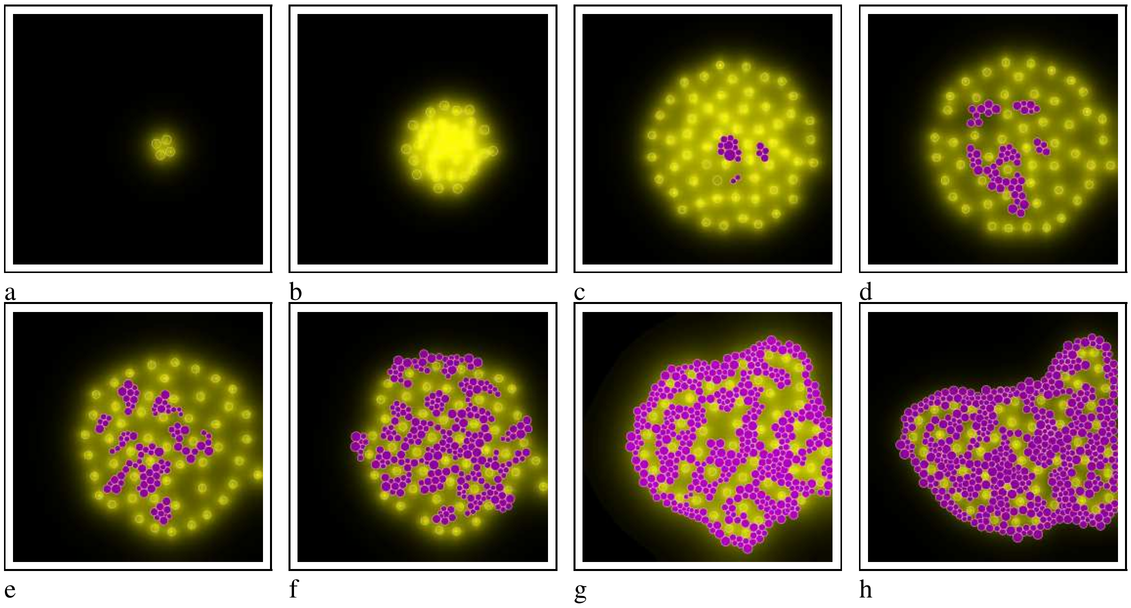


Figure 5.3: Compartments: time lapse of development. □

This sequence begins with a single cell which divides for a pre-defined number of generations. These first cells (call them type Y) emit a diffusing chemical y (shown in yellow). The Y cells have a preference for a certain concentration range of $y_0 < y < y_1$. The Y cells move down the y gradient if the concentration of y is too large ($y > y_1$), and up the gradient if it is too small. This behavior forms a pattern of roughly equally spaced cells (the yellow cells in panel (c)).

Cells at the periphery are able to reach a comfortable state (y within the desired range) more quickly than the cells in the center of the pattern. Cells in the center might still experience a high concentration of y (greater than y_1), but the gradient will be small since they are surrounded by other y emitters. Since they move proportionally to the gradient, this means they are stationary even though the y level is too high.

Some of the Y cells differentiate into P cells (defined below). P cells are drawn in purple. This occurs after the first phase (the division and spreading out of Y cells). The condition used is:

(stationary) and (too much y) and (1st phase over) and (not much p)

where

- ◇ (stationary) means the cell is not moving,
- ◇ (too much y) means the concentration of y is too large ($y > y_1$),
- ◇ (1st phase over) means the Y cells are no longer dividing, and
- ◇ (not much p) means the concentration of p is small.

The diffusing chemical p is emitted by P cells. Note that this means that a Y cell will not differentiate into a P cell if another Y cell has already differentiated nearby.

This large condition has the effect of only allowing a few Y cells to differentiate. Specifically, the Y cells which are near the center are likely to be the first to differentiate since they satisfy all of the conditions. After the first one differentiates into a P cell, others nearby are inhibited from differentiating by the emission of chemical p. So the next Y cell to differentiate will be some distance away from it. This causes the newly differentiating cells to appear roughly equally distributed in the Y cell cluster.

After a few cells differentiate, the ambient level of p is high enough to prohibit further differentiation by Y cells.

P cells are another cell type. They have several behaviors:

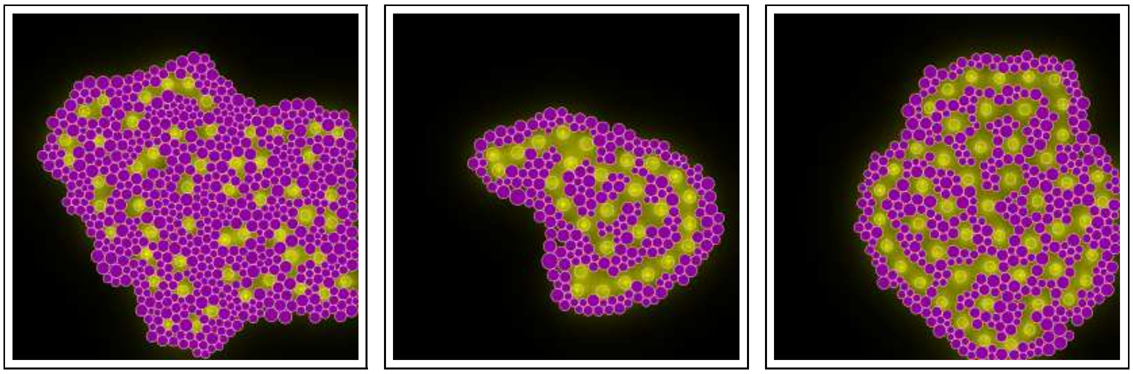
◇ divide for a fixed amount of time,
◇ seek a certain level of chemical y ($y_2 < y < y_3$),
◇ emit chemical p,
◇ attracted to chemical p,
◇ adhere to other P cells using an adhesive homophilic surface chemical.

Thus P cells tend to clump together in regions with a particular concentration of chemical y. The levels y_2 and y_3 have been carefully chosen so that the P cells like to lie between the Y cells.

The P cells are similar to the chain cells in Figure 5.1. In some initial experiments (not shown), they were identical except that they were repulsed by a chemical emitted by the Y cells rather than one they emitted themselves (as in the Section 5.2.1 experiment). This had the unanticipated (but correct) result that the P cells rushed out of the Y cell grid, and ran off to the corners of the dish.

Size regulation is accomplished simply by limiting the number of cell divisions in each phase.

Scaling properties. If y_3 is lowered, then the P cells will leave the Y cells, and for some values of y_2 and y_3 they will form a ring around the Y cells. If the range is raised, the P cells will probably tend to clump around the Y cells.



(a) (b) (c)

Figure 5.4: Variations on the theme. □

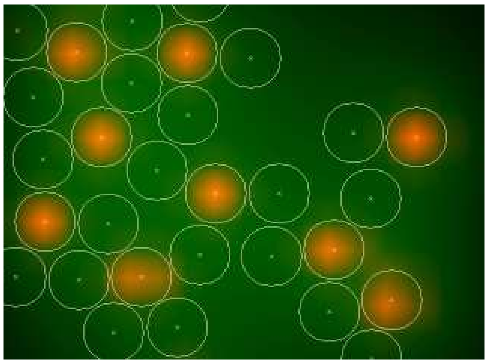


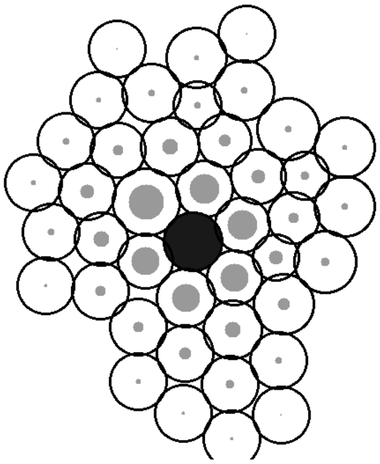
Figure 5.5: Cell differentiation via diffusing chemicals. Concentrations of the two diffusing chemicals are shown in red and green. The cells which are emitting the chemical shown in red are inhibiting the others. Areas with both chemicals are colored yellow. □

5.2.3 Cell Differentiation via Diffusing Chemicals

A single initial cell divides under the control of a cytoplasmically inherited factor (which each cell inherits from its parent). When this is diluted beyond a threshold, the cells stop dividing and begin to emit diffusable chemicals. Each cell has a tendency to become 'activated', which we define to mean that it is emitting the chemical shown in red (for this experiment). Activated cells also emit the inhibitory chemical (shown in green), which prevents their neighbors from becoming activated.

One of the diffusing chemicals is a fast-diffusing lateral inhibitory factor (green), and the other is slow-diffusing and excitatory (red). The combination of lateral inhibition and local positive feedback gives rise to patterns of differentiated cells. This is similar to the behavior of reaction-diffusion equations [Turing, 1952, Meinhardt, 1982, Murray, 1993], except that the model here uses discrete cells instead of a continuous cell sheet.

Initial conditions and environmental variations account for which cells are selected, creating a heterogeneous population. This example exhibits reaction-diffusion behavior in a discrete cell model (Section 6.2.1).



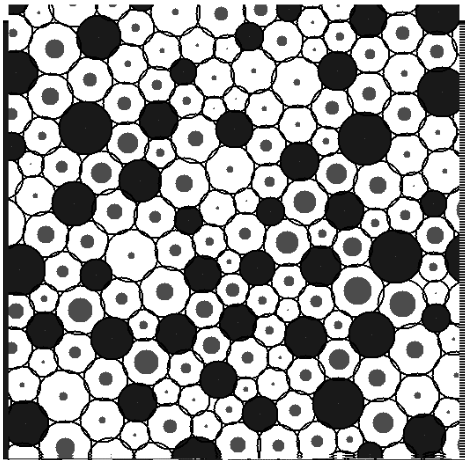


Figure 5.6: (Left) Long-range lateral inhibition with surface chemicals. The dark gray cell is activated; all others are inhibited. The amount of inhibition a cell is experiencing is depicted by the size of the light gray circle. □

Figure 5.7: (Right) Short-range lateral inhibition with surface chemicals. The dark gray cells are activated; all others are inhibited. The amount of inhibition a cell is experiencing is depicted by the size of the light gray circle. □

5.2.4 Lateral Inhibition via Surface Chemicals

Figure 5.6. The central (dark gray) cell is inhibiting its immediate neighbors using a surface chemical. They, in turn, inhibit their neighbors. The amount of inhibition each cell is experiencing is indicated by the size of the light gray circle within the cell. The inhibition is slightly reduced as it is passed along.

Unlike the previous example, which used diffusing chemicals for the inhibition, in this example the lateral inhibition is done via interaction of surface chemicals. With this paradigm, cells must be in direct contact to inhibit each other.

Figure 5.7. Here we see the same image of a simulation using the same state equations as Figure 5.6, but different initial conditions. Now the inhibition is only local. Section 6.2 contains a detailed discussion of this phenomenon, and a discussion of the relationship between continuous and discrete cell models for reaction-diffusion is also considered.

5.2.5 Hierarchical Structure Which Regenerates when Damaged

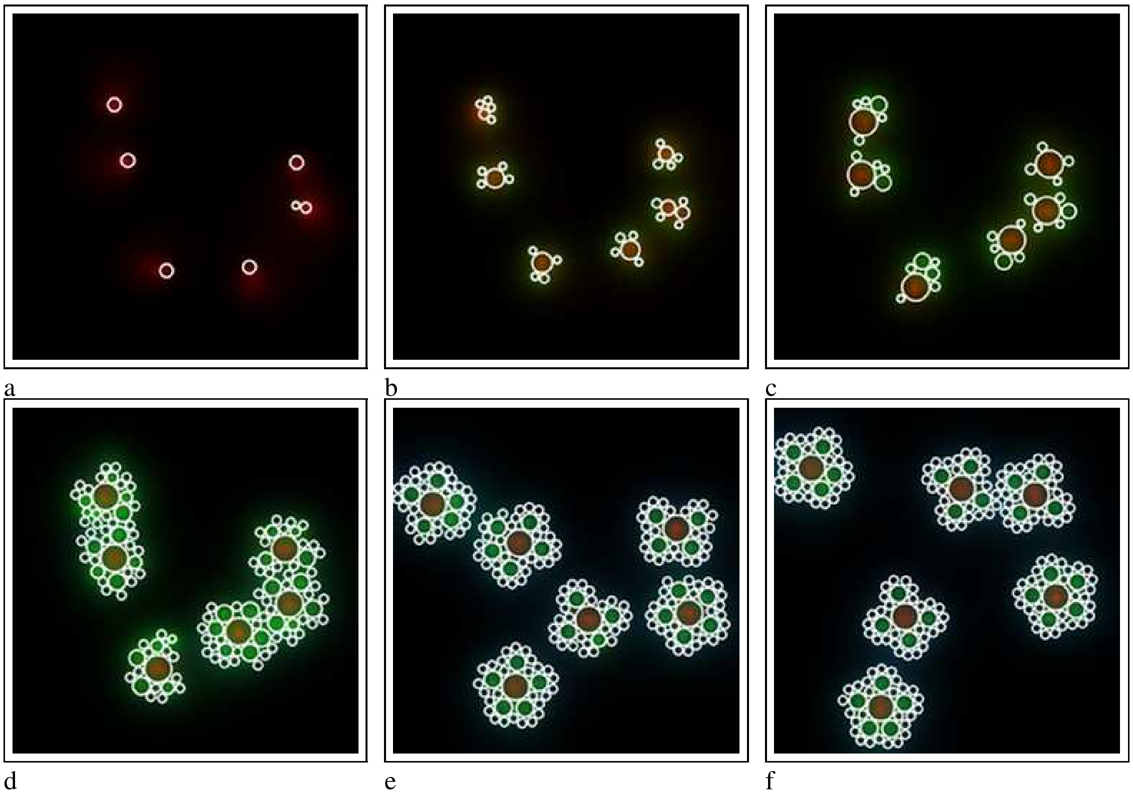


Figure 5.8: Generation of hierarchical structures. □

In this example there are three cell types, R, G, and W.

- ◇ Type R cells are the large cells emitting the diffusing chemical shown in red.
- ◇ Type G cells are the medium-sized cells emitting the diffusing chemical shown in green. They cluster around the R cells because they are attracted to the chemical emitted by the R cells, and they adhere to the R cells. A G cell dies if not in contact with an R cell.
- ◇ Type W cells are the little cells clustered around the G cells. They are attracted to the chemical emitted by the G cells, and they adhere to the G cells. A W cell dies if not in contact with a G cell.

Contact chemicals as well as an attractive diffusing chemical are used to keep cells attached to each other. Each pair of cell types has their own complementary pair of surface chemicals. So there is an RG surface chemical, and a GW surface chemical.

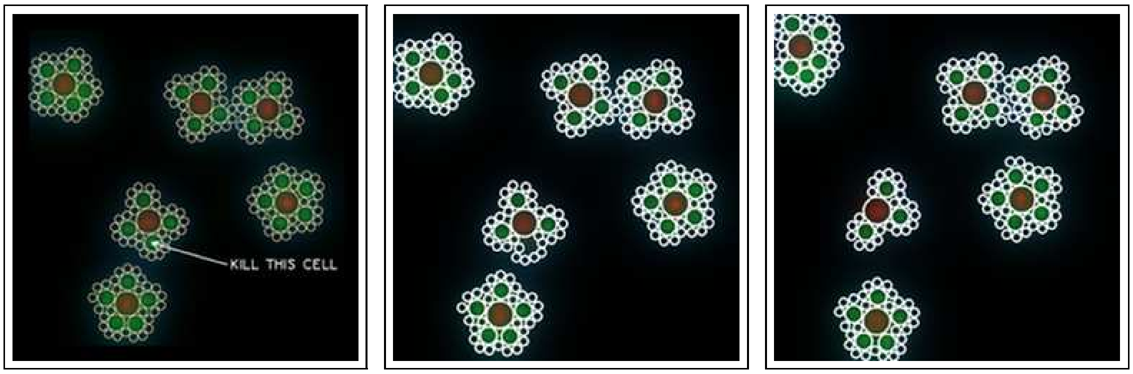
The simulation starts with six randomly placed R cells. Each R cell divides into an R and a G when there are too few G's attached to it (as in panels (a) to (c) above). "Too few" is determined by a threshold on the amount of RG surface chemical bound on the membrane of the R cell. The cell state equation for this is of the form:

$$dz_{split}/dt \quad += \quad (\mathsf{env[surf\_RG]} \mathrel{\tilde{<}} \alpha)\ (1.1\ \mathit{split\_threshold}) \tag{5.1}$$

Equation 5.1: Split condition for R cells in hierarchical structure simulation. □

Each G cell divides into a G and a W when there are too few W's in contact with it, in a similar manner. W cells do not divide.

Noise and differing environmental conditions cause each of the six 'organisms' to be slightly different. Some have four lobes, some have five.

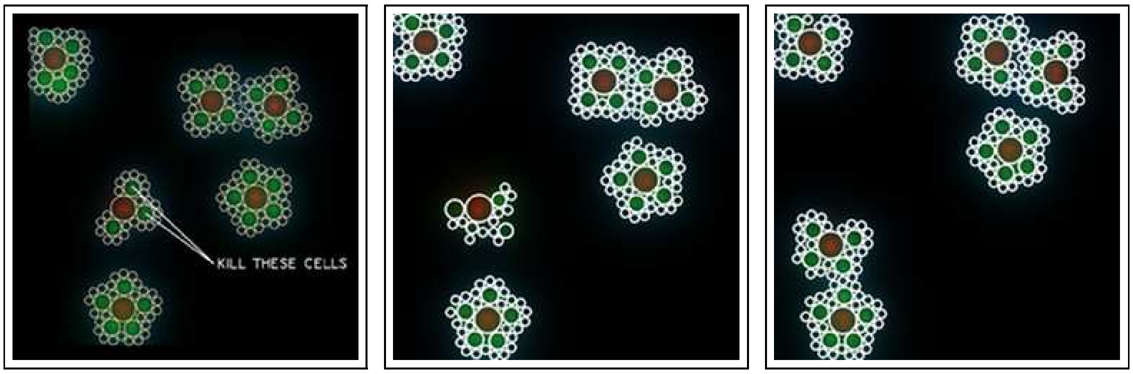


b a c

Figure 5.9: No regeneration occurs if only one cell is killed (hysteresis). □

In panel (b) above, we explicitly kill a G cell so that we can observe the resulting behavior of the organism. Panel (c) shows the simulation at a later time. We can see that several W cells have died (those that were attached to the G cell which was killed).

The death of a single G cell may or may not cause the parent R cell to begin dividing and creating new G cells. This depends on whether the amount of bound R-G surface chemical has crossed its threshold (α in Eq. 5.1). In this case, the R cell is still above threshold, and is not dividing. So we see that a three-lobed organism is also stable.



a b c

Figure 5.10: When two more cells are killed, the structure regenerates. □

In Figure 5.10 we see that killing two more G cells does cross the threshold, causing the R cell to regenerate several G cells, and they, in turn, divide to create their W cells, until a statis is obtained.

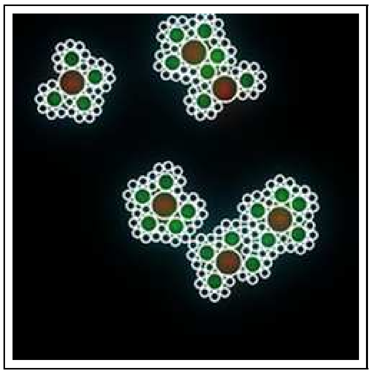
Size regulation in this experiment is via cell-cell communication. Cells divide if they have insufficient neighbors. Unlike the experiment in Section 5.2.1, enlarging the 'virtual petri dish' here will not lead to more cells. The organisms achieve a stasis independent of the environment.

Scaling properties. There are several ways we can imagine scaling this example. Adding more initial cells clearly adds more organisms. It is more interesting to consider modifying parameters to the cell state equations. For instance, if we lower α (the cell-division threshold for the R cells), then the R cells will stop dividing sooner, and will tend to create organisms with fewer lobes.

If we raise α then the R cells may never stop dividing (the threshold is never reached). Only a few G cells can remain in contact with a single R cell (due to the geometric constraints). The remaining G cells will die if they cannot maintain contact with the R cell.

Another method of scaling this example would be to allow the R cells to divide. This would enable our organisms to reproduce, which could be a foundation for some future experiments in artificial evolution.

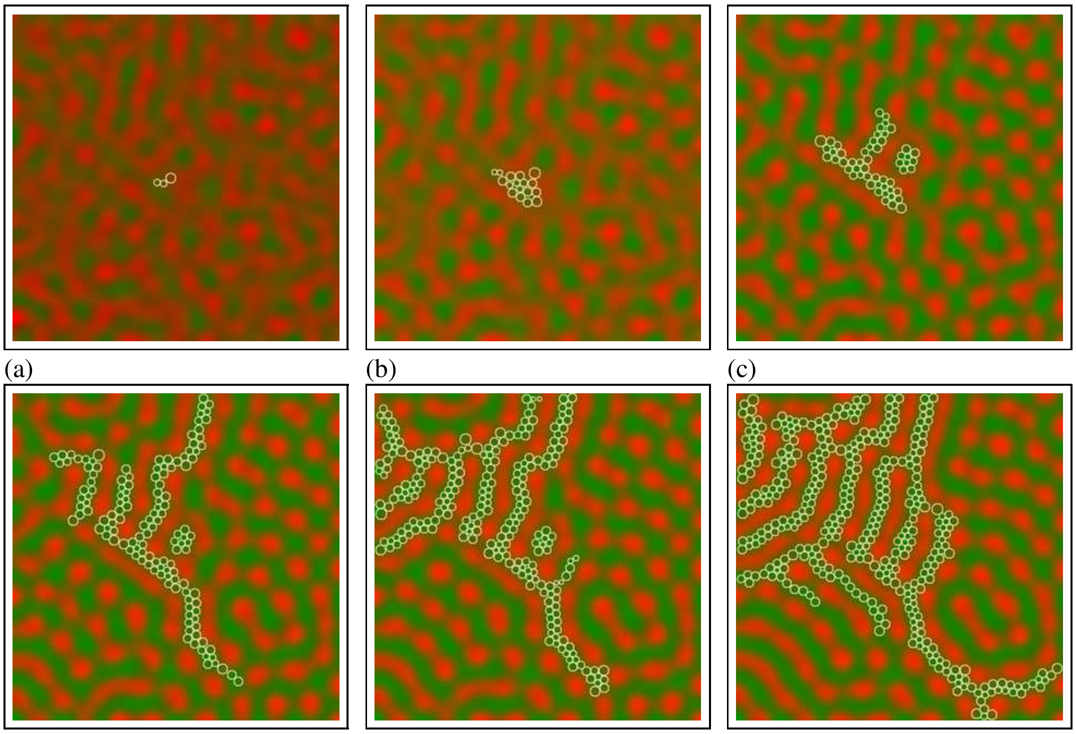
Adding another level of hierarchy is not so straightforward. Many modifications to the cell state equations are required, since each level of the hierarchy is implemented with its own surface chemical (R-G or G-W) and diffusing chemical.



a

Figure 5.11: R cells can 'share' a G cell. □

In Figure 5.11, note that the two organisms in the upper center part of the figure are sharing a G cell. Since the R cell division threshold depends only on surface contacts, it can be satisfied by contact with any G cell. It does not matter that the G cell is in contact with another R cell.



(a) (b) (c) (d) (e) (f)

Figure 5.12: Cells growing on reaction-diffusion patterns. □

5.2.6 Cells Growing on Reaction-Diffusion Patterns

This experiment shows the use of a reaction-diffusion computation to provide a pre-pattern that guides later cell growth. The red and green colors indicate concentrations of two diffusing and reacting chemicals.

In this experiment there are two chemicals r and g. The two reaction functions shown below describe how each chemical is affected by the concentrations. In addition to this function occuring everywhere spatially, diffusion is being computed (see Eq. 3.3). This equation is from [Murray, 1993, Equation 14.7].

$$\begin{aligned} \mathsf{R}_r(r,g) &= \gamma\,(a - r + r^2 g) \\ \mathsf{R}_g(r,g) &= \gamma\,(b - r^2 g) \end{aligned} \tag{5.2}$$

Equation 5.2: Reaction functions for the reaction-diffusion patterns. Values for parameters: $a = 0.1$, $b = 1.5$, and $\gamma = 0.4$. The relative diffusion coefficient for g with respect to r is 20.0. □

The implementation of this general user-definable reaction function R described in more detail in Sections 2.6 and 3.2.6.

The cells are dividing only where the concentration of g is large (shown in green). They are also climbing the gradient of g, and avoiding chemical r.

This example shows that the system can be used to study cell migration based on a reaction-diffusion pre-pattern. In this experiment, the cells do not emit or absorb the extracellular chemicals r and g. In the future, we plan to explore the interaction of the reaction-diffusion process and the emission of the chemicals by the cells.

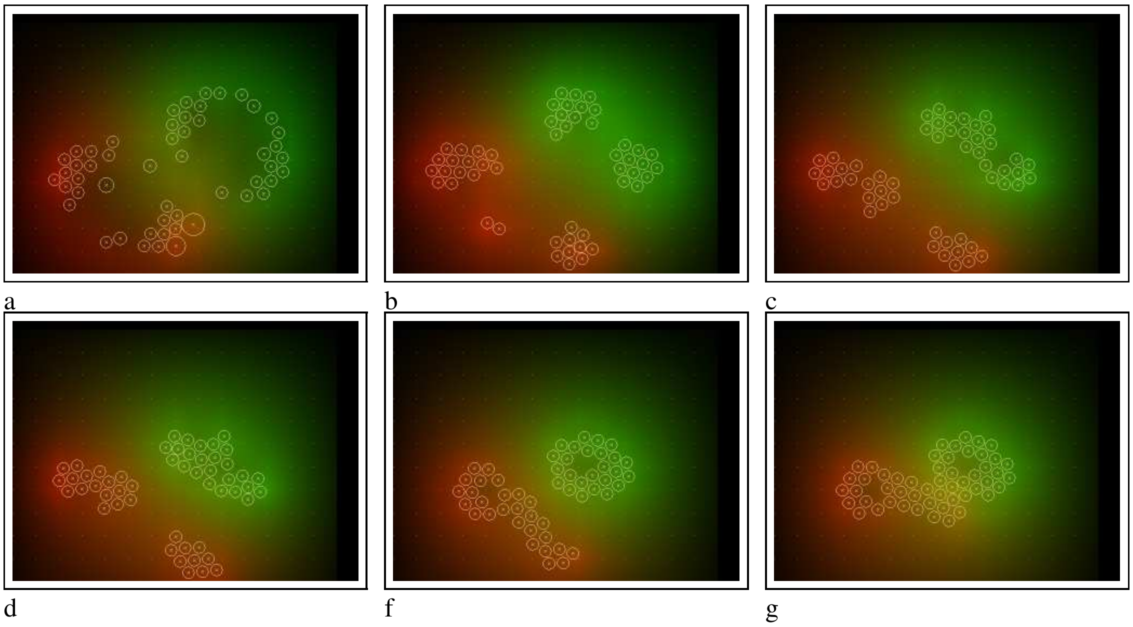


Figure 5.13: Cyclic behavior. This example does not acheive a stable state. □

5.2.7 Cyclic Motion

We include this example to emphasize that it is not trivial to predict what a given artificial genome will do, nor is it easy to concoct by hand a genome to create a particular pattern.

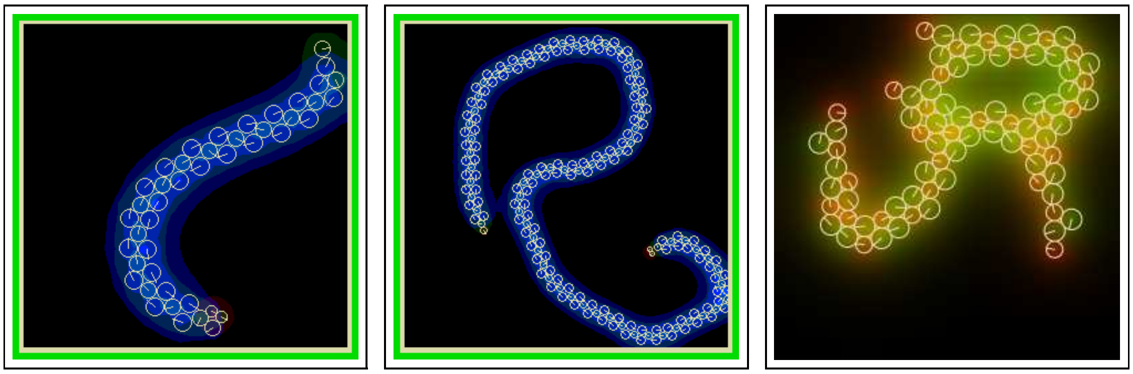
In Figure 5.13, we see an attempt to regulate size using diffusable chemicals. Instead of regulating the size, this experiment unexpectedly resulted in cyclic behavior. There are two cell types: one emits and seeks the chemical shown in red, the other emits and seeks another chemical (shown in green). The cells of the two types do not interact in this example, other than an occasional collision.

The amount of chemical emission for each cell type is regulated by the presence of its respective chemical. This specification forms clusters; however, they are not stable. Once the clusters form, the cells in the center of the cluster begin to reduce their emission of the chemical, which causes a local depression. The gradient then leads these cells out from the center, forming a ring. The ring pattern is not stable either, and the cells tend to cluster together again. This behavior repeats cyclically.

5.2.8 Chains of cells via orientation of cell clevage planes

The cell state equations for these organisms form chains of cells using three cell types we'll call types A, B and C. The chain structure is constructed from A and B cells, in the pattern 1A-2B-1A. The C cell type is the progenitor cell at the tip of the chain that divides to extend the chain. It divides to produce a C and a B, then divides to create a B and an A. Then the A divides once to produce an A and a C. The new C cell starts all over again to make a new 'link' in the chain. Contact is maintained between the cells via adhesive surface chemicals. Gradients of diffusing chemicals are used to determine the orientation of the next cell cleavage plane.

Occasional errors occur in the structure (Figure 5.14(c)), especially when the ambient concentrations of diffusing chemicals get larger. When this happens, the gradient which the progenitor cell is using to choose its cleavage orientation is not quite aligned with the axis of the chain, and it may divide at an improper orientation. Sometimes this effect becomes so severe that the cell contacts are not maintained as expected, and two C cells can get created, causing a fork in the chain.



(a) (b) (c)

Figure 5.14: 1-2-1 Chains. □

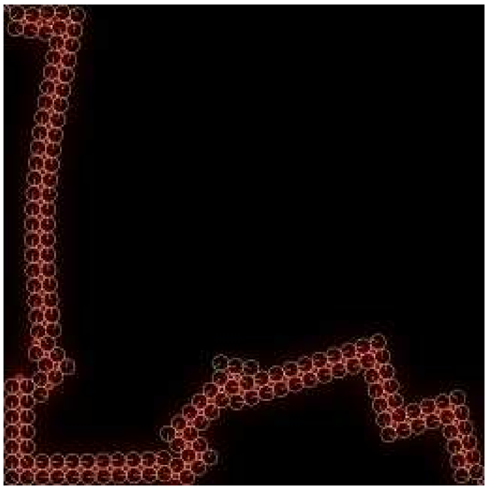


Figure 5.15: Chain with geometric constraint. □

5.2.9 Chain with geometric constraint

This experiment is quite similar to the previous one, with a modification to the sizes of the A and B cells. The small A cells and large B cells here fit together tightly, constraining the chain to be nearly straight. Occasional errors in the cleavage orientation (as discussed above) give rise to right-angle turns. The use of geometric constraints to determine organism shape is discussed further in Section 9.3.4 and Figure 9.1.

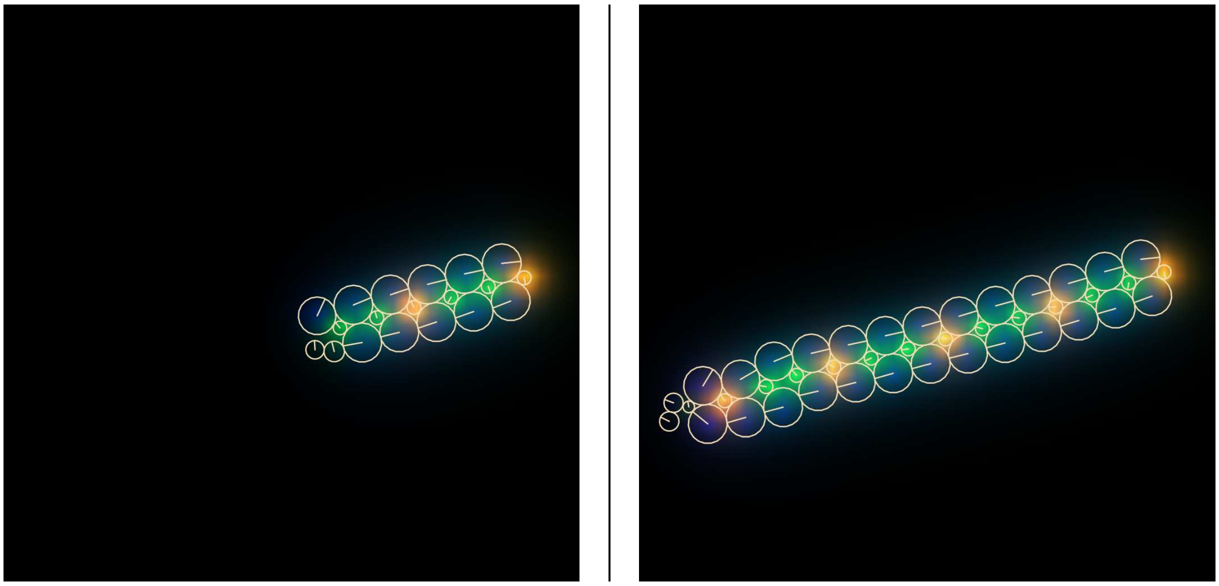


Figure 5.16: Segmented structure. □

5.2.10 Segmented Structure

This experiment builds on the 1-2-1 chains of Section 5.2.8 to create an example of segment formation. Once again, we have two cell types in the final structure, small cells (type A) and large cells (type B) which fit together tightly to form a fairly rigid chain.

Some A cells differentiate into a new type (A') and emit a diffusing chemical y (shown in yellow). This chemical inhibits nearby A cells from differentiating. The range of inhibition is limited, so as the chain grows, eventually one of the new A cells will differentiate as well. How many A cells are skipped before the next one differentiates depends on the diffusion coefficient for y and the sensitivity of the A cells to the threshold of y.

The final structure has A' cells located periodically along the axis of the chain. These cells can serve as a basis for the elaboration of a segmented organism.

5.2.11 Bending Chain (muscle-like contraction)

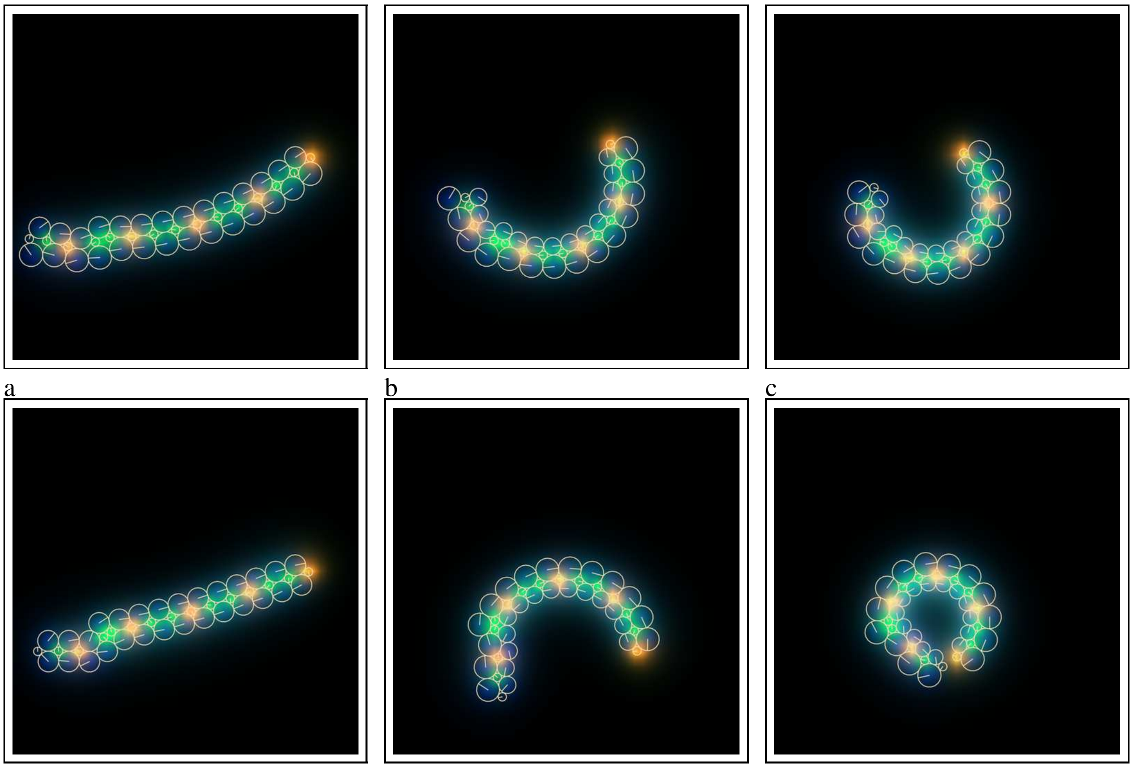

a b c
d e f

Figure 5.17: Curling up, first in one direction, then in the other. □

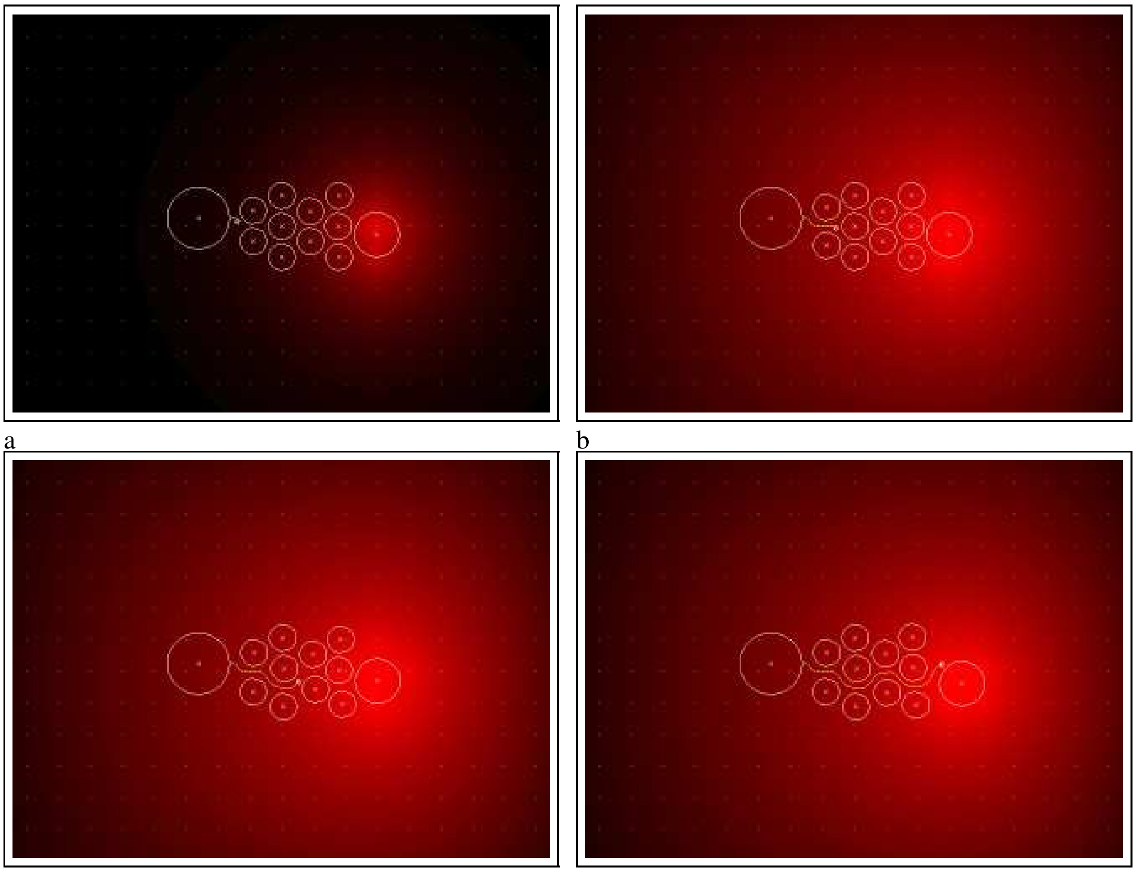
This experiment adds a global mechanical behavior to the segmented chain of the previous example. Once the chain has formed, the cells on the top side of the chain contract in unison while the cells on bottom side of the chain expand. This causes the chain to curl up (Figure 5.17 a-c). Panels d-f show the inverse behavior.

Thus the individual mechanical behaviors of the cells can effect a global shape change, forming a circle of cells from a linear chain. There are many ways to form a similar pattern using different combinations of mechanisms.

This experiment was inspired by the mechanical cell models of gasturlation and eversion shown in [Odell et al., 1981].

5.3 Simulation Experiments: Neural Net Development

5.3.1 Barrier Penetration (by Axons)



a b c d

Figure 5.18: A growth cone pushed through some barriers to reach its target. □

Neurite path finding. In this experiment, a growth cone from a cell on the left climbs the gradient of the chemical (shown in red), pushing through the barriers in its path until it reaches the far cell (which is emitting the chemical). Note the robustness exhibited here: the connection is made despite the presence of barriers. If the barriers were removed or differently located, the connection is still likely to be made.

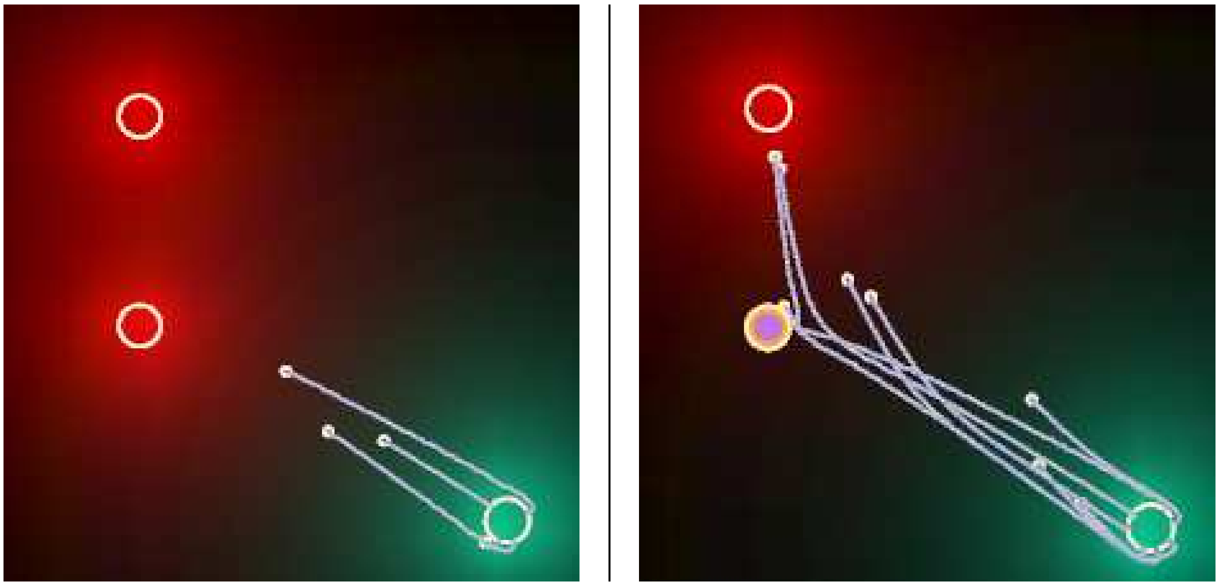


Figure 5.19: After the first growth cone arrives and adheres, future growth cones are not accepted by the target neuron. □

5.3.2 Target Neuron Changes State

This is a simple form of competitive innervation. When the first growth cone contacts a target neuron, it adheres. The target neuron then changes state and is not adhesive to future growth cones.

This is done with two pairs of complementary surface chemicals (a, a') and (b, b'). The first pair (a, a') is used to notice the arrival of the first growth cone. Initially, the target neuron expresses a but not b, and the growth cones all express a' but not b'.

When a first contact is made, the neuron begins to make b, and the growth cone begins to make b'. The neuron also halts production of a and of the diffusing chemical (shown in red). Future growth cones will not adhere to the neuron since it is no longer expressing a, and eventually they will not even be attracted to it because the remaining diffusible attractant will have disappeared as well.

Another component of this simulation is that growth cones die if they do not make contact within a certain time. Once all of the target neurons have been innervated, the remaining growth cones all die. This ‘pruning’ of excess neurites is a phenomenon observed in biological networks [Purves and Lichtman, 1985, Brown et al., 1991].

Note that this strategy addresses the problem raised in Section 4.5, since it ensures that all target neurons will be innervated (if the source neurons create enough growth cones).

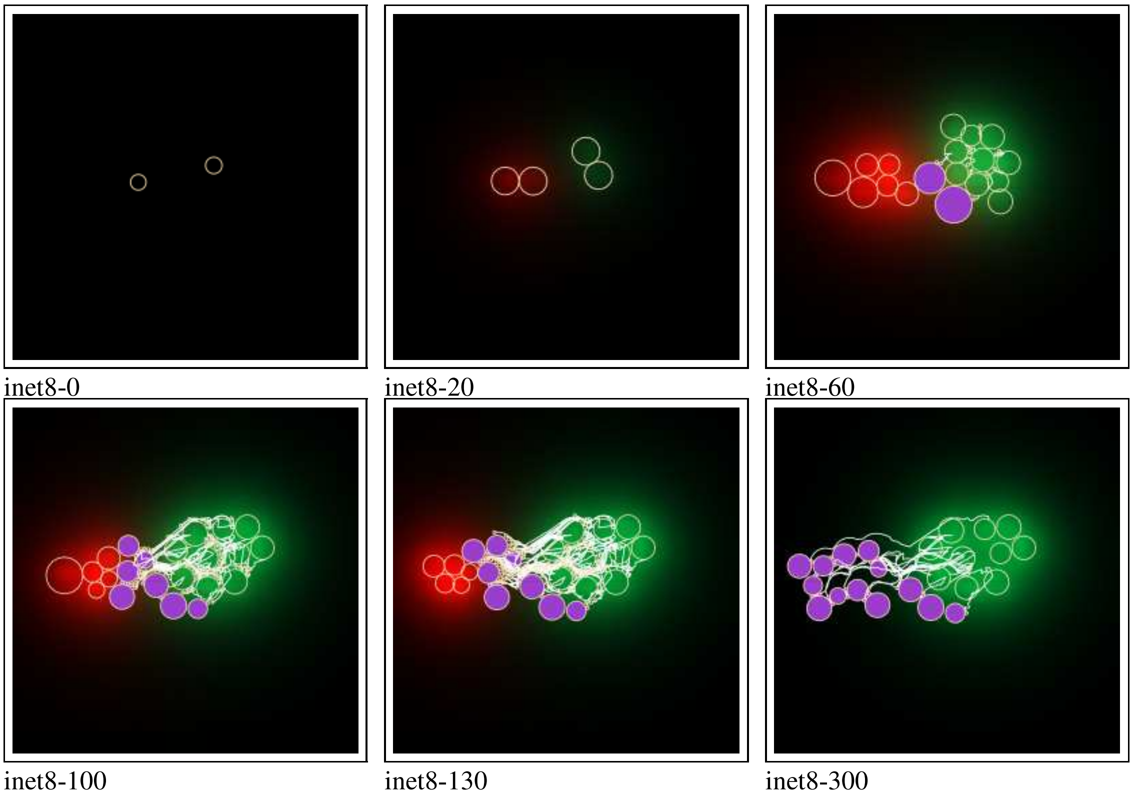


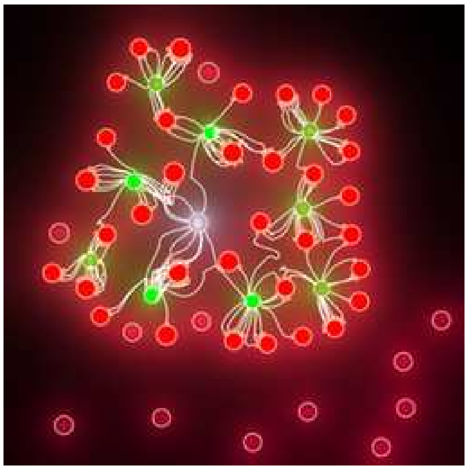
Figure 5.20: Identity function network. □

5.3.3 Identity Function

This simulation uses the mechanisms described on the previous page (Section 5.3.2) to create a one-to-many mapping. Target neurons can only receive one input, since they change state after receiving their first connection.

We begin with two cells (one of each type), which divide to create the remaining cells. Source cells stop dividing based on a cellular clock, and then emit neurites. Target cells keep dividing until they receive a connection. This creates a race condition, and if the neurites do not grow fast enough, the target cells will grow out of control.

Excess neurites die off if they cannot connect. This can be seen in the difference between the number of white neurites in the last two panels of Figure 5.20. This “pruning” of excess neurites is a phenomenon observed in biological networks [Purves and Lichtman, 1985, Brown et al., 1991].



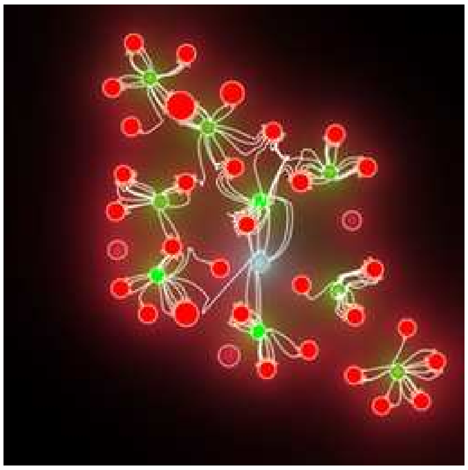


Figure 5.21: (Left) Two-level hierarchical network. The cell in the center (emitting the chemical shown in white) is connected to the cells emitting chemicals shown in green. They, in turn, are locally connected to the nearby cells (emitting chemical shown in red). □

Figure 5.22: (Right) Another hierarchical network, starting from different initial conditions. □

5.3.4 Hierarchical Connectivity (retina-like)

In this model, neurons at each level in the hierarchy emit a different diffusing chemical. This enables the neurites from lower-level neurons to find them easily and without confounding them with neurons from other levels.

Scaling properties. Adding another hierarchical layer of neurons is not trivial using the cell state equations of this experiment. The difficulty is similar to that described in the previous hierarchical example (Section 5.2.5). The new neurons would have to emit a new diffusing chemical, **and** the growth cones from the neurons at the next level up in the hierarchy would have to be modified to be able to find it. Since both of these changes are necessary, and they require the use of a new diffusing chemical, it is a difficult solution for an evolutionary algorithm to find.

An alternative. An alternative model could have a single diffusing chemical for all levels of neurons in the hierarchy. The timing of the release of the chemical could be used to determine the level in the hierarchy. Imagine that each neuron has an internal trigger time at which it starts to release the chemical. Then after a fixed delay, it stops emitting the chemical and starts to grow neurites. Several versions of these neurons with different trigger times will then connect together in a hierarchy. In this 'timed-release' model, adding a new layer is easily done with a single genetic change, by adding a new neural type with a longer trigger time.

The different evolutionary implications of these two strategies underscore the importance of development in the process of evolution [Goodwin, 1994, Gell-mann, 1990].

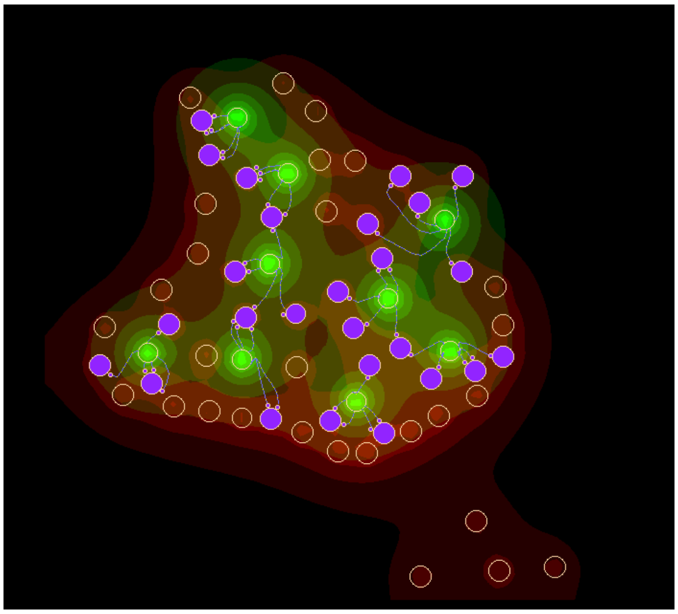


Figure 5.23: A locally connected network, using the same cell state equations as Figures 5.21 and 5.22, but with only one level of hierarchy. □

5.4 Simulation Experiments: Three-Dimensional Development

The three-dimensional extensions were recently made to the simulator. Real morphogenesis occurs in three dimensions, and we anticipate that the ability of the simulator to model this will be useful for examining many aspects of morphogenesis. Very few other developmental simulators currently operate in 3D, and certainly not with the range of mechanisms provided here. (Note: diffusion is currently not supported in our 3D system.)

The following simple experiments give some idea of how the 3D capabilities extend the range of structures that can be generated.

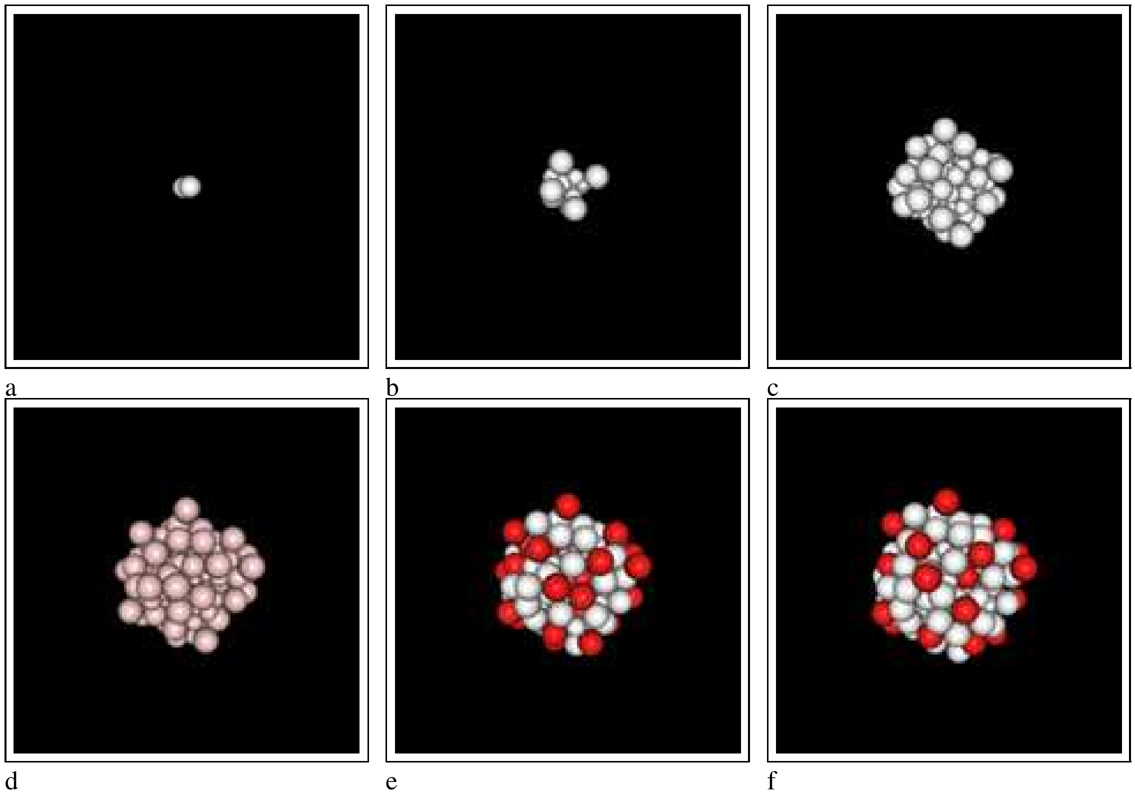


Figure 5.24: A clump of cells grows to a certain size (controlled by an intracellular clock), and then some of the cells differentiate (shown in red). The differentiated cells inhibit their neighbors, preventing them from differentiating. □

5.4.1 Blob of Cells with Contact Inhibition

This clump of cells grows to a certain size (controlled by an intracellular clock), and then the cells differentiate. Lateral inhibition via a surface chemical prevents neighbors of a red cell from also becoming red. This experiment uses the same cell-cell interactions as described in the two dimensional experiment in Sections 5.2.4 and 6.2.

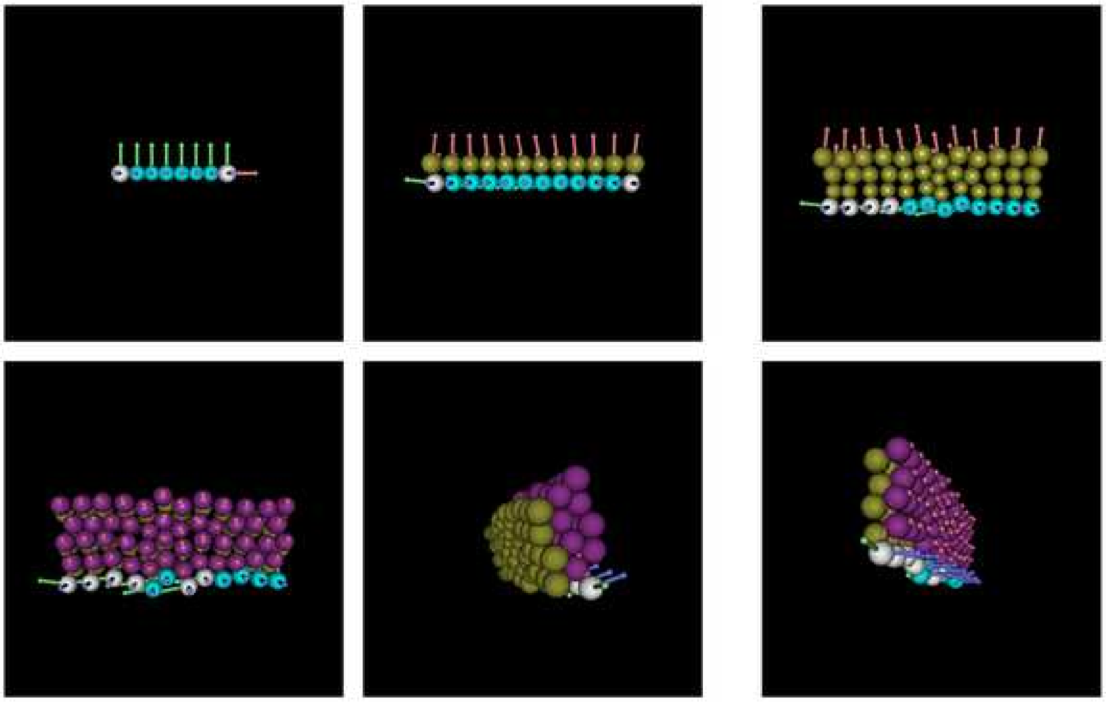


Figure 5.25: Cell layers (in 3D). Cells of different types are displayed in different colors. The local coordinate frame of each cell is shown with three small arrows (x: red arrow, y: green arrow, z: blue arrow). Side views: panels (e) and (f). □

5.4.2 Layers of Cells

The formation of cell layers is critical in most multicellular organisms. In this simple experiment, we show that our system has the capability of creating layers of cells.

The structure is formed in three stages. From a single cell, a line of cells is created by repeated dividing with a vertical cleavage plane. Then each cell in the line divides horizontally a few times to create a sheet of cells. Each cell in the sheet then divides in the third plane to create multiple layers (shown from the side in panels (e) and (f)).

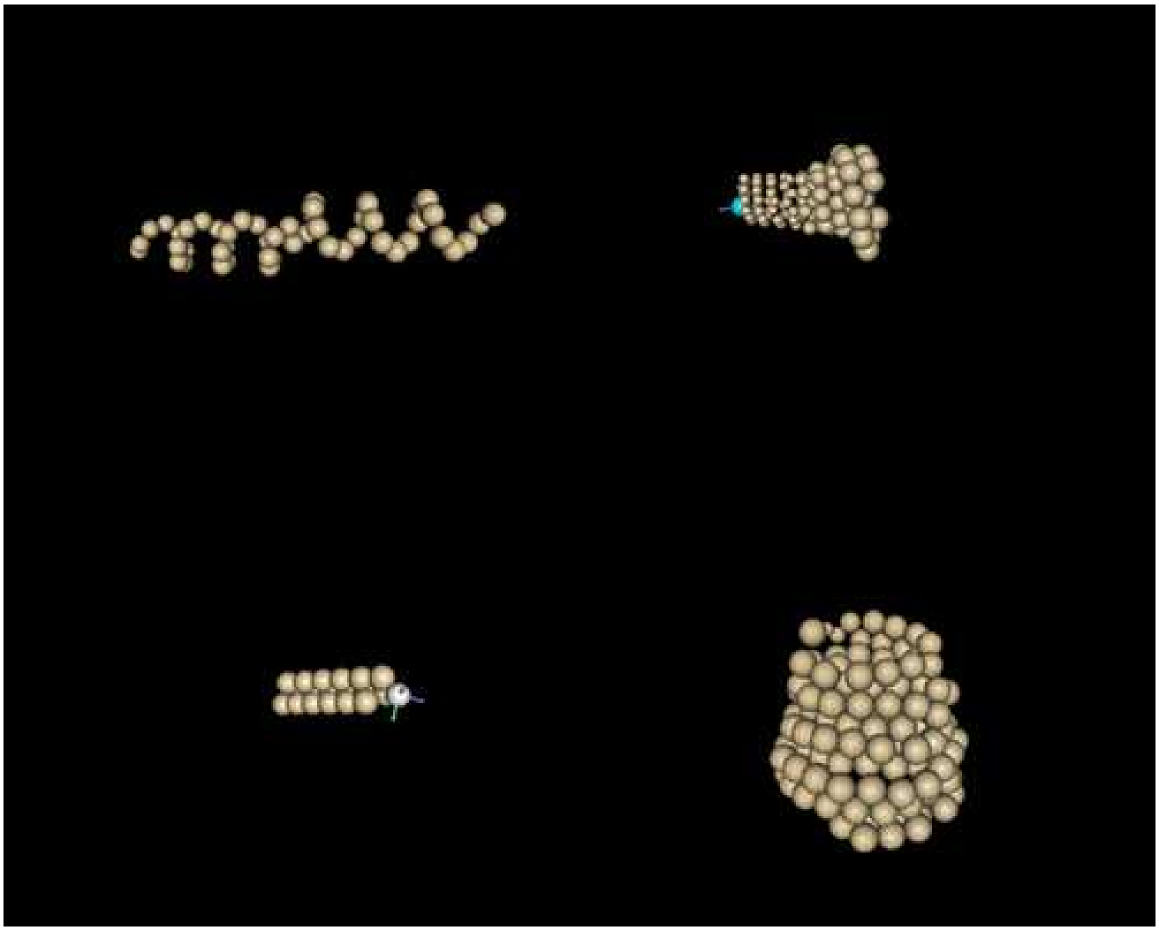


Figure 5.26: Spiral growth with various parameters. □

5.4.3 Spiral Growth

A single progenitor cell was used to create three of these shapes (two cells moving in opposite directions were used for the upper left figure).

The progenitor cell rotates its cleavage axis gradually as it divides. This simple rule, together with the geometric collisions, gives rise to the shapes in these figures. Different settings of cell size, rotation rate and tilt account for the variations in the four simulations.

PART II

APPLICATIONS

Applications

- Ch. 6. Synthetic Biology
- Ch. 7. Evolving Neural Networks
- Ch. 8. Computer Graphics: Cellular Texture Generation

Chapter 6

Synthetic Biology

In the preface to his book, Murray has a nice description of the criteria for evaluating models in mathematical biology:

> "Mathematical biology research, to be useful and interesting, must be relevant *biologically*. The best models show how a process works and then predict what may follow. If these are not already obvious to the biologists *and* the predictions turn out to be right, then you will have the biologist's attention." [Murray, 1993, page v-vii]

The examples of biological modeling in Murray's book are indicative of the style of mathematical biology in general use today. A mathematician provides a theoretical explanation of a biological phenomenon, usually reducing the system to a small number of parameters and identifying regimes of behavior via mathematical analysis.

A different approach is examined in this chapter. Instead of trying to simplify the system sufficiently to enable thorough mathematical analysis, we try to include as many of the relevant phenomena as is computationally feasible. The resulting simulation system has many of the features of the real system being modeled. [1]

We can then use the simulation to gain *intuitions* about the biological system. This is the proposed *synthetic approach to biology*. [2] The user specifies the properties of a cell and its interactions with other cells and the environment, thus creating a developmental system from scratch. The difference between building a system (engineering) and dismantling or dissecting a system (biology) is substantial (Figure 6.1). Each approach brings with it a different point of view, and is likely to discover different things. By enabling biologists to take this engineering approach of constructing organisms, we hope to add a new tool to their repertoire.

Although this approach has its roots in theoretical biology that has been ongoing for many years [Turing, 1952, Waddington, 1968, Wilson, 1970, Harrison, 1993], the ability to bring substantial computer power to bear on the problem is more recent. We believe that this approach will yield a useful tool for biologists sometime in the near future. The key to its success will be if it can satisfy

[1] It will be harder to analyze conclusively. This is a tradeoff.

[2] This approach builds on work in Mathematical Biology and Artificial Life. We differ from these in our emphasis on the use of the simulator to help generate *intuitions* about biological systems, which can later be examined more closely and subjected to analysis.

analytic: take it apart vs. synthetic: put it together

Figure 6.1: A coarse characterization of analysis versus synthesis approaches. □

the criteria indicated above by Murray: biological relevance, and making non-obvious predictions. Features of the approach include:

- an alternative way of thinking about biological models,
- enables experiments which would be difficult or impossible in vivo or in vitro (a method of doing gedanken experiments),
- very controlled experiments can be done,
- study parameter variations,
- study combinations of different mechanisms, and different timings,
- biologists could use the system directly (with appropriate user interface),
- encourages users to consider how initial conditions might arise, or how a smaller model might fit into a larger context (e.g., when using reaction-diffusion models for chemical prepatterning, how do initial conditions arise? What is the temporal sequence?).

A system like this could be used directly by biologists, aiding them to try and test their own models. This is likely to be effective since the simulation system has graphical feedback which shows the results in a manner which biologists are accustomed to.

We predict that difficulties with this approach will mostly be associated with the choice of models. As discussed in Chapter 2, a particular model is valid only in a particular domain. Since a computer system does not know the intent of the user, it cannot determine if the model is being applied appropriately. Thus it is critical that the user understand the model and its range of applicability.

In the exploratory experiments we have attempted, some of the most interesting intuitions gained are actually questions that are raised about the validity of different models in different domains. For example, when investigating patterns of inhibition at a cellular level, reaction-diffusion systems which take the abstraction of a continuous sheet of cells are inappropriate, since they cannot faithfully represent phenomena happening at the scale of a single cell diameter. Thus it is likely that the synthetic biology approach will need to depend on a variety of models. For models where a clear correspondence can be made, comparisons between the models can also be an aid in understanding the phenomena.

Archival model representation (another application area). Another use of a model like this is as a tool for communicating and storing biological knowledge. As our understanding grows, we create more and more complex models of biological phenomena. For example, the current state of knowledge of the developmental process in *Drosophila* retina is kept in the minds of researchers, and, indirectly, in many journal papers, etc. With systems like the retina, hundreds of genes have already been found to be involved, and many of the interactions are known or postulated. It is difficult to keep track of all of the hypotheses and to detect inconsistencies. A suitably flexible simulation framework could serve as a bookkeeping mechanism for the research of such phenomena. [3] This would also serve as a valuable educational tool.

6.1 Synthetic Biology: Using Simulation to Generate Intuitions

The following two examples show how simulation can be used to generate intuitions about biological pattern formation. The first example looks at the possibilities of long-range activity via cumulative interactions of surface chemicals between neighboring cells. The second example explores cell sorting via differential adhesion.

In each case, the work of setting up and running the initial experiment was a few hours, and the entire suite of simulations was done part-time over the course of a few days (with individual simulation runs taking a few minutes for the first example, and a couple of hours for the second).

One important result in the area of Synthetic Biology is that a quick exploration of a phenomenon via simulation can generate useful intuitions. These intuitions may then lead to further explorations via experiment or mathematical analysis.

Although the intuitions gained from the particular examples given here are simple, they show the promise of the approach. It is hoped that future work in this area will generate biological insights of sufficient import to validate the approach.

6.2 Example: Long-range Inhibition via Contact

As a result of conversations with Ajay Chitnis of the Salk Institute, we considered the possibility of large scale patterning due to a membrane ligand which was known to inhibit its receptor. [4] Thus the signal can only be transmitted between cells which are in direct contact. Although cell contact is inherently local, can the inhibition be passed on from cell to cell to create a long-range inhibition? What properties would such a system have? These are the questions we set out to examine.

Cell state equations for this experiment. There are two important mechanisms operating in this experiment:

1. a genetic switch which turns on if the inhibition is low, indicating the cell has differentiated, and

[3] On a related note, we have recently learned that Jim Bower (Caltech) has started a database of compartmental models for pyramidal cells.

[4] A *ligand* is a molecule that binds to another molecule. In this case, we are considering a membrane ligand (one which lies in the membrane of a cell). The ligand and receptor are a pair of complementary molecules. If they are expressed on the membranes of adjacent cells, the ligand and receptor bind together.

2. an inhibition which is passed on from cell to cell via interaction of surface chemicals (*not* diffusion).

In addition, there are some equations which determine the rate of cell division, etc., but they are not germane to the present discussion. The complete experiment file for this experiment appears in Appendix A, but we will cover the salient points in the following paragraphs.

Let z_d be a state variable which causes a cell to differentiate into another type (shown in dark gray) if it reaches a threshold value. Assume also that we have three surface chemicals r (receptor), l (ligand) and n (normalizer). The receptor-ligand pair bind heterophilically, and the normalizer binds homophilically (like binds to like). Recall that the cell will receive as inputs $\mathtt{env}[r, l, n]$ which report the amount of each surface chemical which is bound on its membrane. We'll use shorthand notation $e_a = \mathtt{env}[a]$ for brevity here.

The normalizer n is used to compute e_r/e_n, the amount of receptor bound *per unit area of contact* (see Section 9.4.3 for details on why this works). This value is independent of the number of cells in contact, and reports an approximation to the average concentration of ligand expressed by its neighbors.

The equations for the genetic switch and the inhibitory surface chemicals are given here (and explained afterwards):

$$
\begin{aligned}
\tfrac{dz_d}{dt} &\mathrel{+}= -20\tfrac{e_r}{e_n} + 10\tfrac{z_d^2}{1+z_d^2} - 0.5z_d + 13\\
\tfrac{dz_l}{dt} &\mathrel{+}= 0.95\tfrac{e_r}{e_n}\\
\tfrac{dz_l}{dt} &\mathrel{+}= (z_d \mathrel{\tilde{>}} 3)\, z_d\\
\tfrac{dz_l}{dt} &\mathrel{+}= -z_l\\
\tfrac{dz_r}{dt} &\mathrel{+}= 5 - z_r\\
\tfrac{dz_n}{dt} &\mathrel{+}= 1 - z_n
\end{aligned}
\tag{6.1}
$$

Equation 6.1: Cell state equations for experiment with inhibition via surface chemicals. □

r is a receptor for the ligand l. n is used as a normalizer. The surface chemical r is expressed equally on all cells (they all have the same amount of receptors). The normalizer is also expressed equally on all cells.

To understand the ligand equations (z_l), consider first what happens for a cell which has z_d low (i.e., it is 'off' = not activated). Then the combined equation for l is

$$\frac{dz_l}{dt} = 0.95\frac{e_r}{e_n} - z_l.$$

As this differential equation reaches its setpoint, l is expressed at 95% of what a cell senses via the receptors. This means that it inhibits its neighbors at 95% of the inhibition it gets from them. Thus the inhibitory influence is passed along between non-activated cells. This can be seen in Figure 6.2. The inhibition diminishes for cells further from the activated (dark gray) cell.

For a cell which has z_d high (i.e., it is 'on' = it has differentiated), an extra term is added into the equation for z_l, making it

$$\frac{dz_l}{dt} = 0.95\frac{e_r}{e_n} - z_l + z_d.$$

The output inhibition then approaches 95% of what the activated cell sees, *plus* more, proportional to how 'on' this cell is. Thus a differentiated cell inhibits its neighbors strongly, and they pass on this inhibition to their neighbors.

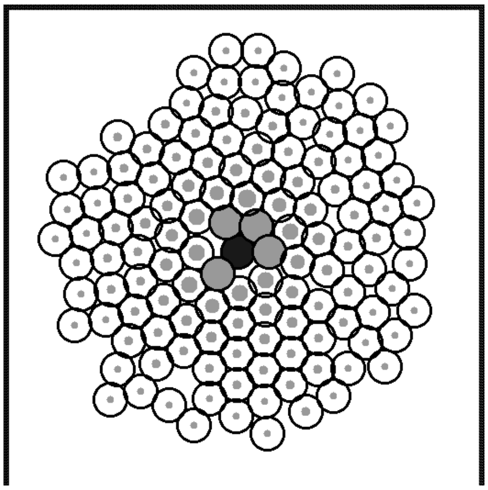


Figure 6.2: Propagation of inhibitory signal via surface chemicals. (Note: size of displayed circle for inhibitory signal is scaled up relative to the Figures 6.3 and 6.4.) □

The remaining equation is for the genetic switch, which implements the differentiation process by making z_d large if the inhibition $\frac{e_r}{e_n}$ is small.

$$\frac{dz_d}{dt} = -20 * \frac{e_r}{e_n} + 10 * (z_d^2)/(1 + z_d^2) - 0.5 * z_d + 13$$

This equation is similar to those discussed in [Murray, 1993, Chapter 5].

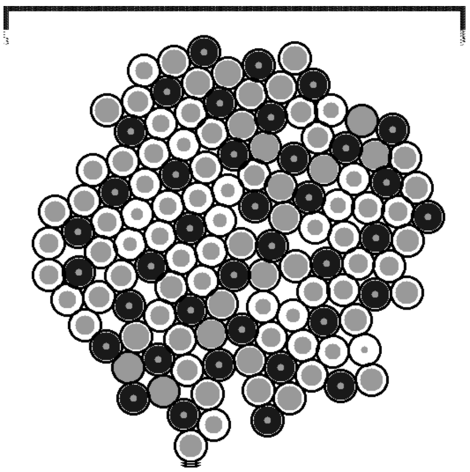
These equations have both elements of the standard reaction-diffusion equations: autocatalysis (a reaction which tends to grow) and lateral inhibition (inhibit the neighbors), but differ in that they are computed in discrete cells, and without the use of continuous diffusion (Section 6.2.1).

Many Stable States are Possible; Temporal Sequence is Important. There are many stable states for this system, and different regimes of stable states as well. This became obvious quite quickly when beginning to work with this experiment. Figures 6.3 and 6.4 show two states with very different densities of activated cells (shown in dark gray).

The different regimes are due to the discrete nature of the lateral inhibition in this experiment. *It can only be passed on from cell to cell if the cells are in direct contact.* A non-activated cell cannot express enough inhibitor to de-activate a neighbor which has already differentiated. If cells ever achieve the state in Figure 6.5(ii), where every other cell is 'on', they will remain in that state. This generates patterns with short-range inhibition.

But what about long-range inhibition? Can it also be accomplished with this mechanism? Yes, as shown in Figure 6.4, but it depends on the temporal order in which cells differentiate. Cells which differentiate early can inhibit their neighbors for a large distance, given enough time. That is the issue – is there time for the inhibitor to propagate to the other cells before they differentiate? This is similar to discussing initial conditions. [5]

[5] **A discussion of the meaning of initial conditions in developmental systems.** In a developing organism, one must be careful when discussing 'initial conditions'. It is probably not the case that a sheet of cells is held at some set of initial values and then suddenly, synchronously, they all begin to differentiate. It is more plausible to consider that the



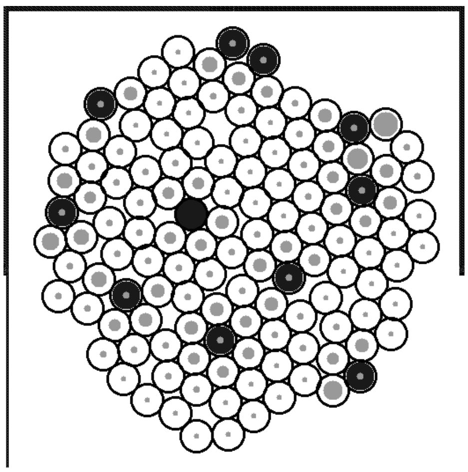


Figure 6.3: (Left) Short-range inhibition via surface chemicals. □

Figure 6.4: (Right) Long-range inhibition via surface chemicals. □

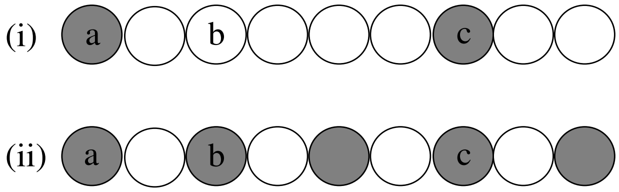


Figure 6.5: Cells shown in dark gray have differentiated, and inhibit their neighbors from differentiating.
There are many different stable states for this system, two of which are shown in (i) and (ii).
Which state is reached depends on initial conditions.
One way to ensure state (i) (long-range inhibition) is to have another chemical signal which enables the cells to activate. Let this signal propagate in time, first enabling cells on the left and later the cells to the right. If the rate at which the enabler is progressing is slower than the rate of the inhibition, then the state shown in (i) can be reached reliably.
Thus there are possible temporal effects which are intimately related with the ability of surface chemicals to exhibit long-range inhibition. □

If we consider a sheet of cells as it grows, there is a natural temporal ordering. The cells which exist first have first crack at differentiating. As new cells are formed at the edge of the sheet, they are inhibited from differentiating by the previous cells. New cells will only be able to differentiate when the edge of the sheet has progressed far enough from the previously differentiated cells. Thus a developing sheet can exhibit long-range inhibition using the surface inhibition mechanism in conjunction with growth at the edge of a sheet.

6.2.1 Limitations of Continuous Reaction-Diffusion Models

These experiments with lateral inhibition using membrane-bound chemicals have uncovered some behaviors that are difficult to represent using continuous reaction-diffusion models.

Continuous reaction-diffusion models are used to describe a wide range of biological phenom-

---

cells are gradually reaching the stage when they are ready to differentiate, and either (a) they begin independently but in near-synchrony or (b) they begin sequentially in some order determined by the time of their last mitosis, or (c) they are enabled for differentiation by another signal, as in the morphogenetic furrow sweeping across the *drosphila* retina.

Different Interpretations of Reaction-Diffusion Equations

1. Chemicals diffuse and react in a continuous medium.
2. Chemicals diffuse in a continuous medium. Reactions occur only within cells which can sense/emit the extracellular chemicals ([Murray, 1993, Chapter 15.2] and Section 5.2.3).
3. Chemicals diffuse only by direct exchange from one cell to its neighbor (e.g., via a gap junction), and reactions occur only within cells ([Turing, 1952, Section 6]).
4. Chemicals are membrane-bound. Each cell can sense its neighbor's membrane-bound chemicals. Reactions occur within cells to change the concentrations of chemicals in their membranes (Section 6.2). Actual diffusion does not occur, but a similar process is achieved as each cell attempts to express a membrane chemical at the level of the the average of its neighbors.

Figure 6.6: These four real-world situations can be modeled using reaction-diffusion equations.
Creating continuum models for situations 2, 3, and 4 is sometimes done by considering a dense sheet of cells in the limit as the cell size goes to zero.
This continuum model cannot represent phenomena at the level of individual cells and may miss out on some important phenomena. □

ena. One use is to represent interactions between many individual, discrete cells as a continuum. However, this continuum model has difficulty describing behavior at the resolution of single cells, due to the assumptions made when creating the continuum model (e.g., cell size goes to zero, or chemicals diffuse freely through the membrane; see Figure 6.6).

There are a variety of related reaction-diffusion models, which we believe may have different characteristics at small scales. These different small-scale behaviors can lead to different large scale behaviors, as in the different densities of dark gray cells shown in Figures 6.3 and 6.4.

6.3 Summary vis-a-vis the Synthetic Biology Approach

What did we learn from this experiment? We learned that temporal ordering is important for long-range inhibition using surface chemicals. We also gained some intuitions about characteristics of reaction-diffusion systems at the limit of their resolution (when a single cell turns on or off), which cannot be conveniently represented by continuous models. This is discussed further in Section 6.2.1.

Could we have learned it more easily some other way? Individual people approach problem solving differently. Some amount of intuition comes from thinking carefully about a problem, doing back-of-the-envelope calculations, etc. For the natural sciences, experimental work is clearly the major source of experience and hence intuition. This rough simulation experiment is another means of gaining experience with a system, not as real as the real system, but not as abstract as the

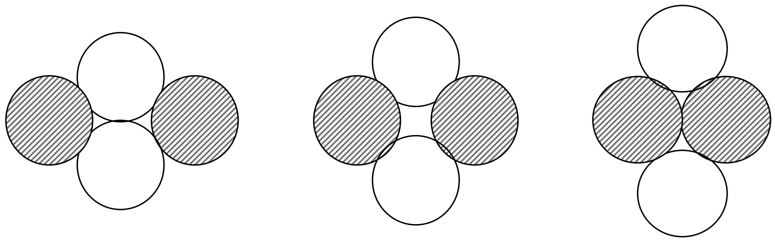


Figure 6.7: **A configuration of four cells which cannot reach the minimum surface-energy configuration without going over an energy 'hill'.** Assume that the dark-dark cell adhesion is stronger than the light-light cell adhesion or the dark-light adhesion. The four cells in (a) cannot reach the minimum surface energy configuration in (c) without either deforming substantially, or passing through state (b), which requires work. Note: an analogous situation can be constructed in 3d. □

back-of-the-envelope calculations. I found it to be surprisingly instructive to use for this problem and others.

Did it make non-obvious predictions? When I began this series of experiments, I did not consider that the temporal order would be so critical. In retrospect it seems reasonable, but the point is that in a short exploratory session with the simulator, this property was immediately evident. It did not require a thorough mathematical analysis, and it did inspire further thought on what might be happening in the biological system to provide this temporal ordering.

6.4 Relationship Between Cell Shape and the Differential Adhesion Hypothesis

In this suite of simulated experiments, we quickly discovered some implications of cell shape for Steinberg's *differential adhesion hypothesis* [Steinberg, 1964, Steinberg, 1970, Armstrong, 1989]. The differential adhesion hypothesis is a model to describe cell sorting (the separation of different types of cells when mixed either in vitro [Gilbert, 1991, Armstrong, 1989, Steinberg and Takeichi, 1994] or in vivo [Gilbert, 1991, Crawford and Stocum, 1988]). It states that "in any population of motile, cohesive cells, weaker cell attachments will tend to be displaced by stronger ones, and this adhesion-maximization process ...[produces a]... configuration in which the total intensity of cell bonding is maximized." [Steinberg and Takeichi, 1994] It is also assumed that the motility is "non-directed." [Steinberg and Takeichi, 1994]

The outcome of this process is that a mixture of two types of cells can rearrange itself into a particular configuration due solely to the adhesive properites of the two cell types.

Our simulated experiments suggest that the cell sorting behaviors depend on additional conditions on cell shape and/or the magnitudes of the motile forces relative to the adhesive forces.

The essential result is shown in Fig. 6.7. [6] If the cells are able to deform sufficiently to remain in contact throughout the transformation, then they can achieve the state shown in Fig. 6.7(c). This is compatible with the models for cell movement mentioned by Armstrong:

[6] This is sometimes called a T1 transition [Glazier and Graner, 1993, Figure 1].

> "Two mechanisms of cell movement have been considered for sorting: active pseudopod-generated locomotion and associative movement. The latter process envisions cell movement to be the consequence of a "zippering up" of homotypic contacts and a resultant reduction in the area of contact between cells. The process does not require the activities of pseudopods as being responsible for cell locomotion." [Armstrong, 1989]

However, if the cells cannot deform in this manner, then the light cells in Fig. 6.7(a) must lose contact due to their "non-directed" motility. This is necessary to escape a local peak in the energy landscape. But if cells have this ability to break an adhesive bond with a neighbor, then some cells will (with some probability) eventually break all bonds and wander off alone. This can be observed in several of the simulation experiments below.

In summary, we find one of the following conditions must hold for the differential adhesion hypothesis to hold:

1. cells must be capable of extensive shape deformation, or
2. a cell's motility must be strong enough to break its bonds with neighboring cells.

Although some real cells are nearly rigid (and spherical) like those of our model, the types of cells which exhibit differential adhesion generally are not. They are also capable of substantial amounts of deformation. These differences are the cause of the behaviors we found. Thus the application of the spherical cell model does not give realistic results for phenomena such as these.

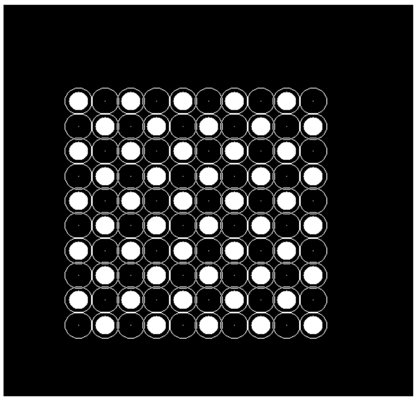
As we mentioned in the introduction to this chapter, we believe that the weakest point of the synthetic biology approach is the possibility of inappropriately applying models to phenomena which they cannot faithfully represent. Other models, such as the lattice model proposed by Glazier and Graner are able to represent this transition more accurately, and hence they achieve much nicer results when simulating differential adhesion [Glazier and Graner, 1993]. We consider the possibilities of extending our model to use lattice methods for shape representation in Section 9.3.1.

Nonetheless, the results obtained are still true – cell shape and deformation are important components of the differential adhesion theory. Thus the synthetic biology approach was again useful for generating further understanding of the problem.

Differential Adhesion Simulation Experiments

The following figures illustrate the experiments that helped us to develop the intuitions described above. The initial motivation was to show differential adhesion as another capability of the simulator. After encountering some unexpected behaviors, we were led to consider the discrepancies between our model, Steinberg's model, and the actual biological phenomena.

In the current implementation of our model, cells are spheres, and are not capable of the large deformations described above. It is straightforward to extend our model to a Voronoi model (computing cell boundaries using a Voronoi or Dirichlet algorithm [Honda, 1978, Honda, 1983]), and in fact should not require any new implementation other than display routines. The Voronoi-style algorithm is more readily able to replicate differential adhesion behavior, but is still less accurate than the lattice models of [Glazier and Graner, 1993], since it computes the angles between contacting membranes using purely geometric criteria rather than a force-based algorithm.



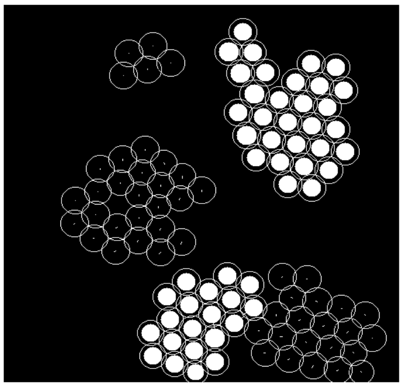


Figure 6.8: (Left) Differential adhesion – starting state. □

Figure 6.9: (Right) Differential adhesion – later state. Cells of each type do not adhere to the other. □

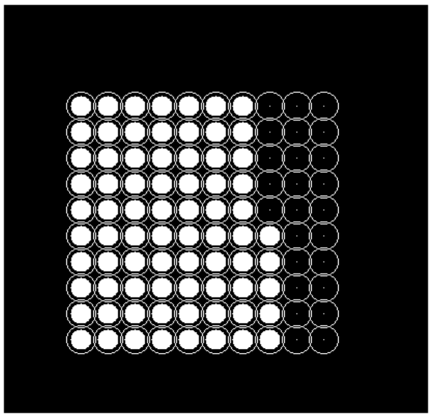
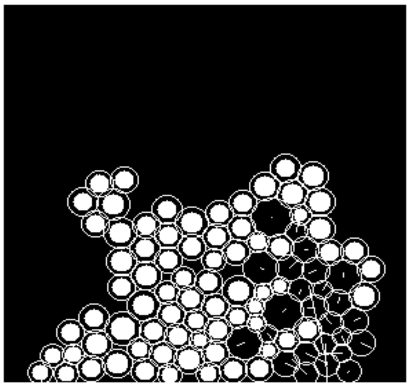


Figure 6.10: Differential adhesion – starting state for remaining experiments. □

Each simulation run for the differential adhesion takes on the order of two hours. This long run-time is partially due to stiffness arising from the collision equations. Another factor is that the final patterns are only achieved after a substantial amount of random motion of the cells. The interaction between the collision forces and the random motion causes small solver steps.



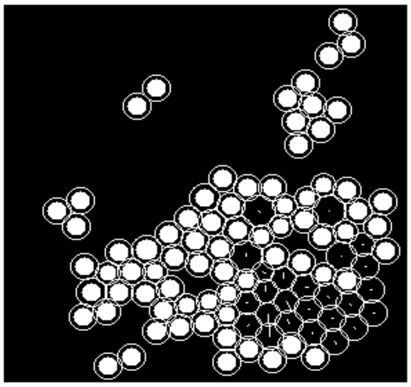
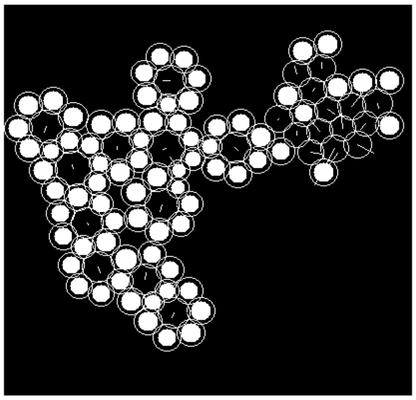


Figure 6.11: (Left) Slight adhesion for a-a, more for a-b, lots for b-b. □

Figure 6.12: (Right) Same as Figure 6.11, but with cell size less affected by compression/expansion forces. See Section 9.3.1 for a discussion of these forces. □



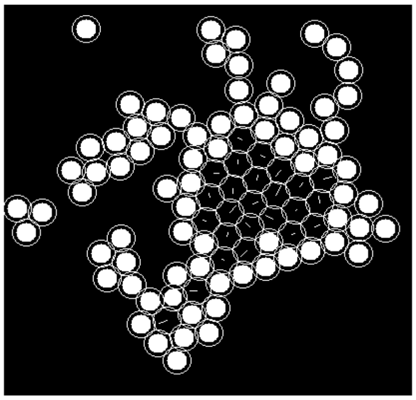


Figure 6.13: (Left) Slight adhesion for a-a, more for a-b, lots for b-b, run for longer time. □

Figure 6.14: (Right) Same as Figure 6.13, but with cell size even less affected by compression/expansion forces. See Section 9.3.1 for a discussion of these forces. □

Chapter 7

Artificial Evolution of Neural Networks

The goal of this project is to create artificial neural networks with *geometric* structure that is specific to the problem being solved. We wish to create networks which have arbitrary, asymmetric geometric structures as well as heterogeneous neural types. Our motivation for this is the observation that geometry is important in some real neural circuits.

Optimization over arbitrary geometric structures is an open problem. Our approach is to use artificial evolution (genetic algorithms) [1] to search the space of possible network structures. This approach has some philosophical differences from traditional work in artificial neural networks (ANNs). We loosely characterize the traditional approach of ANNs as: 'understand natural neural circuits, abstract the fundamental principles, and apply them to machine computation'. The proposed alternative is to explore the *process* by which neural circuits and brains arise in nature, viz., the *evolution of developmental systems*.

In the past few years, there has been substantial interest in 'Evolutionary Artificial Neural Networks' [Yao, 1993]. The approaches which seem most likely to us to be successful are those based on adaptive developmental models, such as ours. Among these, we include the most biological detail. As with the other similar approaches, we have yet to evolve networks to solve specific problems, but we have shown the model to be capable of representing heterogeneous, asymmetric networks with geometric structure, and we have further shown that the representation can be optimized with respect to an objective function using a variant of genetic programming [Koza, 1992].

7.1 Motivation

There are two motivations for this project. The first is the desire to create computing machines which are capable of handling real-world sensorimotor and pattern recognition problems. We are particularly interested in problems which involve a behavioral loop: recognition, decision, and action. Organisms deal with these problems well, even in the presence of noise and 'messy' data.

[1] We use the term 'genetic algorithms' in its broadest sense, interchangeably with 'artificial evolution' or 'evolutionary computation'. Some authors use 'genetic algorithms' to refer to a particular style of evolutionary computation which does recombination on strings of symbols.

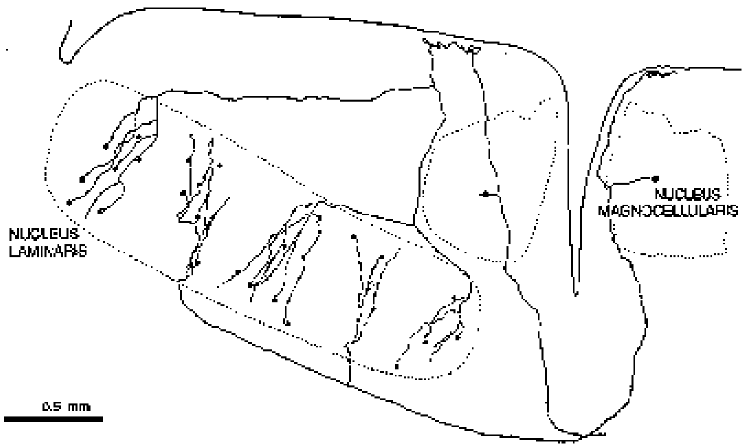


Figure 7.1: An example of a biological neural network, part of the owl's auditory localization circuit. The two axons coming from the different ears are used as delay lines. Thus, the length and position of the neurons is an essential part of the computation. This type of problem-specific geometric structure cannot easily be represented by other approaches (figure reprinted from [Carr and Konishi, 1988]). □

Another nice feature of these problem specifications is that there are external, measurable criteria for success.

The second motivation is the desire to understand the processes by which neural structures arise in nature. We emphasize the study of the process (the evolution of developmental systems), rather than the study of particular networks.

Geometric structure and heterogeneous neural types are important. Geometric network structure is known to be important in biological computation (beyond network topology). The length of a neurite is related to the time it will take for a signal to traverse it. Also, the geometric layout of sensors can be a component of the computation, since it constrains the shape of the input space (e.g., retina).

An example of the use of geometry in neural computation is the owl's auditory localization circuit [Carr and Konishi, 1988, Konishi, 1993], in which the axons coming from the two ears are used as delay lines. Coincidence detectors innervated by axons carrying signals from left and right ears indicate the simultaneous arrival of a signal. The physical location of this coincident arrival encodes the position in the real world of the sound source. Thus, the position and geometry of neurons and their connections is an essential part of the computation.

This type of problem-specific geometric structure cannot easily be represented by other artificial network approaches. Detailed geometric models are used for Computational Neurobiology (e.g. compartmental models [Koch and Segev, 1989]), but they have not been applied in engineering (to create networks to solve particular problems). Recent work on Time-Delay Neural Networks recognizes the importance of the delays that arise from geometric structure, but they abstract away the actual geometry. This gets some of the behavior we are seeking, but lacks other geometric capabilities such as incorporating sensor layouts into the choice of network architecture.

Some of the geometric structure of networks arises from the varied morphologies of neurons in real neural circuits. Another feature of our approach is that it enables the use of heterogeneous

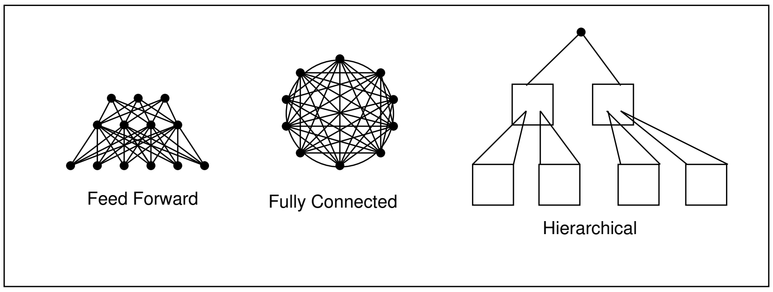


Figure 7.2: Most artificial neural nets specify topological connectivity, but not geometry. □

neural types and the association of morphological differences with those types. This should expand the computational abilities of our networks, and strengthen the analogies with biological networks, which have heterogeneous neural and synaptic types.

In contrast, artificial neural networks are typified by a pre-specified static topological structure, with no geometric information (Figure 7.2), and homogeneous neural types.

Study the process (evolution) rather than the outcome (brain). Artificial neural network (ANN) research attempts to understand neural architectures in an engineering sense, so the techniques can be applied to arbitrary problems. In Carver Mead's group, researchers build analog VLSI chips which emulate a specific biological circuit, using a structure similar to that observed in nature [Mead, 1989, Mead and Mahowald, 1988, Lyon and Mead, 1988].

These approaches aid our understanding of neural circuitry and are likely to continue to do so for some time. However, there is some small possibility that they will not scale well to understanding the function of larger neural circuits, or to the brain as a whole. The fine-scale structure of the brain may not be modular or organized in a manner which is easily understood. On one hand it is likely that a process like the 'developmental gain' mentioned earlier would lead evolved structures to be modular (as they generally appear to be). On the other hand, it is easy to imagine an evolutionary selective advantage to using a neuron for multiple functions, in a way which would be considered a 'hack' in software. What if the brain contains a substantial proportion of 'hacks'? If so, then we might be better off understanding the process by which it was created, rather than the structure as it exists.

This suggests an alternative to understanding the brain: we can try to understand the *process* by which brains are made (evolution of developmental systems).

Of course, this may be an even more difficult problem than that of understanding brains in the first place. We must choose a representation scheme for the networks which is amenable to artificial evolution, and selection criteria must be defined. It is not at all clear a priori what combination of representation and selection criteria will be capable of evolving into useful neural circuits. Also, the computational task is now many times larger.

These are some of the issues we address in this project.

Why use genetic algorithms to find the best structures? One reason to use genetic algorithms is to maintain the analogy to biology. Since biological evolution is clearly able to produce interesting neural circuits, a sufficiently detailed model of the entire process should do the same. We have not yet shown that our current system captures the necessary detail, but we have laid some foundations for exploring this area.

In addition to the analogy with biology, we have two other reasons for using genetic algorithms (GAs):

1. GAs provide a means of performing optimization on systems with discrete structures, and mixed continuous-discrete systems.

2. GAs are capable of testing a wide range of possibilities simultaneously [Holland, 1992, preface]. This sometimes enables more efficient exploration of an irregular search space.

Genetic algorithms exhibit *implicit parallelism* as described by John Holland (all quotes in this paragraph are from [Holland, 1992, preface]). Implicit parallelism is "the testing of many *schemata* by testing a single structure." A schema is "a generalization of an interacting, co-adapted set of genes." So, implicit parallelism means that we can test a variety of combinations of genes when evaluating each organism.

We expect the approach to benefit from this feature of genetic algorithms because we expect the optimization landscape to be irregular. Other optimization approaches exist for irregular landscapes, for instance simulated annealing, and might be modified for application to this problem, perhaps in conjunction with the recombination operators.

Why evolve a *developmental* model? Conjecture: developmental models have two properties which may make simulated evolution more fruitful:

- ◇ **robustness** – the process of development can compensate for deleterious changes to the genome or changes in the environment, and still produce a viable organism, and
- ◇ **developmental gain** – a small change in the genome can make a large change in the organism (e.g., add another layer, add another segment).

These terms are also defined in Chapter 1 and shown by example in Chapter 4.

Because of these properties, development might enhance the ability of evolution to search the huge space of possible structures [Kitano, 1990, Goodwin, 1994, Gell-mann, 1990].

This concept has been utilized in Richard Dawkins' "Blind Watchmaker" program and Karl Sims' various artificial evolution programs. These are systems where the *user* is the selection criterion, by choosing which organisms reproduce [Dawkins, 1986, Sims, 1991c, Sims, 1991a]. The user does not judge the 'genotype' directly, but instead is viewing a 'phenotype', which is computed in a non-trivial way from the genotype. This indirection (analogous to a developmental process), gives these programs a much more interesting range of behaviors than they would have if they were representing their outputs directly.

7.2 Discussion and Relationship to Other Work

In this section, we briefly review work involving the combination of evolutionary computation and neural networks, concentrating on the systems most similar to ours. [2] We pay special attention to the systems with developmental aspects.

[2] See [Yao, 1993] for a broad review of evolutionary neural networks.

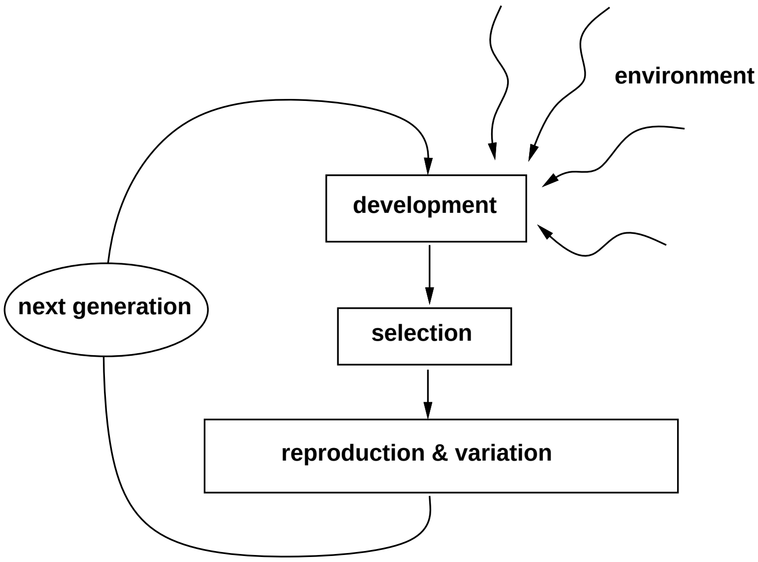


Figure 7.3: In our simulated evolution process, each organism undergoes development selection, and reproduction. before selection. The surviving organisms are subjected to genetic operators (mutation, sex, recombination, etc), to create the next generation. □

Evolving weights of neural networks. The first combinations of evolutionary algorithms and neural networks were systems that use a genetic algorithm to do the neural net learning (e.g., [Beer and Gallagher, 1991, Fogel et al., 1990]). Since the weights form a continuously varying input space, other optimization approaches are equally applicable to neural net learning (e.g., gradient descent, simulated annealing). As described in Yao's review ([Yao, 1993]), there is still dispute about whether genetic algorithms can perform learning more efficiently than other neural network learning algorithms.

Evolving topological structures of neural networks (non-developmental). There have been several recent papers discussing the use of genetic algorithms to create topological network structures without the use of developmental models [Miller et al., 1989, Harp et al., 1989, Whitley and Hanson, 1989, Torreele, 1991, Angeline et al., 1994]. These systems choose representation schemes which directly encode the network, for example a list of connections. These are sometimes called *direct encoding schemes*. Kitano has reported that direct systems such as these generally do not scale well [Kitano, 1990]. In terms of our own criteria, they do not exhibit developmental gain.

Evolving developmental neural networks (non-adaptive). Kitano's paper introduces an *indirect encoding scheme*, evolving a developmental model to obtain better results than the direct encoding schemes [Kitano, 1990]. His developmental model is a grammar-based graph generator, related to L-systems [Lindenmayer, 1968]. Several other grammar-based developmental systems have also been explored [Gruau, 1993, Belew, 1993, Cangelosi et al., 1995].

A feature missing from these developmental models is that they are not adaptive. The developmental process does not include any environmental information, noise, or cell-cell communication. Instead, the network is created by a deterministic decoding of the representation, without any adaptation. In some cases, this can make the representation extremely brittle. For example, in the system by [Cangelosi et al., 1995], small changes in initial conditions or genomes are likely to leave neurons or entire subnetworks completely disconnected from the whole.

The importance of *adaptativeness* during development is addressed further in the following section.

7.2.1 Evolving Adaptive Developmental Neural Networks with Geometric Structure

Adaptation is critical. Imagine a genetic change which causes a new layer of cells to be added to the developing network. In an adaptive developmental system, they will be likely to connect up somehow, either to each other or to other nearby neurons. This comes from their ability to sense and react to an extracellular environment. Kitano has noticed this as well, and his more recent efforts include this concept, which he calls an 'evolutionary super-coupled map', referring to the two couplings of cell-cell interaction and cell-environment interactions [Kitano, 1994b].

Adaptation versus amplification. In a non-adaptive system, the 'development' is actually a deterministic amplification of the genotype. In a real developmental system, the genotype is amplified *in the presence of the environment*, which includes the influence of essentially random processes [Gell-mann, 1990]. The behavior of such "complex adaptive systems" is the subject of considerable interest, and is believed to be important in the process of evolution [Goodwin, 1994].

[Nolfi et al., 1995] present a system which is slightly adaptive, but the extent of the adaptiveness is that when an input neuron doesn't exist, it doesn't get connected to the hidden or output layers. Neurons in their system cannot sense or react to their environment during development. They cannot search out other neurons or find new connections.

[Vaario, 1994] describes adaptive self-organizing networks that have grammar rules to modify cell state. The cells are motile and interact with each other and their environment. In the current implementation, artificial evolution has not yet been attempted, but is stated as a goal.

The Most Closely Related Work

The two systems by [Dellaert and Beer, 1994] and [Kitano, 1994b] are the most closely related to the approach described in this chapter. They are approximately at the same level of completion as our own: the representation scheme has been chosen, and the early results of performing artificial evolution are shown. So far, only structural criteria have been used to guide the evolution. That is, the objective function is measured by some characteristics of the structure (e.g., number of neurons, length of axons), rather than the behavior of the network.

This is clearly just a first step on a difficult path. No one has yet definitively shown that adaptive development is indeed helpful for evolution (either artificial or natural).

It is not yet clear what, if any, aspects of biological development are important for evolution. In our system, we have chosen to include much more detail than the others. They identify the addition of biological detail as an important topic for future work, which encourages us that we might be on the right track, but it still remains to be seen whether this approach will be successful.

Notes on Dellaert and Beer's work. Dellaert and Beer [Dellaert and Beer, 1994] represent cell state as a binary vector which is updated via a Boolean network. The cells form a two-dimensional grid (there is no extracellular space). Using a genetic algorithm, their system can satisfy a structure-based objective function (e.g., try to match a specific set of states in a grid of 64 cells).

The representation scheme used in [Dellaert and Beer, 1994] is a Boolean network [Kauffman, 1993]. [3] Some amount of cell-cell interaction is possible via a 'neighborhood vector' that contains the OR of the binary state vectors of all neighbors. There is no extracellular environment, except for input bits representing 'on the midline' or 'on the external surface' of the organism.

In their future work section, Dellaert and Beer state that they intend to incorporate more biological phenomena (e.g., gradients of morphogens), a less direct mapping from genome to Boolean network, and symmetry breaking (their current system is bilaterally symmetric by definition).

Notes on Kitano's work. Kitano's system [Kitano, 1994b, Kitano, 1994a] is more biological than Dellaert and Beer's but is still less extensive than the one described in this thesis. As mentioned above, he has also noticed the potential importance of cell-cell interaction and cell-environment interactions for artificial evolution. The geometric representation used in his system places the cells on a two-dimensional grid, but it does allow for extracellular effects such as morphogen diffusion as well as local effects to exchange chemicals between adjacent cells. There is no mechanical model of cell interaction or cell migration at this time.

He has shown that the system can increase a structure-based objective function under the influence of a genetic algorithm. The objective function he demonstrates rewards organisms that have more cells and longer neurites. The state equations used in Kitano's system are a fixed type of differential equation which includes terms from the environment and neighboring cells. The genetic algorithm modifies parameters to the equation.

In the future work section, he identifies the need for physical constraints and structure, cell movement, a three-dimensional model, and the use of behavior-based fitness functions.

Other work in Evolution of Adaptive Developmental Systems

Stork et al. present an evolutionary simulation which shows that non-optimal features of a phenotype can arise due to preadaptation (adaptation to a previous selection criterion) [Stork et al., 1992]. In their example, it is shown that a 'useless' synapse in the crayfish tailflip circuit could be due to a previous adaptation to swimming behavior. Later adaptation towards flipping behavior preserved the synapse even though it is not functional for flipping.

In another domain, [Mjolsness et al., 1991, Reinitz et al., 1992] are fitting a developmental model to biological data to gain an understanding of some of the developmental processes in Drosophila (see also Section 1.2). Their model is similar to ours, although they emphasize grammar rules to change states where we use continuously evaluated conditional differential equations. They have successfully run optimization algorithms to find appropriate parameters for their model to fit data, which is analogous in some ways to our attempt to modify the parameters of our developmental model to satisfy a somewhat more arbitrary objective function.

In Yao's review, he suggests that evolving learning rules for artificial neural networks is an important topic for future research [Yao, 1993].

[3] "A Boolean network is much like a cellular automaton, but where in the latter the neighborhood of a node is fixed and consists of neighboring cells, there is no such restriction in the former." [Dellaert and Beer, 1994]

7.3 Using the Cell State Equations as the Artificial 'Genome'

In the genetic algorithm literature, the 'genome' is the representation scheme. The genetic algorithm attempts to find the genome which produces the best result as measured by an objective function. We will use 'gene' and 'genome' in the genetic algorithm sense in this chapter (as opposed to the biological definitions). In our case, the genome is the cell state equations.

The use of the adaptive developmental model described in Chapter 2 has several properties we believe are advantageous for artificial evolution:

- Duplicating a gene in a genome makes more gene product, but does not produce wildly different behavior. Two copies are then available to be modified independently.
- The genes can have regulatory behavior, in that one gene can turn on/off another.
- The genes can have homeobox-like behavior, in that one gene can turn on/off many features all at once. [4]
- Robustness. The genome is expressed via an adaptive developmental model, so deleterious mutations or recombinations may still produce a viable organism. This helps provide some flexibility, so the evolution may be able to tunnel through energy barriers in fitness, out of local minima.
- Developmental Gain. The adaptive developmental model also provides the possibility of making major structural changes with a relatively small change to the genome.

How the += notation for genes helps. The += notation has some particularly nice properties for evolutionary algorithms. For a simple model of gene function [5] (see also Eq. 9.3),

$$dprotein/dt = creation - decay.$$

This means that duplicating a gene simply doubles the rate of gene expression, and doesn't necessarily cause major disruptions in the developmental process (as opposed to some other gene representations). Then one of the copies can undergo severe mutation or recombination while the other copy continues to function as before, perhaps enabling the organism to survive, and hence permitting the evolutionary algorithm to explore more territory or tunnel out of local minima.

Using 'genetic programming' to evolve the cell state equations. Genetic programming [Koza, 1992, Sims, 1991a] is a type of genetic algorithm that operates on LISP expression trees. Mathematical expressions are represented as trees of symbols that undergo 'genetic' operations such as swapping subtrees (recombination) or replacing leaves (mutations). Some details of our implementation are given in Appendix B.

We can represent the cell state equations as expression trees quite easily:

[4] A *homeobox* gene is a type of regulatory gene which directly or indirectly controls the transcription of many other genes. Thus a homeobox gene might be responsible for turning on a cascade of genes to create an entire organ, such as an eye .

[5] Other models of gene function may not combine linearly as this one does.

$$
\begin{array}{llll}
 & \text{cell state equation :} & \frac{dz_{emit\_b}}{dt} \mathrel{+}= & ((z_{ctype} \mathrel{\tilde{>}} 1.5)\ \tilde{\text{or}}\ (z_{ctype} \mathrel{\tilde{<}} 0.5))\ (2.3 - z_{emit\_b}) \\
 & \text{LISP expression :} & \text{(stmt} & \\
(7.1) & \quad \text{condition :} & & \text{(or (> ctype 1.5) (< ctype 0.5))} \\
 & \quad \text{index :} & & \text{emitb} \\
 & \quad \text{consequent :} & & \text{(−2.3 emitb))}
\end{array}
$$

Equation 7.1: Example of a cell state equation and its LISP expression-tree equivalent. □

We would like to allow genetic statements of this sort to undergo both continuous and discrete modifications.

- continuous changes (e.g., changing a numerical constant), and
- discrete structural changes, for example:
  - ◇ add or delete a term,
  - ◇ duplicate a gene,
  - ◇ add, delete or modify a condition (e.g., change $\tilde{>}$ to $\tilde{<}$),
  - ◇ swap conditions between genes,
  - ◇ add the same condition to a set of genes,
  - ◇ and analogous recombination operations (e.g., swap conditions between genes in two genomes).

Structural modifications to the statements are described more fully in Section B.1.

There are many conceivable alternatives for genetic operators, some of which were just listed. One of the areas for future work is to determine which of these are essential for successful artificial evolution with our particular representation.

One of the unique characteristics of our genome representation is the possibility of adding and modifying conditions on groups of genes. Our artificial genes can function similarly to a homeobox gene using a conditional to turn on or off many other equations simultaneously or in sequence. We believe this type of hierarchical operation will enable our representation to maintain and enhance useful building blocks (groups of genes which perform a particular function).

Some way to form groups of functionally related genes is probably important. In real genomes, this is partially accomplished by locality on a chromosome, or proximity within a DNA strand. Things which are close together tend to remain close together under simple recombination operations. In our genome, locality can be accomplished simply using the order of the cell state equations (genes).

Examples of networks the developmental model can generate. Our developmental model can represent a wide range of networks. Some of these are shown in the examples:

◇ cyclically connected ring Figure 4.3,
◇ fully connected Figure 4.5,
◇ hierarchical connectivity Figures 5.21 and 5.22, and
◇ one-to-many mapping Figure 5.20.

These examples show evidence for the ability of the representation to describe networks with asymmetric geometric structure.

Representing and combining building blocks (functional units). Some of the GA search characteristics described by Holland depend on the availability of building blocks and how they can be combined. Building blocks certainly appear to exist in real genomes, as evidenced by the extraordinary conservation of particular DNA sequence across many phyla. Some of these consist of sets of components which operate together to perform a particular function (e.g., cellular clocks [Page, 1994], DNA repair enzymes [Marx, 1994]).

Building blocks can occur at different scales. A set of genes that control the behavior of one cell can be a building block (e.g., to climb a gradient of a chemical). Or a set of genes that operate in multiple cells can be a building block. Consider the CPG network example from Chapter 4. The growth cone of one neuron climbs the gradient of a chemical emitted by another neuron. They also express complementary surface adhesion chemicals. The combination of genes operating in both cells is necessary for the behavior, and is a useful building block.

7.4 The Evolution Simulation

Performing artificial evolution on developmental neural nets requires specifying the following elements:

- ◇ Representation for the artificial genome
- ◇ Developmental model to express the genome as a (neural) structure (with an optional training method)
- ◇ Objective function to evaluate performance of neural structure on the problem
- ◇ Search method to search the space of structures (a genetic algorithm)[6]
  - ◇ Selection criteria to decide which structures to keep
  - ◇ Genetic operators to combine and mutate genomes
- ◇ Initial population of organisms (networks)

Specifying objective functions. The objective function needs to be able to distinguish between the performances of the artificial organisms (the networks, in our case) even if they perform very poorly. In the initial population, the neurons might not even connect up into a network at all.

To deal with this situation, we propose a *graded* objective function, with small rewards for behaviors such as making neural connections (to encourage connectivity in the early stages) and larger rewards for more difficult behaviors such as computing some function we desired. A series of graded objectives that might be useful are: (1) find the input and output terminals, (2) connect input and output terminals, (3) make a signal at output correlated to input, and, finally, (4) compute the desired function output $= f(\text{input})$. Performance on each of these tasks can be measured and then the scores combined with appropriate weights, larger for the more difficult tasks. An alternative is to sequentially change the objective function once a task has been completed.

Nondimensionalized and scaled parameters improve the optimization space. Genetic programming can benefit from applied mathematics techniques, for instance: scale and nondimensionalize to make optimization space better. We mention this in response to a lengthy discussion observed in a GA newslist on the difficulties of dealing with parameters of different scales.

[6] See Section 9.6 for a brief discussion of the relationship between evolution and optimization.

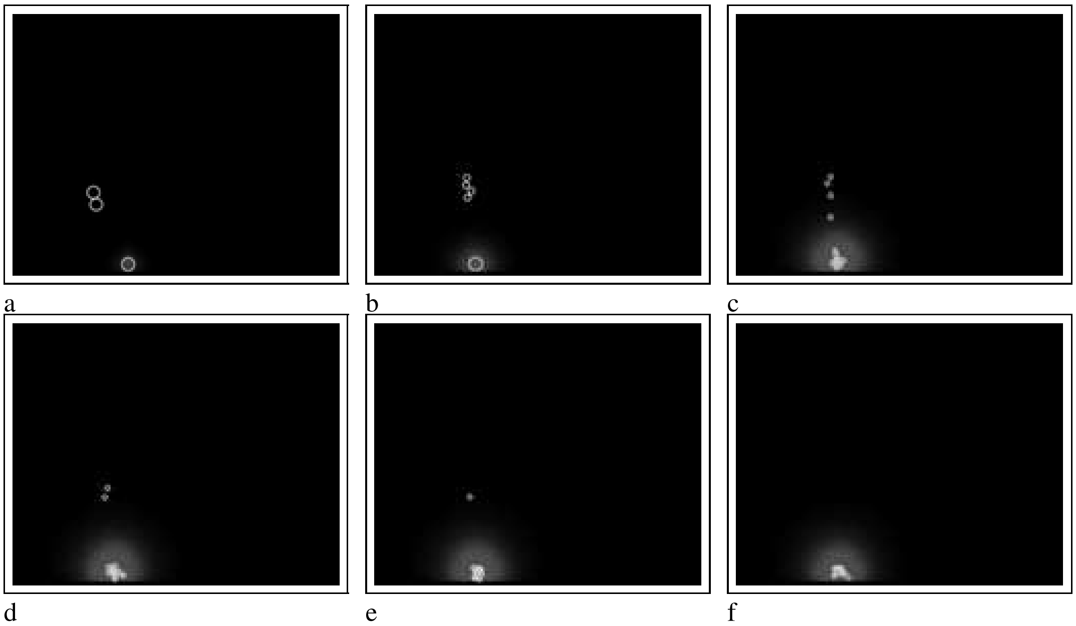


Figure 7.4: This pattern evolved gradient following. □

7.4.1 Evolution 1: Evolution of Parameters to a Fixed Template

In an early version of our system, we tested the representation by choosing some cell state equations and modify only the coefficients and not the form of the equations.

This is an easy implementation, and it enables us to verify that simulated evolution can modify developmental models. We have experimented briefly with some Evolution Strategies[7] to vary the coefficients. The results of these experiments show that the algorithm does modify the genome such that performance under the objective function is improved. The objective function is evaluated after the development process has been run on the genome for a fixed length of simulator time.

Organisms evolved to improve their fitness for a variety of simple selection criteria, such as following gradients, clumping together, splitting, and not taking too much time to compute. Our evolution results are preliminary; in color Figure 7.4 we show some of the simulated evolution results, a developmental model that evolved to follow gradients of diffusing chemicals.

We evolved organisms for up to 60 generations, typically with about 30 organisms per generation, with the largest organisms containing up to 70 cells. Thus, each organism contained up to 3000 discontinuous differential equations that needed to be solved to simulate the life and development of the organism (the organisms contained up to 70 cells $\times$ 16 state variable ODE's per cell + 2000 ode equations for the diffusion grid, producing about 3000 differential equations per organism).

7.4.2 Evolution 2: Implementation Using Genetic Programming

The genetic programming approach described above (Section 7.3) was applied to the developmental model, using two different objective functions.

[7]Evolution Strategies are another variant of genetic algorithms, which computes with continuous variables [Back et al., 1991].

Clump of cells. This graded objective function rewards the following tasks:
- ◇ total number of cells,
- ◇ average distance between cells, and
- ◇ number of cells adhering to each other.

A population of 20 organisms evolved for 50 generations and ended up with small clumps of cells (approximately five cells per clump).

Neurite growth. This graded objective function rewards the following tasks:
- ◇ length of neurites, and
- ◇ number of synapses.

A population of 50 organisms evolved for 100 generations and ended up with cells which emitted many neurites.

7.5 Conclusions

The results of these few and simple evolution experiments convinced us that there was much to be learned about applying genetic algorithms to developmental models. The major problem areas are listed below, as well as suggestions for the next steps to take.

7.5.1 What Is Difficult About This Approach?

The hardest part of this approach (evolving developmental models) is that there are too many parameters. The genetic algorithm has many options, the developmental model has many options, and the objective function and initial conditions need to be specified as well. When it doesn't work, it is hard to know where to look. When it does work, we have to do control experiments to verify that the result wasn't inevitable from the starting conditions. This is very important.

Here are the difficult issues:
- ◇ many parameters to genetic algorithm,
- ◇ many parameters to developmental model (which to vary),
- ◇ holes in the system: need a 'reality check' on parameters to developmental model,
- ◇ search space is too large and includes many things we aren't looking for (things which aren't networks, which have dynamic behavior, etc.),
- ◇ specifying objective functions to get what you want is hard,
- ◇ computation time scales poorly with complexity of networks, and
- ◇ choice of initial population (what is a good starting point?).

Reducing the range of parameter variation on the developmental model will help solve two problems. (1) It will avoid non-physical behaviors such as making state variables grow at ridiculous rates (which also causes computation time to grow). (2) It reduces the size of the parameter space we need to search with the genetic algorithm. This requires that the genetic algorithm has some knowledge of the valid parameter ranges for the developmental model. Similar constraints can be placed on the form of the equations, so that the genetic algorithm doesn't propose ridiculous cell state equations. The downside of this is that we have to characterize these things mathematically and be careful not to restrict ourselves away from potential solutions.

Future directions for this work are discussed in Section 10.2.

Chapter 8

Computer Graphics: Cellular Texture Generation

In this chapter, we apply the developmental model to generate structures for computer graphics. This chapter consists of a paper entitled “Cellular Texture Generation” that appears in the Proceedings of SIGGRAPH ’95 [Fleischer et al., 1995], (c) 1995 Association for Computing Machinery, Inc. (ACM), reprinted by permission. SIGGRAPH is an annual conference of the ACM Special Interest Group in Graphics. This year, SIGGRAPH is being held August 6-11, 1995, in Los Angeles, CA. My co-authors were David H. Laidlaw, Bena L. Currin and Alan H. Barr.

Abstract

We propose an approach for modeling surface details such as scales, feathers, or thorns. These types of *cellular textures* require a representation with more detail than texture-mapping but are inconvenient to model with hand-crafted geometry.

We generate patterns of geometric elements using a biologically-motivated cellular development simulation together with a constraint to keep the cells on a surface. The surface may be defined by an implicit function, a volume dataset, or a polygonal mesh. Our simulation combines and extends previous work in developmental models and constrained particle systems.

Key Words: particle systems, developmental models, data amplification, constraints, texture mapping, bump mapping, displacement mapping

8.1 Introduction

For several years computer graphics researchers and practitioners have been grappling with the problem of creating and displaying surfaces having an organic appearance. Texture maps, bump maps, and related methods often attain the *appearance* of detailed geometry without actually creating it. These techniques do not suffice, however, when the viewpoint is close enough that the three-dimensional (3-D) geometric structure of a surface texture is apparent.

We are interested in making images of surfaces covered with interacting geometric elements, such as scales, feathers, thorns, and fur. We model these elements as small 3-D cells constrained to lie on a surface. The cells interact to form *cellular textures*: surface textures with 3-D geometry,

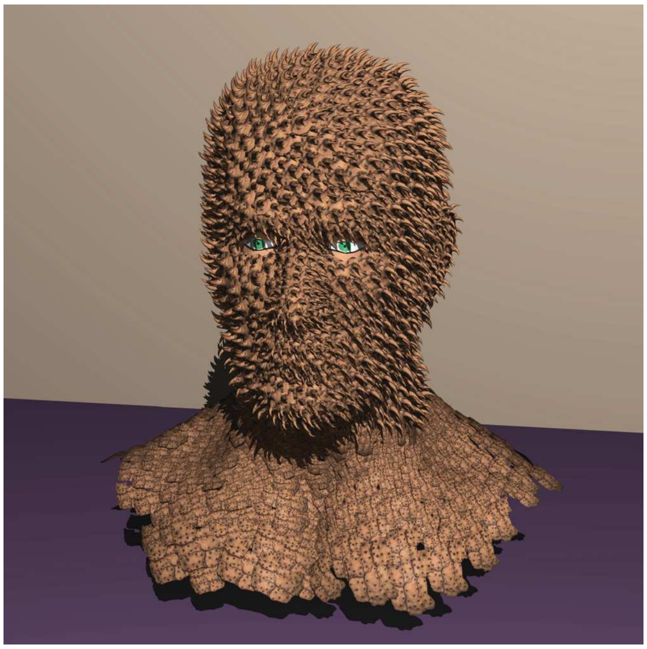


Figure 8.1: Thorny Head: Both flat and thorn-shaped cells are constrained to lie on a surface defined by a polygonal dataset of a human head. Flat cells are used in the neck and chest regions, while thorn-shaped cells are used on the head. The orientation of each thorn approaches that of its neighbors, leading to a continuous field of thorns that sweeps across the head. The size of the thorns is related to the level of detail of the model; smaller thorns are placed on smaller features. □

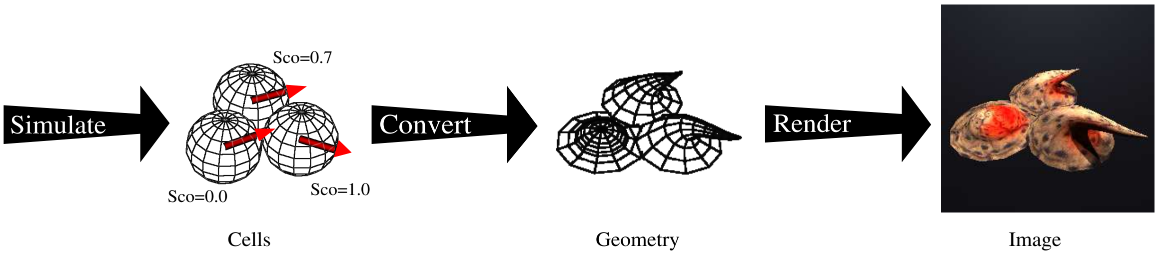


Figure 8.2: The cellular particle simulator computes the locations, orientations, and other values associated with the cells. This information is converted to geometry and appearance parameters, which is then passed to a renderer to create the image. Note that the cell orientations (red arrows) become the orientations of the thorns. Using a geometric modeler, we created a geometric object that changes shape from a bump to a thorn based on a single parameter [Snyder, 1992]. We use the cell state variable S_{c0} to control this parameter (Section 4). □

orientation, and color. Our approach combines properties of particle systems, developmental models, and reaction-diffusion methods into one system. Figure 8.8 shows an example combining all of these approaches.

There are a few challenges in making images of these types of materials:

- The geometry is often too pronounced for using texture- or bump-maps.
- It is often difficult to map appropriate texture coordinates onto the global geometry and topology.
- The placement, orientation, coloration, and shape of the individual elements may depend on:
  - neighboring elements,
  - surface characteristics such as local curvature, or
  - global phenomena such as sunlight.

Because of the potentially complicated interdependencies of the elements, it is difficult to create either geometric or textural models of such objects by hand. So we turn to automatic data-amplification techniques, which are similar to the structured particle systems used to generate models of plants [Reeves and Blau, 1985, Smith, 1984].

Developmental Approach For generating organic patterns, it is natural to consider a biologically-based simulation. In previous work [Fleischer and Barr, 1994], we developed a biological developmental model to simulate and study patterns generated by the motions and interactions of discrete cells (Figure 8.3). These artificial cells move about, grow, and divide in a simulated petri dish, the *extracellular environment*. The extracellular environment can contain physical barriers, diffusing chemicals, gravity, etc. The ability to form a variety of interesting patterns with the system has prompted us to explore its application to geometric texture generation (Figure 8.1).

The textures we model are formed from many interacting geometric elements. Actual fur, scales, and thorns may be formed from single cells or multiple cells [Nagorcka et al., 1987]. In either case, we assume that the texture patterns arise from the interactions of discrete elements capable of movement and orientation change, and model each of these elements as one cell. The patterns are formed as the cells experience physical processes of collision, adhesion, and other local interactions.

Software Structure The approach advocated by this paper is to automatically grow cellular textures by simulating discrete cells on surfaces. We then convert the resulting cellular information into model geometry and coloration, which is rendered. The images of this paper were generated using oriented, spherical cells, which are converted into thorns, scales, and other shapes for rendering (Figure 8.2).

Overview The remainder of this paper is structured as follows. Section 8.2 describes related work. It is followed by an overview of the system architecture in Section 3. Section 8.4 describes the cellular particle system, and includes examples of cell programs that implement various behaviors. In Section 5 we discuss the particle converter, which produces geometry from the cell positions, orientations and other parameters.

Results are presented in Section 8.6, which describes the examples shown in the figures. The final section presents a discussion of the approach and some directions for future work.

8.2 Related Work

This approach is a synthesis and extension of work ranging from morphological models to general texture mapping. In this section, we discuss our approach in the context of four related areas:

- ◇ Levels of Detail
- ◇ Biologically Motivated Morphogenesis
- ◇ Reaction-Diffusion Methods
- ◇ Particle Systems

Levels of Detail Choosing the appropriate level of detail for image synthesis at a given viewing distance has long been recognized as an important topic in computer graphics [Crow, 1982, Kajiya, 1985, Kajiya and Kay, 1989]. At large scales, geometric models are necessary; intermediate scales, texture mapping and similar techniques may be sufficient; at the smallest scales, illumination models suffice to describe the microgeometry of the object [Westin et al., 1992].

The level of detail of the models addressed in this paper falls somewhere between the use of hand-crafted geometric models and bump- or texture-mapping. A range of geometric levels is available to us because of the modular nature of our technique.

Complex, oriented textures have been created and rendered in many ways, notably with texels [Kajiya and Kay, 1989]. The texel approach is intermediate between geometry and mapping techniques, but leaves open the question of how to arrange the texel elements appropriately. Our approach addresses this problem, and can produce models to be rendered using texels.

Displacement mapping is another technique for adding geometric detail to surfaces [Cook, 1984]. As with texels, the displacement mapping technique does not address the problem of determining which displacements are necessary to create a specific effect, such as a field of similarly oriented thorns. A possible application of our technique is to create such displacement maps, for example by creating flow fields [Pedersen, 1994].

Biologically Motivated Morphogenesis The cellular development system which forms the basis of this work [Fleischer, 1994, Fleischer, 1995, Fleischer and Barr, 1994] incorporates elements of several established biological models of morphogenesis: Turing's morphogens [Turing, 1952], Odell's mechanical models [Odell et al., 1981], and Lindenmayer-system cell lineage determinants [Prusinkiewicz and Lindenmayer, 1990], as well as our own model of cell contact and adhesion.

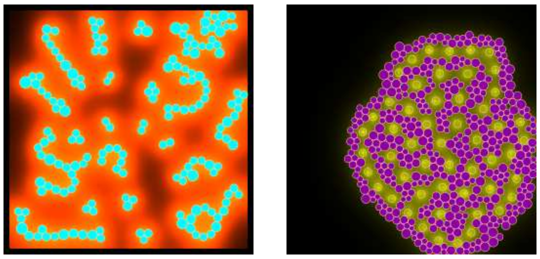


Figure 8.3: These images demonstrate the pattern formation capabilities of our 2-D cell simulator [Fleischer and Barr, 1994]. □

Much well-known computer graphics work is biologically based. The combination of developmental models with geometric constraints enables the creation of many organic patterns. It has been explored in work on plant growth [Greene, 1989, Prusinkiewicz et al., 1994], plant organ placement [Fowler et al., 1992b], and seashell patterning [Fowler et al., 1992a].

Interacting geometric elements were used by [Fowler et al., 1992b] to model the placement of plant organs. Our cells are a generalization of these elements, with many additional capabilities, including independent movement, adhesion, and changes in size and orientation due to cell-cell interaction.

In [Fowler et al., 1992a], pigmentation patterns on seashells are modeled using reaction-diffusion equations on surfaces defined by sweeping a generating curve along a logarithmic spiral. This shares with our work the concept of applying pattern formation models on 3-D surfaces. Their use of continuous reaction-diffusion equations to generate the patterns differs from our use of discrete cells. For the types of cellular texture we are investigating, the choice of a discrete model seems appropriate.

Spatially-oriented models of plant growth are capable of generating attractive plant images [Arvo and Kirk, 1988, Greene, 1989]. The placement of geometric objects in the environment of plants affects their growth. The importance of combining environmental and endogenous mechanisms in forming organic shapes in computer graphics has also been demonstrated using environmentally-sensitive L-systems [Prusinkiewicz et al., 1994], which allow interaction between the environment and the development of a structure defined by an L-system. In an application to synthetic topiary, elements sense their global position and orientation, and are pruned according to a bounding surface. Our work also combines geometric environmental factors with an endogenous developmental model to describe cell behavior. We differ from these plant models by the use of discrete motile cells that are able to move and rotate independently.

Reaction-Diffusion Methods Reaction-diffusion equations were first proposed as a model for morphogenesis by Turing [Turing, 1952]. They are a continuous approximation to a sheet of many discrete cells interacting over time. Our system models discrete cells explicitly, and can generate patterns similar to continuous reaction-diffusion equations since it is actually a more detailed model of the same biological system.

Reaction-diffusion equations have been successfully applied to the generation of texture maps [Turk, 1991, Witkin and Kass, 1991]. Because they are based on natural phenomena, they have an appealing organic quality. In addition, they avoid problems of parameterization and topology by creating the pattern directly on a surface. Our approach shares both of these benefits.

In our 2-D implementation (Figure 8.3), we include both discrete cells and a continuous reaction-diffusion computation. The two models are also able to interact, since the discrete cells can sense and emit the continuously diffusing chemicals. The 3-D implementation does not yet support continuously diffusing and reacting chemicals. However, we are able to reproduce some forms of reaction-diffusion behavior using cell-cell interaction in the discrete model.

Particle Systems Early particle systems [Reeves, 1983, Sims, 1990] had little or no interparticle interaction, unlike later work based on molecular models and other criteria [Miller and Pearce, 1989, Terzopoulos et al., 1989]. Our work includes elements of Witkin and Heckbert's surface-constrained particles [Witkin and Heckbert, 1994], and the orientation constraints of Szeliski and

Tonneson [Szeliski and Tonnesen, 1992]. Reynold's "boids" [Reynolds, 1987] introduced somewhat more sophisticated interacting particles with programmable behaviors. In addition, his boids, like our interacting cells, can sense and react to each other and to their environment.

8.3 Software Architecture

We arrange the process into three software modules (Figure 8.2).

cellular particle simulator with surface constraints: computes locations, orientations, sizes and other parameters of cells based on a behavioral specification. Allows particles to be constrained to a surface (implicit, polygon mesh, or volume dataset isosurface).

parameterized particle-to-geometry converter: converts cell positions, orientations and other parameters into shape and appearance parameters.

renderer: takes shape and appearance parameters plus a scene description and renders the scene. The images in this paper were generated with John Snyder's ray tracer [Snyder and Barr, 1987], which was chosen primarily for its speed on large datasets.

The implementation of our framework involves the addition of cell-cell interactions, orientation constraints and surface constraints to a more traditional particle simulator. A simple version of a particle converter can be implemented using a geometric modeler to place a geometric object at each particle's location with the appropriate size and orientation. The cellular particle simulator and particle converter are described in further detail in the Sections 8.4 and 8.5.

8.4 Cellular Particle Simulator

The cellular particle system combines cell-cell interactions, cell-cell adhesion, oriented particles, and surface constraints into one unified framework. Additional discussion of the cell simulator and its implementation can be found in [Fleischer, 1995, Fleischer and Barr, 1994].

Our system allows the user to specify cell behaviors such as 'go to a surface' and 'align with neighbors' by combining modular *cell programs*. Cell programs are first-order differential equation terms that modify the cell's state (see Table 1).

In Section 8.4.1 we discuss how to use the simulator, and then delve into a more detailed mathematical description of the cell programs in Section 8.4.2. Section 8.4.3 describes the cell programs used to create simulations like those shown in the figures. Section 8.4.4 presents methods for incorporating various types of surfaces into our surface constraint method.

8.4.1 Using the Simulator

For a particular simulation run, the user defines

- ◇ the cell state variables,
- ◇ the extracellular environment, and
- ◇ the cell programs.

The user also specifies initial placements and other initial conditions for the cells.

Users can control the simulation by writing cell programs to describe the behaviors of the cells, and by putting surfaces into the environment. More direct interaction is also possible during the simulation process. The user can halt the simulation, change the cell programs, and choose individual cells or groups of cells to modify or remove. The simulation is then restarted with the modifications. It is sometimes convenient to freeze certain cell values when they have reached a desirable state. Frozen values remain fixed while others continue to vary when the simulation is restarted. Particular seed cells can also be placed and frozen in situations where the user wishes to achieve a certain effect. For instance, frozen cells with a particular orientation could encourage fur to run in a particular direction on a surface.

8.4.2 Definitions

A *cell* is an entity that has position, orientation, shape, and an arbitrary length state vector for parameters such as chemical concentrations in a reaction-diffusion simulation. It is a generalization of a particle in a particle system.

Cell State Variables The state of a cell,

$$\begin{aligned}\vec{S} \;=\; (\;& \vec{p}, \vec{q}, r, S_{die}, S_{split} \\ & S_{c0}, S_{c1}, S_{c2}, \cdots, S_{a0}, S_{a1}, S_{a2}, \cdots)\end{aligned}$$

is a vector containing values representing position ($\vec{p}$), orientation ($\vec{q}$), size (r), and concentrations of chemicals within the cell (S_{ci}) or in the cell membrane (S_{ai}). The variable S_{split} is used to trigger an event, the cell splitting. This occurs when the value of this state variable exceeds a threshold, θ_{split}. S_{die} is defined similarly, with an associated threshold, θ_{die}.

In real cells, chemicals in the cell membranes of adjacent cells can bind together and enable cells to sense that they are in contact. The chemicals can also be adhesive. The binding of membrane chemicals is specific; some chemicals bind in complementary pairs, and others bind to themselves. Our model allows the user to specify the adhesive properties of the membrane chemicals, and provides the amount of each bound membrane chemical as an environmental parameter (described below).

To define a cell's motion, we specify cell programs that contribute to $\vec{p}'$, the viscous force on the cell. This is the *attempted* motion of the cell, which is further modified by the influence of collisions, adhesion, and viscous drag. We do not currently compute inertial dynamics, but instead use viscous dynamics ($F = mv$), which makes the cells easier to control and predict. Collision forces are computed using a polynomial penalty function (kx_o^n where x_o is the overlap between two cells).

We represent cell orientation in 3-D using a quaternion. In the exposition that follows, we sometimes refer to the cell's local coordinate frame using the three basis vectors: $\vec{e}_x, \vec{e}_y, \vec{e}_z$. This is the coordinate frame obtained by rotating from the lab frame using quaternion $\vec{q}$.

Extracellular Environment The cell's external environment is a vector of parameters that are provided as an input to the cell programs. These parameters describe everything the cell can sense from its current position:

$$\begin{aligned}\vec{\Lambda} \;=\; (& \Lambda_{a0}, \Lambda_{a1}, \Lambda_{a2}, \cdots, \Lambda_{p0}, \vec{\nabla}\Lambda_{p0}, \Lambda_{p1}, \vec{\nabla}\Lambda_{p1}, \cdots, \\ & \vec{\Lambda}_{v0}, \vec{\Lambda}_{v1}, \cdots, \\ & \vec{\Lambda}_{\omega x}, \vec{\Lambda}_{\omega y}, \vec{\Lambda}_{\omega z}, \Lambda_{u0}, \Lambda_{u1}, \cdots)\end{aligned}$$

The Λ_{ai} values represent the amounts of membrane chemicals that are bound to membrane chemicals on neighboring cells. The value and gradient of a potential field, such as an implicit function or the concentration of a diffusing chemical, are provided in Λ_{pi} and $\vec{\nabla}\Lambda_{pi}$. These fields are evaluated at the current location of the cell, and will generally have different values at different locations. Other scalar and vector fields can be provided in Λ_{ui} and $\vec{\Lambda}_{vi}$, which can also be functions of position.

The orientation of a cell relative to its neighbors is made available to the cell programs in the vectors $\vec{\Lambda}_{\omega i}$. This vector describes the rotation that would align this cell's $\vec{e}_i$ axis with the average orientation of the adjacent cells. This parameter is used to align the orientations of cells, as shown in Figure 8.9(a) and others. The direction $\vec{\Lambda}_{\omega i}$ specifies the axis of rotation, and the magnitude specifies the rotation angle (similar to angular velocity). As an example, consider the computation of the average relative x-axis for a cell b, computed as a sum over neighboring cells c:

$$\vec{\Lambda}_{\omega x} = \frac{1}{n} \sum_{c \in \text{ neighbors}} \frac{\vec{e}_x^{\,b} \times \vec{e}_x^{\,c}}{||\vec{e}_x^{\,b} \times \vec{e}_x^{\,c}||} \cos^{-1}(\vec{e}_x^{\,b} \cdot \vec{e}_x^{\,c})$$

where $\vec{e}_x^{\,c}$ is the x-axis of the cell c, and cell b has n neighbors.

Cell Programs Each cell has several cell programs, which are first order differential equations describing how its state changes over time. Examples are given in Table 1 and Section 8.4.3. A cell program is a function of the cell's current state $\vec{S}$ and its environment as expressed by $\vec{\Lambda}$. Different types of cells use different cell programs or different combinations of the same cell programs to define their behaviors. Even if two cells share the same set of cell programs, they will generally behave differently because they experience different local conditions depending on their position.

The entire system of differential equations to be solved is obtained by superposing ordinary differential equations from the cell programs for every cell. Additional equations arise from computation in the environment (e.g., diffusion of chemicals, although this is not in the current 3-D implementation). In order to handle discontinuous changes, such as when cells are created or die, we use a piecewise ordinary differential equation solver [Barzel, 1992, App. C].

Mathematical Basis for Cell Programs Differential equations are a general tool for creating dynamic behavior. In our cell programs, we employ equations arising from physical models, as well as those arising from constraint solution techniques.

Higher order linear differential equations, such as those for mechanical or chemical systems, can be rewritten as multiple first order differential equations (i.e., cell programs) with the addition of state variables. In this case, the simulation dynamics reflect the dynamics of the equations.

In order to write constraints as cell programs, we formulate them as energy functions[1] to be minimized [Witkin et al., 1987]. Each constraint is expressed as an energy function $E_i(\vec{S}, \vec{\Lambda})$ of the

[1]Note that when we use constraint-based cell programs, the dynamics of our simulation depends upon the gradient descent algorithm, and is not necessarily physically meaningful.

state of the system, $\vec{S}$, and the parameters describing the environment, $\vec{\Lambda}$. Hard constraints could also fit into this framework using Lagrange multipliers [Platt, 1989].

Using relative constants k_i to weight the soft constraints, we express the overall energy to minimize as:

$$\hat{E}(\vec{S}, \vec{\Lambda}) = k_1 E_1(\vec{S}, \vec{\Lambda}) + \cdots + k_n E_N(\vec{S}, \vec{\Lambda})$$

We can minimize this energy according to gradient descent by modifying the state according to

$$\frac{dS_\ell}{dt} = -\sum_{i=1}^{N} k_i \frac{\partial E_i}{\partial S_\ell}$$

Superposition of Cell Programs Because the overall energy is expressed as a sum, the cell programs dS_ℓ/dt are also sums of terms, one for each constraint. We find it convenient to write cell programs as incremental collections of constraints. We write this as $dS_\ell/dt += \{\text{constraint term}\}$ in our example programs in Table 1. Multiple cell programs can thus be added together conveniently.

Figure 8.4: Table 1. □

Behavior	**Environment Requirements**	**Cell Program**
Go to a surface.	An implicit surface $f(\mathbf{x}) = 0$.	$\mathbf{p}' \mathrel{+}= -k\, f(\mathbf{p}) \nabla f(\mathbf{p})$
Die if too far from surface.	An implicit surface $f(\mathbf{x}) = 0$.	$S'_{die} \mathrel{+}= \frac{1}{d} f(\mathbf{p})\, \theta_{die} - S_{die}$
Align an axis with a vector field.	A vector field, $\mathbf{v}(\mathbf{x})$. $\mathbf{e}_y$ is the cell's y-axis.	$\omega_v \equiv k \frac{\mathbf{e}_y \times \mathbf{v}(\mathbf{p})}{\lVert \mathbf{e}_y \times \mathbf{v}(\mathbf{p}) \rVert} \cos^{-1}(\mathbf{e}_y \cdot \frac{\mathbf{v}(\mathbf{p})}{\lVert \mathbf{v}(\mathbf{p}) \rVert})$ $\mathbf{q}' \mathrel{+}= (1/2)\, \omega_v\, \mathbf{q}$
Align x-axis with neighbors.	$\mathbf{\Lambda}_{\omega x}$, x-axis orientation relative to neighbors.	$\mathbf{q}' \mathrel{+}= (1/2)\, \mathbf{\Lambda}_{\omega x}\, \mathbf{q}$
Align z-axis with neighbors.	$\mathbf{\Lambda}_{\omega z}$, z-axis orientation relative to neighbors.	$\mathbf{q}' \mathrel{+}= (1/2)\, \mathbf{\Lambda}_{\omega z}\, \mathbf{q}$
Maintain unit quaternion.		$\mathbf{q}' \mathrel{+}= 4k(1 - \mathbf{q} \cdot \mathbf{q})\mathbf{q}$
Adhere to other cells	Membrane chemical *a2* which binds to itself.	$S'_{a2} \mathrel{+}= 1.0 - S_{a2}$
Divide until surface is covered.	Λ_{a2}, amount of *a2* which is bound.	$S'_{split} \mathrel{+}= \phi(\gamma, \Lambda_{a2})\, \theta_{split} - S_{split}$
Set size relative to surface feature size	Λ_{u0}, a value which reflects the size of the nearest feature on the surface.	$r' \mathrel{+}= \Lambda_{u0} - r$
Example of reaction-diffusion in discrete cells.	Λ_{a0}, Λ_{a2} amounts of bound membrane chemicals. The user has specified that the membrane chemical *a0* binds to *a1*, and that *a2* binds to itself.	$S'_{c0} \mathrel{+}= -20\, \Lambda_{a0}/\Lambda_{a2} +$ $10\, S^2_{c0}/(1 + S^2_{c0}) - 0.5\, S_{c0} + 13$ $S'_{a1} \mathrel{+}= 0.95\, \Lambda_{a0}/\Lambda_{a2}$ $S'_{a1} \mathrel{+}= \phi(3, S_{c0})\, S_{c0}$ $S'_{a1} \mathrel{+}= -S_{a1}$ $S'_{a0} \mathrel{+}= 5 - S_{a0}$ $S'_{a2} \mathrel{+}= 1 - S_{a2}$

Table 1: Example Cell Programs. Scalar or vector fields are given as a function of spatial location, $\mathbf{x}$, and are evaluated at the current location of the cell, $\mathbf{p}$.

8.4.3 Example Cell Programs

The cell programs listed in Table 1 exemplify the types of cell programs used to make the figures in this paper. We describe each briefly below.

Remember that the contributions from multiple terms are added together to make a single differential equation for each state variable (using the $+=$ notation). Many of the cell programs shown here are of the form $S_i'(t) = dS_i/dt = \beta - S_i(t)$ for some constant β, which causes S_i to quickly approach the value β.

Go to a surface. This cell program implements a constraint to keep a cell on the implicit surface $f(\vec{x}) = 0$. An approximation to the gradient, $\vec{\nabla} f(\vec{x})$, is also available in the environment. As the simulation runs, a cell with this program will descend the gradient and come to rest on the surface. The parameter k determines the speed with which the particle approaches the surface.

Die if too far from surface. Recall that when the variable S_{die} crosses the threshold θ_{die}, it triggers cell death (Section 8.4.2). In this cell program, we cause S_{die} to rise towards the threshold quickly if the cell is greater than a certain distance from the surface. Computing this requires a measure of the distance from the surface. For the implicit surfaces used in the figures, $f(\vec{x})$ is an approximation to the distance from the surface.

The equation in Table 1 causes cells at a distance greater than d to die. Similar cell programs can cause a cell to die if it becomes too large, its orientation strays too far from neighboring cells, or due to any other condition that is a function of the cell's environment and internal state.

Align with a vector field. In this example, we align the cell's y-axis, $\vec{e}_y$ with a given vector field $\vec{v}(\vec{x})$. The vector field is evaluated at the cell's current location, $\vec{p}$.

We first compute the vector $\vec{\omega}_v$, which represents the transformation required to rotate the $\vec{e}_y$ into $\vec{v}$. $\vec{\omega}_v$ is the axis of rotation, and the length of $\vec{\omega}_v$ specifies the angle through which to rotate. The formulation works with any of the cell's axes.

The form of the cell program comes from the equation

$$\vec{q}\,' = (1/2)\,\vec{\omega}\,\vec{q},$$

which defines the rate of change of a quaternion, $\vec{q}$, for angular velocity $\vec{\omega}$ [Goldstein, 1980].

If we have multiple cell programs of this form, the ω_i terms add, which allows us to constrain one, two, or all three axes of the cells. If the orientation constraints conflict, the cell's orientation will approach the average orientation. If they don't conflict, all constraints will become satisfied.

Align with neighbors. The three orientation constraints on the cell's x-, y- and z-axes fully constrain its orientation. Constraining two axes would be sufficient in most cases, except where the vector field $\vec{v}(\vec{x})$ happened to be collinear with the x- or z-axis. Having the extra constraint keeps us from running into problems in that case, and also aids the convergence of the cell alignment process.

Maintain unit quaternion. It is wise to add a constraint to ensure that the quaternion does not stray too far from a unit quaternion during the integration of the differential equations. We can do this with another simple constraint. Table 1 shows a term for this constraint that comes from minimizing the energy expression $E = (1 - \vec{q} \cdot \vec{q})^2$, which describes the deviation of $\vec{q}$ from a unit quaternion.

Adhere to other cells. This equation causes the variable S_{a2} (representing the surface chemical *a2*) to approach and stay at the value 1.0. A pair of cells expressing *a2* will stick together once they come in contact, and a force is required to pull them apart. The environment vector for each cell will report the amount of chemical bound on each cell, Λ_{a2}, which may be used in other cell programs to determine if the cell has contacted another cell. The amount bound is computed from the contact area between the two cells, and the concentrations on each cell.

Divide until surface is covered. Divide until the amount of bound surface chemical *a2* reaches the level γ. Λ_{a2} reports the total amount bound from all cells that are in contact, which gives the cell a means of determining how many neighbors it has. Note that the mechanism has more general utility than just counting neighbors. For instance, a cell with twice the concentration of *a0* will contribute more to Λ_{a2}.

The auxiliary function $\phi(\gamma, \Lambda_{a0})$ is used in this cell program to compute a continuous version of the condition $(\Lambda_{a0} > \gamma)$. The function $\phi(a, b)$ computes a continuous version of the boolean condition $(b > a)$:

$$\phi(a, b) \equiv (tanh((b - a)) + 1)/2.$$

The value of this function will be near one for $(b >> a)$ and near zero for $(a << b)$.

Set size relative to surface feature size. In Figure 8.1, the cell sizes are related to the sizes of features in the polygonal database. This is achieved by providing the cells with a value Λ_{u0} that represents the area of the nearest triangle. The value Λ_{u0} could be used to pass information about local curvature or any other parameter that we wish to use to change the cell behavior.

Example of reaction-diffusion in discrete cells. The full derivation [Fleischer, 1995] of this set of equations is beyond the scope of this paper, however we will describe the equations briefly. The first equation in this set defines a genetic switch [Murray, 1993] that tends to drive S_{c0} towards one of two values, depending on the influence of the term $\Lambda_{a0}/\Lambda_{a2}$. In terms of Meinhardt's activator-inhibitor models [Meinhardt, 1982], S_{c0} is the activator and S_{a1} is the inhibitor, which is propagated by the activity of membrane chemicals. The other equations determine interactions of membrane chemicals that lead to an effective diffusion of the value of S_{a1} among the cells. The value of S_{c0} can then be used to determine the final rendered shape of the cell, as illustrated in to Figures 8.2 and 8.8.

8.4.4 Surface Constraints

We have applied surface constraints to a variety of surface classes:

- polygonal mesh,
- implicit function, and
- isosurfaces of volume data.

The surface constraint cell program evaluates an implicit function to enable the cell to find and stay on the surface (Table 1).

Several of the surfaces used to create the figures are defined by triangular meshes. We create a rough approximation to an implicit function for these meshes. Any implicitization method will work, and in fact it doesn't have to be very exact. We implement this function by constructing an approximate *kd*-tree for the triangular mesh. In constructing a true *kd*-tree, each additional vertex may add several new partitions. Our approximation adds only the one necessary partition for each added vertex. This makes the tree smaller and faster to precompute, but no longer actually gives the closest point. To evaluate the function, we look up the triangle center in the *kd*-tree supposedly nearest to a given point. We then check adjacent triangles to see if they are closer, to improve the characteristics of the approximation. We then compute the direction and distance to that triangle, and use it as we would the gradient of an implicit function. We find the approximation, in conjunction with the local search, to be satisfactory for this application.

8.5 Particle Converter

The particle converter converts information about the particles and their environment into geometry and appearance parameters for rendering. It receives all of the results of the simulation, including the position, orientation and size of each cell, concentration of reaction-diffusion chemicals, and other arbitrary user-defined parameters, such as type or color. It also may have access to information about how far each cell is from the surface and properties of the surface near the cell (e.g, curvature) The converter also knows which cells contact each other.

The particle converter concept has proven to be extremely convenient. It enables us to do a variety of useful operations, including:

- ◇ choosing an appropriate representation for each cell based on its screen size (Figure 8.5);
- ◇ smoothly changing the appearance of a cell based on a continuously varying parameter (Figure 8.8);
- ◇ using the cell positions to generate a spatial subdivision (similar to [Szeliski and Tonnesen, 1992, Turk, 1992, Witkin and Heckbert, 1994]);
- ◇ using the cell orientations to compute a flow field on the surface (useful for displacement maps [Pedersen, 1994]); and
- ◇ experimenting with various colorations and geometries using the same simulation dataset.

The output of the particle converter is a collection of geometry and appearance information suitable for a particular renderer. This collection will generally include one or more geometric primitives for each cell, and the local texture, transparency, or bump information. The geometry can be simple, as in Figure 8.6, where each cell is rendered as a few polygons with a mottled-green texture map. Or it can be more complicated, as in the parameterized 3D thorn shape that curls based on cell state information (Figure 8.8 (c)).

The particle converter can also use contact information to calculate the size or shape of geometric primitives based on neighboring cell proximity, or to interpolate parameters such as orientation between cell centers.

We have implemented two particle converters. One provides options for choosing a particular geometry and texture for each cell. In addition, it considers information associated with the underlying polygons, such as which body part it represents in an anatomical model (lips, eyes, etc.) This can be used to change the rendering of cells in certain areas, as can be seen on the lips of the man in Figure 8.1.

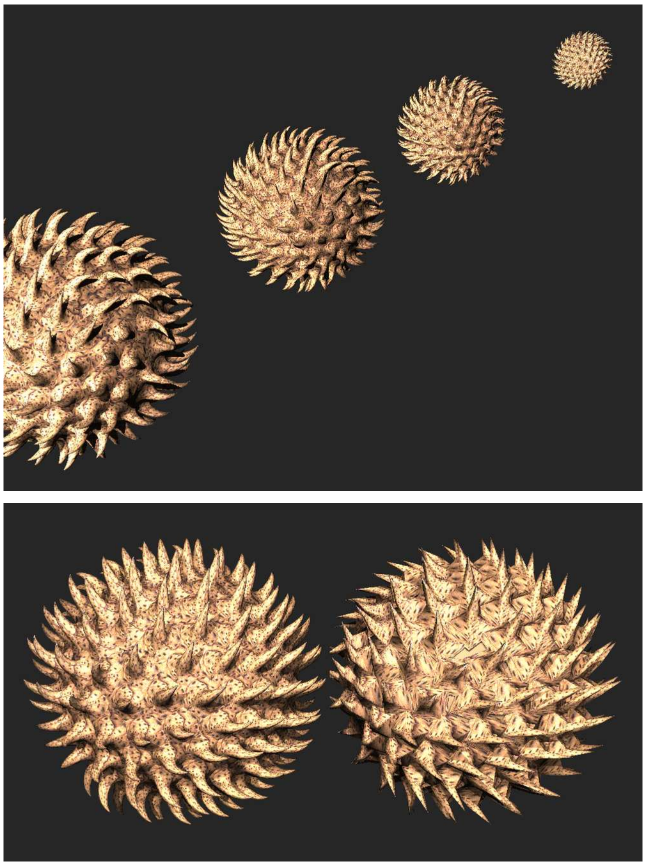


Figure 8.5: In the top image, the thorny spheres at further distances are rendered with fewer polygons. The bottom image shows a closeup of the nearest and furthest objects, so we can see the reduced number of polygons. □

Our second particle converter was implemented using a general purpose modeler [Snyder, 1992]. Taking advantage of the flexibility of this modeler, we can create parameterized objects such as the bump-to-thorn shape shown in Figure 8.2. The modeler is used to create the thorny spheres in Figure 8.8.

Rendering an appropriately scaled representation One of the drawbacks of data amplification techniques [Smith, 1984, Reeves and Blau, 1985] such as ours is their ability to generate a ridiculous amount of geometry to render. To ameliorate this, we use the particle converter to choose geometric primitives appropriate to the size of the object in the final image (Figure 8.5). This approach could be carried even further, for instance, by creating texture maps based on the cell positions.

8.6 Results

In this section we list a series of examples that highlight features of our system.

Scales Figure 8.6 shows four views of a spherical object with a uniform covering of similarly-oriented cells. The cell programs used here incorporate terms to divide until the surface is covered,

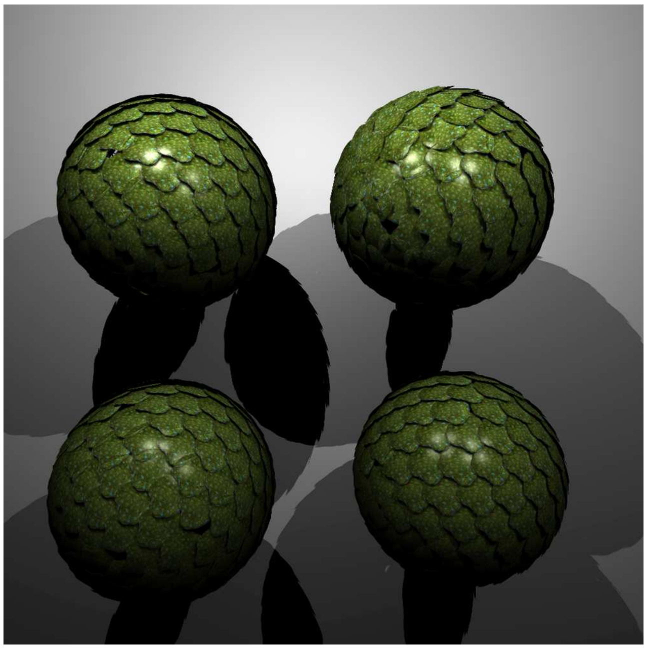


Figure 8.6: Scales: Four views of a spherical object uniformly covered with similarly oriented cells. Each cell is rendered as a group of four polygons with a texture and transparency map. The polygons are tilted slightly to give a layered appearance. □

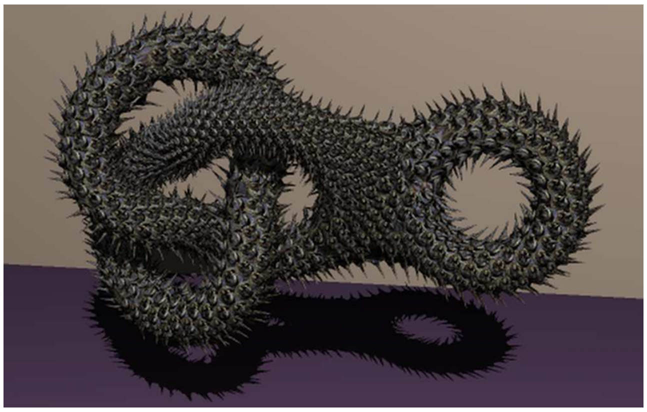


Figure 8.7: Cellular textures can handle unusual topologies. □

Figure 8.8: Varying Thorns. Reaction-diffusion-like equations determine the pattern of bumps and thorns on these spheres. Note the continuously varying thorn height and thorn curvature on the center and rightmost spheres. □

to stay on the surface, and to die if pushed too far off of the surface. Initially, several cells were placed near the surface, and allowed to divide and wander. The cells were also given a soft constraint to align their y-axes with the gradient of the surface implicit function, and to align their x- and z-axes with their neighbors.

Note that two singularities in the orientations of cells arise naturally on the sphere, due to its topology. One is visible on the near side of the upper right sphere. Unlike standard texture mapping, this method introduces no particular parameterization problems, such as stretching or shrinking of the texture.

A Knotty Problem Figure 8.7 shows that the cellular texture approach is capable of creating textures for surfaces with unusual topologies. It is not necessary to have a parameterization for the surface. This surface was designed by John Hughes and John Snyder.

Thorny Head: Changing cell size to match surface features The examples described so far have used cells that are relatively uniform in size. Figure 8.1 shows an example where the cell size is related to the detail level in the underlying polygonal model. We achieve this by providing the cells with another environmental variable: the area of the nearest triangle in the underlying polygon mesh. Note the finer texture and geometry around the eyes and mouth.

Different rendering parameters were chosen according to properties of the underlying polygonal model. Each polygon in the underlying database is associated with a region of the body. The particle converter assigned different shading properties to the cells in the head and neck regions. At the eyes, the underlying polygonal representation shows through the cell texture.

Thorny Spheres: Differentiated cellular textures This example shows several important capabilities of the system. It shows

- ◇ the creation of simple reaction-diffusion patterns on a surface,
- ◇ the use of the concentrations of cell chemicals to change parameters of the rendered geometry, and

Figure 8.9: The top bear is fully combed, with all cells oriented like their neighbors. The bottom bear has patches of similarly-oriented cells. □

◇ the ability to restart simulations from an previous state with new cell programs, causing new behaviors to occur.

These cells are using reaction-diffusion equations similar to those in Table 1 to create patterns of chemicals in the cells. The diffusion of chemicals occurs by contact between cell membranes, thus it can only occur between adjacent cells.

Using a geometric modeler, we created a parameterized geometric object that changes from a bump to a thorn based on a single parameter [Snyder, 1992]. The particle converter sets this parameter to the value of a state variable representing the concentration of one of the reaction-diffusion chemicals.

We can see that there is a patch of cells on the front of the sphere with very little of the chemical (rendered as bumps), and a larger patch on the back with more of the chemical (rendered as thorns). In addition to the sharp boundary between the patches, note that the height of the thorns on the back patch varies continuously as they sweep around the sphere.

These simulations began where the earlier sphere simulation of Figure 8.6 left off, with new rules to cause the cells to differentiate. This is a common motif of user interaction with the system: halt a simulation, modify cell programs and parameters, and then continue simulating.

A Bear of a Surface In Figure 8.9, we show a fur-covered model of a bear defined as an isosurface of sampled volume data. We would like for the bear's fur to have a natural-looking orientation [Kajiya and Kay, 1989]. The bear on the left, with the fully combed fur, started from a single cell and used a set of rules similar to those used for Figure 8.6 to distribute and orient the cells. Each cell on the bear is rendered with a group of geometric objects meant to roughly represent a hunk of thick hair.

The bear on the right, with the patchy fur, was the result of a serendipitous combination of unintentional cell programs. Rather than having each cell align with all of its neighbors, each cell chooses one neighbor to align with. Also, cells do not attempt to align with neighbors that are oriented in the directly opposing direction. This bear started from about 2000 arbitrarily chosen cells on the surface.

Additional, more specific, orientation constraints could cause the fur to run more naturally down the limbs. Other cell programs could be added to cause the fur to be shorter in the region of the face and longer on the haunches, or to change the coloration based on the orientation or curvature of the bear's features.

8.7 Discussion

The combination of particle constraint techniques with developmental models enables the generation of a variety of cellular textures, as shown in the figures. We have found the approach to be a powerful method of creating attractive computer graphics models of organic objects. In our experience with making cellular textures, we encountered some difficulties, which we describe below. Some of these limitations are associated with our current implementation, and can be remedied without changing the basic framework. The problems with simulation speed and data explosion are less easily finessed, and will require further research to address fully.

Some commercially produced computer graphics films and videos contain models that have textures that appear similar to ours. The techniques used to generate them are generally proprietary and unpublished, hence we cannot definitively compare them with our work. Software for orienting fur on a CG character has been developed at Industrial Light & Magic [Duncan, 1994]. It is interesting to note that their discussion of the difficulties encountered closely parallels our own experience.

Shapes The spherical shapes of cells in a simulation generally are not the shapes we want to render, and so the particle converter might make objects with undesirable intersections. This can be minimized by a careful choice of cell geometry, but a more robust solution is to use the desired geometric shape directly in the cellular particle simulation. This would allow cell programs to calculate collisions based on more accurate geometry.

Experience with Writing Cell Programs Writing cell programs can be difficult. Programming independently moving cells by specifying differential equations has many desirable properties, but requires a different intuition than other types of programming, and often takes a while to get right. As with many tasks, it gets easier with practice. Here are some suggestions for using this programming paradigm:

- ◇ Copy and combine known cell programs from other researchers, such as surface or orientation constraints [Witkin and Heckbert, 1994, Szeliski and Tonnesen, 1992].
- ◇ Think about the constraints in the energy formulation (Section 8.4, and [Witkin et al., 1987]).
- ◇ Satisfy one constraint at a time; e.g., first get cell positions right, then modify other attributes.
- ◇ Force certain problem cells to be a certain way (through direct interaction, Section 8.4.1).
- ◇ Kill problem cells and regrow (Section 8.4.1).
- ◇ Apply artificial evolution [Sims, 1991b], and be patient.

Simulation Speed Simulations can be slow for some kinds of cell programs. We have some that run in seconds, and others, like the large datasets, that take many hours (e.g., the bear in Figure 8.9, and the head in Figure 8.1). Generally, performance degrades as the differential equations get stiff [Gear, 1971]. For some behaviors, clever cell programs like those described in [Witkin and Heckbert, 1994] avoid creating stiff differential equations, and so run faster.

Data Explosion The data produced both by the simulation and by the particle converter can get very large. We have partially addressed this by parameterizing the particle converter output by viewing distance (Figure 8.5). However, the simulation still has to compute enough cells to cover the surfaces, independent of viewing distance.

Future Work

There are several directions in which we would like to extend this work. First, we plan to continue extending and refining the cell programs to generate more complex cellular textures. We also are interested in running simulations on objects as they move and change shape. Modeling the motion of feathers on the wings of a flying bird, or hair on a running animal would be exciting. Initial experiments (not discussed in this paper) indicate that this will be feasible.

Implementing more sophisticated cell geometries in the particle simulator will give us more realistic placement of detail, and avoid self-intersections in the rendering. Finally, we would like to explore the possibilities of creating shapes directly from the fundamental interactions of the cells, without the surface constraint.

Acknowledgments

Many thanks to Erik Winfree for designing and implementing the *kd*-tree approximation, as well as for providing many helpful suggestions. We are grateful to Allen Corcorran, Matt Avalos, Cindy Ball, Dan Fain, Louise Foucher, Marcel Gavriliu, Barbara Meier, Mark Montague, Alf Mikula, Preston Pfarner, Ravi Ramamoorthi, Dian De Sha, and Denis Zorin for valuable discussions, support, code, and proofreading. MRI data was taken at the Huntington MRI center in Pasadena, CA.

This work was supported in part by grants from Apple, DEC, Hewlett Packard, and IBM. Additional support was provided by NSF (ASC-89-20219) as part of the NSF/ARPA STC for Computer Graphics and Scientific Visualization, by the DOE (DE-FG03-92ER25134) as part of the Center for Research in Computational Biology, the Beckman Foundation, and by the National Institute on Drug Abuse and the National Institute of Mental Health as part of the Human Brain Project. All opinions, findings, conclusions, or recommendations expressed in this document are those of the authors and do not necessarily reflect the views of the sponsoring agencies.

Chapter 9

Summary and Discussion

9.1 Summary

This thesis describes a general framework for developmental modeling which incorporates multiple developmental mechanisms. The framework is based on a conditional differential equation formulation for intracellular processes, combined with a piecewise-continuous formulation for describing cell behaviors and the extracellular environment. The basic model is described in Chapter 2.

Many developmental experiments are exhibited in Chapters 4 and 5, together with some general comments concerning how to regulate self-organizing multicellular structures.

The developmental model is applied to three domains in Chapter 6, Chapter 7, and Chapter 8. The contributions of each chapter are described in its introduction, and the conclusions are presented in the indicated sections:

- **Synthetic Biology:** to obtain intuitions about multicellular morphogenesis and pattern formation (Section 6.3),
- **Artificial Evolution of Neural Networks:** as a representation of neural networks (Section 7.5), and
- **Computer Graphics Texture Generation:** to generate geometric texture (Section 8.7).

Each of the applications chapters is mostly self-contained.

9.2 Discussions

Exploring the capabilities of this model has provided some insights in other areas, which we elaborate in the remainder of this chapter:

- **Lessons about multiple-mechanism development:** issues involving multiple-mechanism developmental models,
- **Lessons about genome programming:** hints on how to program in a genome-like language, and

- **Computer systems issues:** things to consider when writing developmental simulators (computational issues, databases, efficiency, software engineering, and numerical algorithms).

9.3 Lessons About Multiple-Mechanism Development

When making models, it is instructive to keep in mind the differences between the model and the real thing. Reflecting on these differences gives rise to many of the discussions in this chapter. Other discussions in this chapter arise from thinking about what patterns or processes were difficult to make with the developmental simulator.

This section contains:

- Interactions between Motion, Adhesion and Cell Shape (Section 9.3.1),
- Directional information, especially from cell contact (Section 9.3.2),
- How does a cell count its neighbors? (Section 9.3.3),
- Geometric constraints on organism shape from size and shape of individual cells (Section 9.3.4),
- Following the gradient of a self-emitted chemical (Section 9.3.5),
- Continuous cell-sheet models versus discrete cell models (Section 6.2.1).

9.3.1 Interactions Between Motion, Adhesion and Cell Shape

Many of the difficulties I encountered arose from interactions between motion, adhesion, and cell shape (e.g., Section 6.4). In real cells, these are closely related. Adhesion is often the main mechanism behind the motion and shape changes of a cell. By representing these as separate effects, we end up with more parameters than necessary, and we sometimes cause ourselves problems which real cells never experience. However, adding more detail to the representation of motion and adhesion requires adding more detail to other components of the representation as well, which may be undesirable.

Problems with the abstraction of spherical cell shape. Cells really aren't easily compressed or expanded, as we know from seeing what happens in tumors. They maintain their desired volume pretty well using osmotic forces. If cells are both **spherical** and **volume preserving**, then they are rigid, which makes the equations of motion stiff since cells are pushing and pulling on each other via their adhesive surface chemicals. One mechanism we implemented to ameliorate this was allowing cells to respond slightly to compression or expansion forces. Either choice is unrealistic and points out some of the limitations of the spherical model of cell shape. Different behaviors arising from using the compression forces can be seen in Figures 6.11 and 6.12.

Sketch of a more detailed cell shape model. To see some of the benefits of a more detailed shape model, we consider the cell membrane model used in [Glazier and Graner, 1993] to study differential adhesion. They use a discretized spatial representation (by analogy to a spin-lattice) and then anneal over time to locally minimize a Hamiltonian. A separate value is assigned to each lattice site indicating which cell it belongs to. Each cell also has an associated cell type, and the adhesive bonds between cells of different types are characterized by different surface energies. As the simulation progresses, the cell boundaries move and thus the cells exhibit motion and shape deformation which arise from lower level adhesive mechanisms. However, extra terms must be added to the system to keep cells at a reasonable size, and also to deal with cases when parts of a cell become disconnected.

How are cell shape and motion controlled in a more detailed model? In reality? If we adopt this 'lattice' model, some of our problems with interactions between motion, adhesion and shape disappear since they are all basically one phenomenon in this model. However, this model has a fairly weak representation of a cell as an entity. What happens if opposite sides of the cell want to move in different directions? It must either pull apart, or not change much due to the size of conservation terms. There is no mechanism for the cell to abandon one attachment in favor of another, as can occur in real cells. This decision-making capability is likely to be important in cell migration and pattern formation. And when integrating this into our multiple-mechanism approach, there are other factors to consider, like chemotaxis. How does this relate to the adhesion-based motion? If there is a tasty chemical gradient to the left, and an adhesive gradient leads to the right, which way do we go? Clearly there is some interaction between these phenomena, since the cell can't move towards the chemical gradient if adhesion is zero.

Thus adopting the 'lattice' model requires some changes to how the state equations control cell behavior, and to how the cell senses its condition. We may need some sensory apparatus to determine which directions the cell is being pulled, and then some behavioral function which enables the cell to choose one dominant direction. Real cells probably mediate such decisions by competitive mechanisms involving the construction of intracellular structures to support each lamellipodium. Perhaps the 'lattice' model will force us to model these intracellular structures somehow.

Lattice methods may not scale well to three dimensions. Despite these additional complications, we remain enthusiastic about the possibilities of using the 'lattice' model for cell shape in combination with a multiple-mechanism model. However, implementing the lattice methods in three dimensions will result in a huge computational burden, which is only partially ameliorated by the possibilities of parallel implementation.

9.3.2 Directional Information (Especially from Cell Contact)

Where does directional information come from? Real cells can get directional information from chemical gradients, contact with neighbors, gravity, electromagnetic field, light, heat, etc. Note: there are no coordinate system dependencies since these are all tensors (Section 3.2.2).

Currently, the simulated cells in my system can only get directional information from fields such as chemical gradients or light, and not from the position of a contact with a neighboring cell. Let us use the term *contact direction* to denote the relative position of a contacting cell. Note that if there are several adjacent cells, there will be several different directions which are of potential importance.

Why is 'contact direction' especially useful? Contact direction is difficult to implement (see below), and yet quite useful since it provides local direction (gradient is probably better for larger scale information). It is useful to:

- orient next cell cleavage properly,
- move along a row of cells while remaining in contact with them,
- choose direction to move for cell intercalation [Oster et al., 1990],
- follow an adhesive gradient,
- move towards something that 'tastes' good (remain in contact),
- move away from something that 'tastes' bad ("contact inhibition of movement" [1], [Edelstein-Keshet and Ermentrout, 1990]).

As always, it is important to keep in mind the differences between the model and the real thing. 'Contact direction' per se is probably not available to the cell. In fact, the mechanisms underlying some of the behaviors are intimately linked to the geometry of the situation, as discussed above (Section 9.3.1). For instance, cell movement due to pseudopods or lamellipodia depends on adhesion. If the pseudopod cannot attach to something, it retracts. If it can attach, it pulls. Thus a real cell can use information about contact direction at a very low level.

Why is 'contact direction' difficult to implement? Computation of contact direction is difficult to implement because it is unclear how to represent the multiple contacts that a cell has with its multiple neighbors. In real cells, specific sub-cellular structures can be created in regions of the membrane which are in contact with a neighboring cell [Gilbert, 1991]. The structures are then used to perform functions specific to that region (e.g., adhere, exchange chemicals).

One way to implement this is by extending the model to allow non-uniform distributions of surface chemicals. However, this requires choosing a representation for the distribution of surface chemicals, and providing a sensible means for the cell state equations to control the distribution via the cell behaviors. These choices are important, and non-obvious. An alternative is to maintain a list of contact regions, which are treated specially. This technique requires careful thought about which state equations will be operational for each contact, and whether the resulting system is still biologically relevant.

A workaround solution in the current simulator. In our current simulator we often use a chemical gradient in conjunction with a contact chemical to get the contact direction information. This is done by emitting a diffusable chemical from the target cell, which gives its neighbors the directional information. A contact chemical is then used to determine if the cells are actually touching. This combination works, but has some problems: (1) requires two mechanisms where one should suffice, and (2) if other nearby cells are emitting the same diffusable chemical, directional information is corrupted.

[1] *Contact inhibition of movement* can arise from the retraction of a lamellipodium upon encountering a neighboring cell. "A new lamellipodium is then formed elsewhere on the cell, thereby taking the cell away from its neighbor" [Gilbert, 1991, page 532].

9.3.3 How Does a Cell Count Its Neighbors?

Why would a cell want to count its neighbors?

- to make decisions, e.g., decide if it should divide, die or differentiate,
- to know if it is in center or edge of a sheet or ball of cells.

If a cell wants to know how many neighbors it has, it can integrate the amount of some chemical bound in its membrane (assuming that the chemical is bound to a complementary chemical on the neighbors). A problem with this is that it is hard to know what measurement means since it depends on (a) amount of membrane chemical on every adjacent cell, (b) contact area with each adjacent cell, and (c) number of cells in contact. Comparing this value to a threshold is unsatsifactory, since variations in many parameters can affect it. It's like adding uncertainty to the rules in Conway's 'Game of Life', probabilisticaly deciding whether to require two neighbors to live instead of three. Using the value in a continuous fashion to regulate cell decisions is more reasonable.

9.3.4 Geometric Constraints on Organism Shape From Size and Shape of Individual Cells

Figure 9.1 and Figure 9.2 show an example of the size and shape of cells determining the shape of the organism. The cells fit together snugly, constraining the organism to have a moderately rigid lattice shape. The orientations of cell divisions are controlled during development so that it forms a long chain rather than forming a grid of cells.

The cell shapes used in this simulation are simply circles, so changing the size and location of a cell is the only way to affect the total shape of the organism. In real organisms, cell shapes vary greatly, and yet the principle of using cell shape to geometrically determine the shape of a multicellular assembly still applies. For instance, long and narrow epithelial cells tend to form sheets with the long axes of the cells aligned (this maximizes the contact surface area, where they adhere). Of course, other mechanisms such as bones are also used to determine organism shape.

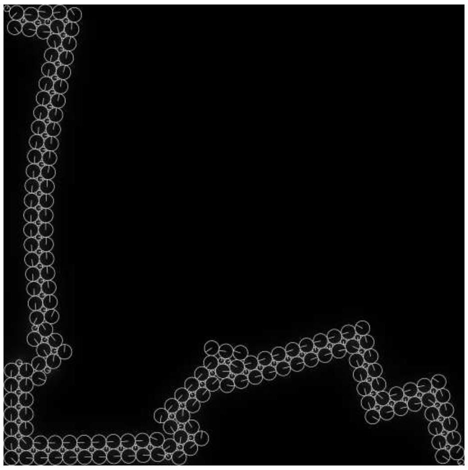
Detailed shape of each cell can affect global shape:

◇ hexagonal packing of near-circular cells (Figures 5.6 and 5.24),
◇ epithelial layers (formed of elongated cells), and
◇ cell shape change in gastrulation ([Odell et al., 1981]).

9.3.5 Following the Gradient of a Self-Emitted Chemical

Problems can arise when a cells follows/avoids a chemical that it emits. For instance, if a cell emits a chemical from one end and senses it from the other end, it might chase its own tail. This particular example is unlikely to happen in a biological system, but there are many biological examples where cells are sensitive to the concentration and gradients of a diffusing chemical which they also emit (for example, the cellular slime molds).

Cellular slime molds emit a diffusable chemical (CAMP) and also climb the concentration gradient to form a multicellular slug. The CAMP is emitted in pulses – perhaps partially to lessen the effects of the cells being confused by their own emissions. Experiments have shown that mutant slime molds which do not emit CAMP in pulses are still able to aggregate, so this is probably not the whole story. It may be a contributing factor, or it may be involved in the evolution of the pulsatile emission behavior.



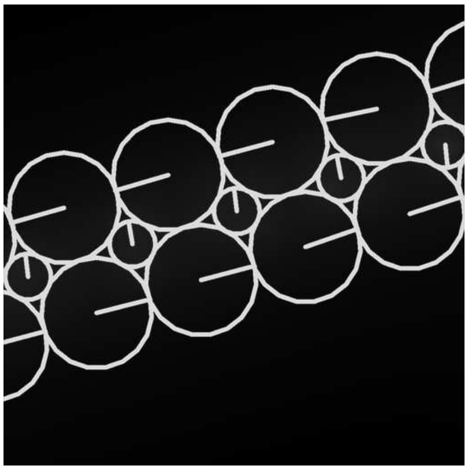


Figure 9.1: (Left) In this example, the shape of the organism is primarily determined by geometric constraints which depend on the size and shape of the constituent cells. □

Figure 9.2: (Right) In this closeup of Figure 9.1, note that the smaller cells fit in the interstices of the larger cells. Thus the two cell types of different sizes combine to form a rigid chain structure (in this `1-2-1-2` sequence).
Note: The line segment inside the cells indicates the orientation of the next cell cleavage. Proper orientation of cell division is critical for the propagation of this chain pattern. However, the cells in this closeup are post-mitotic, so their orientation is no longer important. □

9.4 Lessons About Genome Programming

- Size regulation
- Experience writing cell state equations

9.4.1 Size Regulation (Total Colony Size/Shape)

Size regulation is a difficult problem for multicellular organisms. We separate this into two related problems: limiting growth, and clustering (keeping the cells together). Each of these may be solved using chemical, mechanical or genetic operations. For limiting growth, we tried several methods:

- ◇ using a threshold on the concentration of a diffusable chemical to turn cell division on and off,
- ◇ dividing only in the presence or absence of a neighboring cell, as determined by surface chemicals (e.g., contact inhibition of growth),
- ◇ an equilibrium between cell birth and death rates,
- ◇ amount of innervation (Section 5.3.3), and
- ◇ using cytoplasmically inherited factors. Cells divide until the supply of an irreplaceable chemical is exhausted. Since the chemical is not regenerated, each cell division reduces the total amount per cell, and as the cells grow in size, the concentration diminishes.

For clustering, we also tried two methods:

- ◇ cells move up the concentration gradient of a diffusable chemical, and
- ◇ using cell adhesion to keep cells together.

The most robust size regulation behavior was produced using a combination of approaches.

9.4.2 Experience Writing Cell State Equations

In the process of writing state equations for many different experiments, I developed some ways of combining the conditions and differential equation fragments that seemed particularly useful, and were re-used from experiment to experiment. From a mathematical point of view, there is nothing too surprising in the following equations. But from a systems point of view, they help to give some impressions of what it is like to control a system which is governed by conditional differential equations. Insomuch as real cell processes are similar to these conditional differential equations, they may provide some insights.

Rate/setpoint equations. One way to use the cell state equations is to choose a rate r and a setpoint s, as in the following equation for a state variable z:

$$\text{if cond then} \quad dz/dt \quad += \quad r(s - z) \tag{9.1}$$

Equation 9.1: Rate/setpoint style of cell state equation. □

Then the variable z will approach s at a rate determined by r. If we combine two of these equations for one state variable, then we get:

$$\begin{array}{lll} \text{if cond}_1 \text{ then} & dz/dt & += \quad r_1(s - z) \\ \text{if cond}_2 \text{ then} & dz/dt & += \quad r_2(s - z) \end{array} \tag{9.2}$$

Equation 9.2: Combining rate/setpoint equations. □

If both cond_1 and cond_2 are true, then this leads to setpoint at

$$z = \frac{r_1 s_1 + r_2 s_2}{r_1 + r_2}.$$

Otherwise, if just cond1 is true, then

$$z = s_1.$$

Thus equations formulated in this style combine such that z will approach a value between s_1 and s_2. If the conditions are exclusive, then it will go the appropriate setpoint.

Separated equations. Another way to formulate cell state equations is to separate out the term which causes the state variable z to decay. Then each equation with a positive contributing term adds to the setpoint of the state variable. In the following equations, we see two such contributing statements each with its contribution s_1 and s_2. If only cond_1 is true, then z approaches a setpoint at s_1, and likewise for cond_2. If both conditions are true, then the resulting setpoint is $s_1 + s_2$. This behavior is useful for a simple model of gene expression, since expressing two genes for the same protein makes twice as much (assuming raw materials are available, etc.)

$$\begin{array}{lll} \text{if true then } dz/dt & += & -z \\ \text{if cond}_1 \text{ then } dz/dt & += & s_1 \\ \text{if cond}_2 \text{ then } dz/dt & += & s_2 \end{array} \tag{9.3}$$

Equation 9.3: 'Separated equations' style of cell state equation. □

Accumulation equations. This is useful for controlling event behaviors like cell death (Section 4.2.2). Assume that chemical z_{die} causes death if it gets over some threshold. For some small α,

$$\begin{aligned} &\text{if cond}_1 \text{ then } dz_{die}/dt \quad += \quad \alpha \\ &\text{if cond}_2 \text{ then } dz_{die}/dt \quad += \quad -z_{die} \end{aligned} \tag{9.4}$$

Equation 9.4: 'Accumulation equations' style of cell state equation. □

When cond_1 is true, the cell accumulates the chemical which eventually causes death. But, if cond_2 occurs, the state will quickly get reset (z_{die} will drop to near zero), and the cell gets a reprieve.

Another motif using accumulation equations is to have another condition that cancels the contribution from cond_1, but leaves the accumulated value in the variable:

$$\text{if cond}_3 \text{ then } dz_{die}/dt \quad += \quad -\alpha \tag{9.5}$$

Equation 9.5: Another accumulation equation. □

9.4.3 Surface Chemical Normalizer

Here we present a method for a cell to compute the average amount of a surface chemical bound per unit area of contact.

The model of surface (membrane) chemicals discussed in Section 3.2.4 allows a cell to know the total amount of its surface chemicals which are bound. Using just one surface chemical, a cell cannot tell the difference between being in contact with

1. two cells,
2. one cell which is twice as big, or
3. one cell with twice the concentration of surface chemical.

Often it is useful to be able to distinguish between these cases. We can distinguish between cases 1 and 3 by using another surface chemical as a normalizer.

Assume we have two surface chemicals a and b, each of which bind homophilically (like binds to like). And further assume that b (the normalizer) is exhibited on the surface of all cells in the same concentration. Then the value of env[b] (the amount of b which is bound on this cell) is proportional to the total contact area. Using this as a normalizer, $\text{env}[a]/\text{env}[b]$ gives us the average amount bound per unit area of contact on the cell.

For n cells with the same surface chemical concentrations of a, this gives us $(n\,\text{env}[a])/(n\,\text{env}[b]) = \text{env}[a]/\text{env}[b]$. This is true even if the cells are different sizes. If the cells are the same size, but the concentrations of a differ, then this computes the average amount bound per unit area of contact.

This technique is used in the simulation in Section 6.2 and Appendix A.

9.5 Computer Systems Issues: Lessons About Building Developmental Simulators

This section contains suggestions for people who are implementing large developmental simulators. Some of these are things that we have tried and found to be useful. Others are approaches that we did not use, but found ourselves wishing that we had.

9.5.1 Useful Techniques for Developmental Simulation

During the construction of the simulator, we focused on getting qualitatively biological behavior while aiming for computational efficiency. The following techniques were effective in achieving this balance:

- ◇ cell state equations: (Section 2.5 and Section 2.5).
  - ◇ use condition to allow regulation of genes and groups of genes .
  - ◇ compute condition as a continuous function (avoid a discontinuity at the firing of every condition).
  - ◇ sum the contribution of multiple artificial genes.
- ◇ use simplest stable solver (adaptive Euler solver, Section 3.3).
- ◇ use viscous dynamics ($F = kv$) for cell motion (Section 3.2.3).
- ◇ use penalty method for sloppy collisions (Section 3.2.4).

Noise can play an important role in dynamical systems, knocking a system off of an unstable point or popping it out of a local minimum (Section 3.3.2). It also can be used as part of a stochastic estimation process. For instance, some bacteria move up a nutrient gradient by the strategy of moving randomly in various directions, but with a smaller likelihood of changing direction if there seems to be more nutrient. We incorporate noise by adding it in at the cell's sensors (user can specify the amount of noise).

9.5.2 Verifiability of Simulations

In a simulation with variable time steps and pseudo-randomness and so many mechanisms, it is difficult to be certain that things are implemented correctly. Each time a bug is fixed or a feature is added, the behavior might change slightly. Is it a bug or not? Maybe the new code changed the calling sequence of some numeric routines, and the differences are due to numerical precision, which might get amplified by the 'butterfly effect'.

Once the simulations start to drift apart, the behaviors can be very different. Since there are discontinuous events, a small numerical difference can eventually lead to a very different outcome. Thus small changes can lead to bifurcations in the system behavior. This can be a problem even just when comparing the results on different computers, or between different operating systems on the same computer (we have seen this).

Yet we don't want to spend hours trying to verify the code each time a change is made. An approach for dealing with this problem is to to adopt some software engineering procedures from industry, and run 'unit tests' on the code. This means taking the time to write some good tests which compare the simulation results to some known correct result.

The tests need be simple enough that they don't cause a major bifurcation based on small numerical differences. Also, because the solutions are numerical, they must be compared using some tolerances.

We used some of our existing experiments as tests, and compared the results of earlier simulation runs with later ones to ensure that later versions of the simulator were running properly. In some cases, we compared the results by eye rather than numerically since we knew changes to the simulator had affected the small-scale behavior of some aspect. However, the large-scale behavior should remain the same, and this proved to be a useful debugging aid.

We believe it would be preferable to set up some simple tests for which we know an analytic solution. Then we could compare the computed simulation to the analytic solution. We have not done this except for extremely limited cases.

Having a set of tests which can be run automatically and come back with a binary answer is well worthwhile (even if they take overnight to run). Although we built a few such tests, having many more would be valuable.

9.5.3 Saving Information About Experiments

Dump files to record the state of the simulation are essential. It should be possible to restart from them as if nothing happened. We implemented a dump and load facility that allowed this for our adaptive Euler solver. However, with various types of numerical solvers, this can be difficult. Some solvers do not let you know enough of their internal state so that you can reload.

It would be handy to have a database, or a least a bookkeeping system to keep track of all the experiments, including: which system version they were run with, where the dump files are, what the parameters were, what the intent of the experiment was, where the images are, and the processed images, and the images with captions, etc.

Unfortunately, we do not currently have such a database, and the information is stored by keeping related files in a single directory. In the course of running many similar experiments, this sometimes led to a confusing trail of modifications; thus we believe the database approach might be valuable in the future.

9.5.4 Maintaining Backwards Compatability: Moving Forward Without Losing The Past

As new features are added to the simulator or bugs are fixed, sometimes old experiments don't work any more. They depended on the old features, or (worse) on the bugs. This is really a pain. But to fix the old experiments can take a lot of time in some cases (if it changes the behavior near a sensitive parameter setting). Yet to have them no longer work is terribly inconvenient.

A solution to this is to provide a flag each time a major change is made, and use the flag setting to provide backward-compatability. Version numbers on each dump file can then identify which flag settings are needed. We implemented a simple version of this, and it was quite helpful.

However, this approach leads to some amount of messiness in the code, since so much of the old functionality is retained. After a few years of this, the code becomes more and more cumbersome to work with, and it is hard to keep each branch working with a new change. At some point it becomes worthwhile to clean house, removing branches of the code that are clearly undesirable (e.g., if we discovered a better implementation). Then we must bring old experiments up to date. This sort of large-scale change was made three times on the simulator since its inception.

9.5.5 The Value of a Math Interpreter (and Dynamic Compilation)

An interpreter of mathematical expressions is an essential tool for a simulator. It enables users to enter expressions instead of just values. We use a C interpreter written by John Snyder to evaluate small expressions on the fly [Snyder, 1992]. To gain speed in the execution of large simulations, we write out a file of C++ code, compile it, and dynamically link it into our running executable. Then we can call compiled routines to evaluate the cell state equations. [2]

Uses of the math interpreter:

◇ to specify the equations to be solved (in this case, the artificial genome), and

◇ in a graphical (or other) analysis tool, to specify the values to be displayed as a function of the simulation variables (see Section 9.5.6 below).

9.5.6 Graphical and Other Analysis Tools

It is well known in the field of simulation and modeling that analysis tools are of great value not only to the user but also to the programmer. The system described in this thesis contains many tools for graphical display, so that the display can reflect the state within each cell that the user finds important.

Graphical tools enable the immediate visualization of the state of the system. They are incredibly valuable for evaluating the state of complicated multivariate systems like collections of hundreds of cells. Using color or icon size to indicate the value of a particular key variable in a cell makes it clear what is happening inside of many cells simultaneously. Examples:

- the amount of inhibition is shown in Figures 5.7 and 5.6,
- the cell type (reflecting internal state of the cell) is shown by the color of the cells in numerous examples from Chapter 5,
- the local coordinate frame of each 3D cell is shown in Figure 5.25.

In the system, the user can indicate how the color or various icons should appear within a cell as a function of the state variables of the cell. For instance, the color [3] of the cell can be set to $[z_i, 1.0, 1.0]$ (in red-green-blue color space) which will make the cell appear white if $(z_i = 1)$ or cyan if $(z_i = 0)$. For values in between, intermediate colors are displayed (values outside the range are clipped to $[0, 1]$). Any valid C expression of the variables can be used, since the expressions are interpreted or compiled. This underscores the value of a mathematical interpreter of some sort as an essential tool for a simulator.

It is also be useful to have an easy way to get a plot of *any* variable or combination of variables versus simulator time. I did not implement a general tool for this, and that was a mistake. Instead, for debugging or examining particular items, I wrote special purpose code to dump the traces. I advise other implementors to incorporate a very general facility for dumping traces over time of variables and functions of variables. I emphasized "any" variable above; this includes variables involved in the internals of the solver (e.g., the stepsize in an adaptive solver) or logging variables (e.g., how

[2] If you are thinking of writing a simulator like this, you might wish to use LISP. The current LISP environments are supported on most workstations, and it would be more convenient to do the dynamic compilation, and many other things, in a LISP environment.

[3] Color is represented here as three numbers to specify red, green, and blue, each between 0.0 and 1.0.

many times has the PODE event function for cell splitting been called). It is also tremendously useful to be able to easily plot functions of the variables (e.g., the volume of a cell), or functions of an array of variables (e.g., the average of all cell volumes).

9.5.7 Units and Non-dimensionalization

Non-dimensionalization is clearly a useful technique for mathematical analysis, and we non-dimensionalized the equations in the system. For this programmable simulator, though, it caused some headaches.

What units are in the user-specified cell state equations? The user is probably most familiar with dimensional parameters, but we have already non-dimensionalized things by that point. Even if the parameters are non-dimensional, the user would like to know what the parameters mean.

There are a few problems:

◇ the variables can have different interpretations (hence different units),
◇ if the given parameters (e.g., chemical gradient measured locally) are specified in non-dimensional units, they may be less clear to the user,
◇ user must specify non-dimensional parameters.

Proposal (unimplemented). Units are critically important, and it should be possible to incorporate them more fully into the system, so that every number has a unit attached to it at all times. Even if a number is a non-dimensional parameter, it should have an interpretation, e.g., “ratio of this cell’s radius to minimum cell radius.” With this information, we can then present the user with a meaning for every number.

This would also aid the programmer in keeping the code error-free, since he/she will always have the meaning of a variable available. Units can be checked semi-automatically, and assertions can be made that variables have appropriate units.

One way to keep units around is to make every number or array into an object (e.g., a `C++` object). If one were careful with this approach, it is probably possible to do this without undue computational overhead. This generalization needs to be incorporated from the project’s inception, since it is very difficult to replace all existing numbers and arrays with a different class once there are thousands of lines of code.

This idea would result in some extra programming time, but is likely to be an overall time-saver due to having reduced coding errors. The user interface would be improved, too.

9.5.8 Efficiency and Numerical Algorithms

It is difficult to determine what the bottleneck is for system speed. I have spent a lot of time speeding things up, and the speedups sometimes come from unlikely places. At least they seem unlikely until you find them.

Here is a list of places to look for speedups. Most of these are fairly generic, and they give an idea of the range of issues which arise in designing and implementing any large simulator. I have found several speedups by examining these areas:

◇ stiffness of the differential equations (which can be caused by parameter values typed in by the user),

- ◇ small step sizes by the solver (one cause: diffusion grid spacing determines a minimum step size for stability of Euler's method),
- ◇ solver tolerances tighter than necessary (they should be as lenient as possible to get the desired behaviors; Determine this by trial and error),
- ◇ display (if sending commands to an X client, or graphics hardware, be sure buffering is working properly and that the display is not operating on each element independently),
- ◇ memory size and fragmentation,
- ◇ inappropriate data structures. [4]

9.6 Evolution, Optimization, and Genetic Algorithms

There has been much discussion of the relationship between evolution and optimization [Buss, 1987, Maynard-Smith, 1986, Dawkins, 1976, Dawkins, 1982]. In my opinion, most of the controversy is due to overstatements and willful misunderstandings. Perhaps much of this could be avoided by using the word 'adaptation' in place of 'optimization.'

In a literal sense, evolution and optimization can never be reconciled since an optimization problem is a mathematical description of 'the best answer' (e.g., find x such that $f(x)$ is maximal), and evolution is a phenomenological description of a natural process. There is no inherent problem to be solved in evolution, and no clear choice of definition for x or f.

However, populations of organisms do appear to exhibit progressive adaptation to their environments under natural selection. And evolution has produced some wonderful natural computers. So it is reasonable to examine how the process works, and to consider emulating it as a model of how to achieve 'improved' results, if not optimal ones.

Evolution operates under some severe constraints. It is constrained to produce a viable organism in every generation. It cannot traverse severe local minima directly, since dead organisms cannot reproduce.

Another constraint arises from the reproduction process. Each new genome is a combination and modification of existing genomes. It may be produced via recombination, cross-over, random mutation, or other processes, but it is not created *de novo*.

These two constraints distinguish evolution from global optimization. However, evolution is similar to local constrained optimization in the sense that organisms which have a better ability to survive and reproduce will dominate the population. If the genetic makeup of a particular organism in its environment enables it to reproduce successfully, then that genome will stick around. Of course, in evolution the 'objective function' is a continually changing, discontinuous, mathematically monstrous function.

In this spirit, we consider genetic algorithms as an adaptive search method inspired by evolution. It is valuable to consider genetic algorithms in contrast to standard optimization methods, in cases where the problem is presented in a form that both approaches can be applied.

[4]This can easily occur in auxiliary code that was written without thinking about how it might be used. With the large numbers of variables used by this simulator, this was sometimes a problem. For instance, some numerics code which I use keeps lists of things which later it searches for particular IDs (thus it is $O(n)$ for each lookup). A hash table or binary tree would make this much faster for long lists.

Chapter 10

Future Work

- Create predictive models of specific biological systems.
- Create neural networks by applying artificial evolution to developmental models.
- Examine theoretical questions in evolution and development.
- Miscellany.

10.1 Predictive Biological Models

This thesis has presented a multiple-mechanism developmental model that can represent a variety of biological phenomena in the abstract. The next step is to apply the model to a particular biological system.

One biological system that we have considered is development of *Drosophila* retina, in particular the progression of the morphogenetic furrow. This system involves cell interactions via membrane chemicals and diffusing chemicals, as well as cell division, cell death, and cell differentiation. There are a substantial number of genes whose product and function have been identified or postulated. However, there are still many unanswered questions about the propagation of the morphogenetic furrow and the differentiation of cells in the ommatidia.

This might be a fruitful system for investigation using a computer model, since there are so many interacting factors. The wealth of experimental data provides checks to test the model's validity (it should be able to reproduce the appropriate loss-of-function experiments).

Other systems we are considering include the formation of boundaries between segments in invertebrates, *Drosophila* neuroblast differentiation, insect bristle formation, frog neural plate development, etc.

This work is best carried out in close collaboration with a biological laboratory; the particular model chosen will certainly depend in part on the laboratory in which the work takes place.

10.2 Artificial Evolution of Neural Networks

We remain optimistic that our conjecture concerning the evolution of developmental systems is true: the properties of robustness and developmental gain are helpful for evolution. Perhaps the system which we devised to test this was too ambitious, although the less complete systems that other researchers use for testing evolutionary hypotheses generally seem to us to be oversimplified to the point where they do not really teach us much (Section 7.2).

The following strategy embodies our experience as well as recommendations from the literature. We believe that these components may allow us to reach our goal of evolving problem-specific neural networks.

- The initial population should be fairly capable, containing many useful building blocks (e.g., how to make connections).
- The genetic algorithm should mostly perform recombination as opposed to mutation (as suggested by [Koza, 1992, Holland, 1992]).
- The parameters to the developmental model need to remain within reasonable ranges (by restricting the search space, or by imposing constraints within the developmental model).
- Use a progressive objective function.

The problem of finding appropriate parameter settings for the genetic algorithm is one with which we have little experience, and it seems the most daunting. Parameters to the algorithm include things like choosing the rates for various types of mutation and recombination, as well as choosing the method of selecting which individuals from one generation will reproduce into the next generation.

10.3 Theoretical Questions in Evolution and Development

The ongoing discussions of several theoretical issues in evolution and development might benefit from the use of a general simulation tool like this one. The issues include:

◇ generation of global structures from local rules
◇ the structure of fitness landscapes
◇ robustness of morphogenesis [Goodwin et al., 1993]
◇ other hypotheses espoused in [Mittenthal and Baskin, 1992, Goodwin, 1994].

Several researchers are considering theoretical questions concerning the nature of development and evolution, and their relationships. We believe that our flexible developmental model might be helpful to address some of these questions.

A framework for studying the generation of structure from local rules. One of the fascinating features of morphogenesis is the reliable creation of global structures from local rules. Even though the structures may vary in their fine detail, the overall character is the same: every frog is a frog, despite slight differences at a fine scale. The simulation framework described in this thesis enables a computational investigation of these phenomena.

Experiments such the two hierarchical networks shown in Figures 5.21 and 5.22 provide a starting point for exploring this type of phenomenon. The overall structure in both networks is a two-level hierarchy (white-green-red). Yet the detailed locations of cells and connections is clearly quite different between the networks.

Further experimentation with systems like this may help to address questions such as the following: How does a genome robustly create structures that are behaviorally or structurally similar while the fine-scale details are different?

10.3.1 Self-Organization Versus Morphogenesis

A note on self-organization versus morphogenesis. We distinguish between organizing from an arbitrary state versus developing from a restricted set of states into the final configuration.

We characterize a distinction between a system which **generates** a particular shape from some set of initial states, and a system which **organizes into** a particular shape from a perturbed state. For example, if you separate out the cells of an organism, will they recombine to form the organism? This is harder than generating the shape initially.

A system which can self-organize *from any initial state* is depicted in Figure 10.1 (a). An example of this is a set of cells exhibiting differential adhesion Section 6.4, which can reassemble into a similar state from any starting state where the cells are in contact. One can stir the cells until they are evenly mixed, and they will then regroup [Armstrong, 1989]. In early stages of development, the embryos of many organisms can recover from fairly severe perturbations (removing a cell or changing the locations of cells [Gilbert, 1991]).

On the other hand, many developmental processes depend on the progressive nature of development (Figure 10.1 (b)). That is, later stages depend on the success of earlier stages. This is true of most organisms, especially in the later stages of development – if you mix up the cells of an adult mouse, the mouse is unable to reassemble. This is somewhat easier than the more general case of self-organization, since there is a smaller state space to consider.

This may have some import for considering the evolution of developmental systems.

10.3.2 Robustness in Morphogenesis

[Goodwin et al., 1993] suggest that morphogenesis is inherently robust due to its dependence on multiple mechanisms. They claim that the basin of attraction of a multi-mechanism parameter space is larger than otherwise.

- They could be right, but it is a bit hard to believe that 'random' combinations of mechanisms would give rise to coherent patterns.
- Their conjecture is difficult to prove/disprove – how do you measure these things?
- It seems likely that counter-examples exist, and we could probably create one with our system. Clearly there are situations where different mechanisms are used to create different elements of a pattern, and both are essential and independent, or even worse, they depend on each other to unfold properly.
- A weaker claim: it is likely that evolution would make use of combinations of mechanisms that were robust, especially in early development. So there would be a tendency to favor robust combinations.

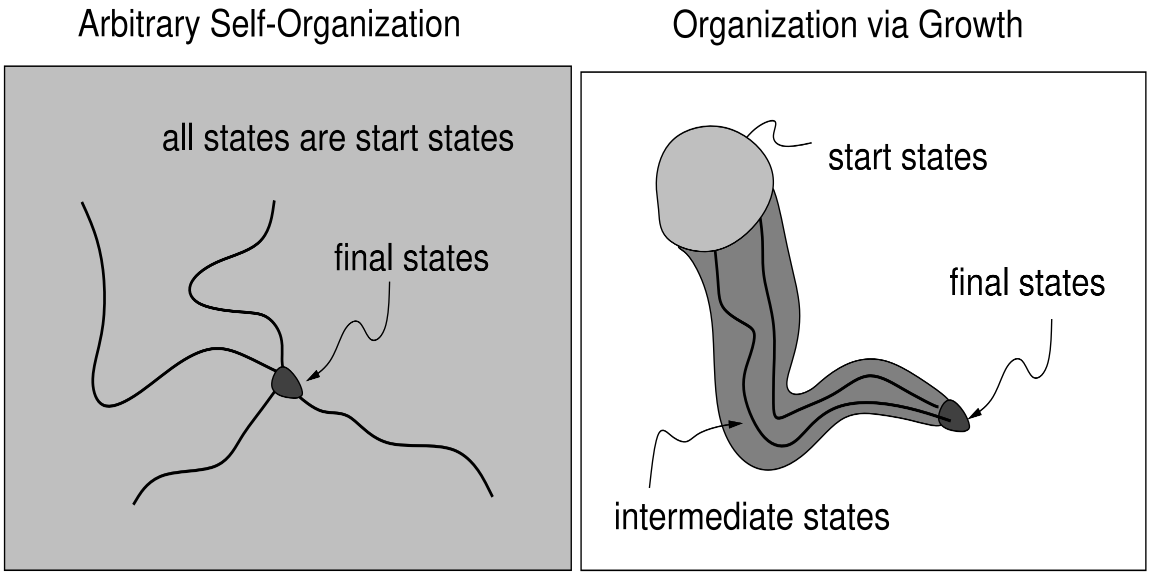


Figure 10.1: This figure illustrates the distinction between two types of self-organization. We are looking at a 2d state space where initial states are colored light gray, and the final "organized" states are colored dark gray. Thick black lines indicate possible paths through state space during the self-organization process.
(a) A system which can self-organize from an arbitrary state.
(b) A system which self-organizes only from a restricted set of start states, and which must pass through particular intermediate states to reach the final state. □

- We might be able to make some investigations with the simulator. For example, we could devise different methods of forming a circle of cells (in 2D) using different combinations of mechanisms (diffusion alone, diffusion and cell-adhesion, mechanical factors and diffusion, etc.). Then compare sizes of parameter space variation which produce something vaguely 'circular' (thanks to Erik Winfree for valuable discussions on this topic).

10.4 Miscellany

10.4.1 Tensor Implementation of Cell Local Coordinates

One approach for implementing a local coordinate system for each cell is described in Section 3.2.2. Here, we present a different proposal using tensor notation. This is not in our current system, but it looks attractive for future implementation, especially for artificial evolution applications. In general, allowing tensor notation for the cell state equations would be quite convenient.

Let's consider again the generation of a motile force as described in Sections 3.2.2 and 2.7. The direction of the motile force depends on either a tensor which comes from an external measurement, or from a local tensor (current heading). We can express the force as a tensor transform:

$$\bar{f} = \text{motile force} \tag{10.1}$$

$$= \bar{M}(\alpha_\theta, s_\theta)\bar{\theta} + \sum_{i \in \text{sensors}} \bar{M}(\alpha_i, s_i)\bar{c}_i, \tag{10.2}$$

where $\bar{M}(\alpha, s)$ is a tensor transform that rotates by angle α and scales by s.

The first term $\bar{M}(\alpha_\theta, s_\theta)\bar{\theta}$ enables dead-reckoning of the cell, using its internal heading $\bar{\theta}$. The cell's heading, $\bar{\theta}$, is another state variable (or set of state variables, for three dimensions) which can change over time due to an applied force in the cell.

The second term is a sum of $\bar{M}(\alpha_i, s_i)\bar{c}_i$ terms which take into account the attraction or repulsion of sources $\bar{c}$. For a chemical gradient, $\bar{c}$ will be the gradient of a chemical field ∇C.

Note that this scheme actually allows two ways for a cell to evolve the ability to move up a gradient. One is simply to have $\bar{M}(\alpha, s) = 1$ for some chemical and $\bar{M}(\alpha_\theta, s_\theta) = 0$. This corresponds to simply moving up the gradient independent of the cell's current orientation.

Another (less biologically plausible) method of following a gradient is to have the cell rotate its internal heading $\bar{\theta}$ until it is aligned with the gradient, and then move forward using $\bar{M}(\alpha_\theta, s_\theta) = 1$.

10.4.2 Thoughts on neurite growth

We occasionally had problems with things like growth cones adhering to or being attracted to the parent neuron instead of heading for their target, especially if the target neurons were similar to the parent. Real neurons may avoid some of these problems because the neurite construction process involves adding new structures which tend to push the growth cone forward. A better model of growth cone guidance might be "go forward until something causes a turn." There is also some evidence that the speed of neurite extension might be controlled somewhat by the soma. So maybe the growth cone steers, but the soma pushes. That would also be an interesting model to explore.

10.4.3 Cell Metabolism and High-Order Differential Equations

High-order systems of differential equations arise naturally from the combination of many first-order equations. Since the equations describing chemical interactions are often strongly coupled, the activities of real cells are probably described by high-order systems of equations.

Such systems are notoriously difficult to control. A topic for study is to consider how real cells control high-order differential equations created by many interacting chemicals. This is a topic which can be investigated within our simulation framework.

Glossary

cell model A cell is modeled as a geometric shape (currently a circle, with optional neurites) with a given response to applied forces, as well as an array of *cell state variables*.

cell behavior functions (protein structure-to-function model) The cell's current state determines what it is trying to do: the forces it is applying, events such as cell division, etc. The behavior functions compute all of the cell's forces and events from the state variables. Thus if the state variables represent protein concentrations, these functions describe the activities these proteins take in the cell (causing motion, division, etc.) Section 2.3.

cell state equations (the genome, metabolism, etc.) The state equations modify the cell's internal state variables based on the local environment and the cell's current state (Section 2.3).

cell state variables ($\mathbf{state}_c[\,]$) An array of variables which loosely represent the amounts of proteins within the cell (or differences, as noted above). The values of these variables affect the cell's movements, the timing of events, and the cell's interaction with the environment.

continuous cell behaviors Cells exhibit several continuous behaviors, determined by the cell behavior functions (Section 2.7):

- ◇ attempt to move in some direction (may be limited by collisions, adhesion, or drag)
- ◇ attempt to grow in size
- ◇ emit or absorb chemicals from the environment
- ◇ change amount of particular proteins in the membrane (eg. cell adhesion proteins, which mediate how much this cell will adhere to another cell)

determination The commitment of a cell to a given fate. This may occur before the cell begins to exhibit many of the aspects of the type into which it will differentiate.

differentiation "The generation of cellular diversity." [Gilbert, 1991, page 4] "The development of specialized cell types." [Gilbert, 1991, page 250.]

discontinuous cell behaviors (events) The cell provides functions which determine the timing of the following events. An event is a discontinuity in the solution, which stops the solver and may create or destroy data structures. The event occurs when the corresponding cell behavior function crosses zero (Section 2.7).

- ◇ split (cell division)
- ◇ die
- ◇ emit neurite with growth cone

equations of motion Given the cell's behavior functions which describe what the cell is *trying* to do, these equations compute the end results using viscous dynamics. For example, the cell may be applying forces to move to the right, but the collision forces may counteract that movement, producing a net movement to the left or right (Section 2.3).

environment All of the simulated cells interact within a single global environment. The environment contains diffusing, reacting chemicals, as well as physical barriers. Within the simulation, cells access information about their environment locally through an array of *local environment variables*.

homophilic binding Like binds to like.

heterophilic binding Two complementary molecules bind together.

ligand A *ligand* is a molecule that binds to another molecule. In this thesis, we are usually referring to a membrane ligand (one which lies in the membrane of a cell). The ligand and receptor are a pair of complementary molecules. If they are expressed on the membranes of adjacent cells, the ligand and receptor bind together.

local environment variables ($\mathbf{env}_c$[ ]) An array of variables which represent the local environment of a cell. The values are available to the cell as a function of time, and they depend on the extracellular environment. Since each cell is in a different location, in general the local environments of two cells will differ. These variations can then lead to different behavior for the cells, even though their genomes may be identical.

morphogenesis A process that organizes differentiated cells into multicellular arrangements such as tissues and organs. The creation of form and structure. [Gilbert, 1991, pages 9 and 523.]

pattern formation "The activity by which embryonic cells form ordered spatial arrangements of differentiated tissues." [Gilbert, 1991, page 607.]

Appendix A

Experiment Files

A.1 Guide to the Experiment File Format and Commands

The simulator uses a command line interpreter (CLI) [Kay, 1988]. The commands to describe each experiment are generally kept in a single file which then serves as a record of the experiment.

SOME CLI SYNTAX & COMMANDS:

```
help <command>
apropos <string>
read  <filename.cli>

Commands like the following appear within .cli files:

cinterp "<stuff>"
  define values and routines for cinterp, which can be used in the
  statements of the differential equations

=i varname value      Set an integer variable
=i varname            Show an integer variable
=d varname value      Set a double variable
=d varname            Show a double variable
=s varname "value"    Set a string variable

setgcprogrami     begin making growthcone program
setprogrami       begin making cell program

addstmt <condition> <index> <consequent>
   defines a differential eqn to be executed in the cell.
   if (condition) then damount[index]/dt += consequent
```

```
addcell <cellname> <value> <value> ... <value> geom <position> <size> <orientation>
  add a cell. the number of values is the number of state variables ('n' in
  the table below). the string 'geom' separates state variable initial values
  from the geometry initial values. Position and orientation are may
  be several values depending on whether theDimension is two or three.
```

A.2 Inhibition via Surface Chemicals

This is discussed in Section 6.2 and Section 5.2.3. An image generated from it is in Figure 5.6.

```
# If not differentiated, grow and split until the clock passes 'stop_thresh'.
addstmt "(z[stopsplitting] > stop_thresh) && (z[local+neuroblast] < 4.0)"
      split "(3.2 - z[split])*0.2"
addstmt "(z[stopsplitting] > stop_thresh) && (z[local+neuroblast] < 4.0)"
      fsize "(3.0 - env[radius]) * 0.1"
addstmt "(z[stopsplitting] < stop_thresh) || (z[local+neuroblast] > 4.0)"
      fsize "(2.5 - env[radius]) * 0.1"

# Wiggle slightly
addstmt "(1)" fx "(0.5 - drand48()) * 1.0"
addstmt "(1)" fy "(0.5 - drand48()) * 1.0"

# The clock.
addstmt "(z[stopsplitting] > 0.0)" stopsplitting "-1.0"

# Be sticky. This is also used to normalize the value from the receptors.
addstmt "(1)" "surf+autostick" "1.0 - z[surf+autostick]"

# Use contact chem to inhibit neighbors.
addstmt "(1)" "surf+ligand" "-ligand_prod_rate*z[surf+ligand]"

# This equation causes the cell to differentiate
# max(-z[i], ...) constrains it to be > 0
addstmt "(z[local+stopsplitting] < (stop_thresh-10))"
      "local+neuroblast"
      "max(-z[local+neuroblast],
         1.0*(-20*(env[contact+receptor]/env[contact+autostick]) +
                10*(z[local+neuroblast]*z[local+neuroblast]) /
                 (1 + z[local+neuroblast]*z[local+neuroblast])
            - 0.5*z[local+neuroblast]) + 13)"
```

```
# If I'm a neuroblast, inhibit a lot
addstmt "(z[local+neuroblast] > 3.0)"
      "surf+ligand" "ligand_prod_rate*(z[neuroblast])"

# else, inhibit about 95% of the inhibition I'm seeing.
# Add some noise here to avoid getting stuck in unstable equilibria.
addstmt "(1)" "surf+ligand"
      "ligand_prod_rate*max(0.0,
       min(1.0, 0.95 *
          (env[contact+receptor]/env[contact+autostick]
           + 0.1*(0.5 - drand48()))))"

# Have enough receptors to see all of the inhibition!
addstmt "(1)" "surf+receptor" "5.0 + -z[surf+receptor]"

# Filled dark gray circle to show amt of neuroblast (indicates differentiation).
adddrawfn 2 .1 .1 .1 "max(0.0,min(1.0,z[neuroblast]/6.0))"

# Draw filled green circle to show how inhibited this cell is.
adddrawfn 2   0 1 0
       "min((env[contact+receptor]/env[contact+autostick])*0.25, 0.98)"
```

A.3 'Compartments' Experiment

This is the experiment file for Figure 5.3.

```
nchems = 1;  /* How many diffusable chemicals in environment */
user_state = 1; /* Number of state variables user wants */

/* diffusion and dissipation coefficients for all chemicals */
diffusion[0] = 4.0;
dissipation[0] = 1.0;

setcolor(0, 0.8, 0.2, 0.0);  /* color of diffusing chem 0 */

/* The values fx, fy, fsize, etc are integers
*  which are used as array indices into the state array
*
* Array indices for the state array (z[])
*
*  System defined (defined for every experiment):
*   fx, fy, fsize  – forces on x, y, radius
```

```
*   split, die, emitgc – event indices
*   homophilic – homophilic cell surface chemical,
*      sticks to same chemical on adjacent cells
*   spew – rate at which to emit chemical 0
*
*  User defined (just for this experiment):
*   rundown – this indexes a state variable
*     which decreases over time, diluted by cell division
*     as well as consumed inside the cell. Controls how
*     many generations of cell division occur.
*
* Array indices for local environment array (env[])
*  rvl, rdx, rdy  – chemical 0 value and gradient
*  radius – size of cell
*  stuck – how much of the homophilic cell adhesion molecule
*   on our surface is bound.
*/
int rundown = nstate+0;

/* addstmt(condition, index, consequent);
* adds a ’gene’ to the artificial genome which does:
* if (condition(state, env))
* then dstate[index]/dt += consequent(state, env)
*/
beginprogram();
addstmt("(TRUE)", fx, "-state[fx]");
addstmt("(TRUE)", fy, "-state[fy]");
addstmt("(TRUE)", fsize, "-state[fsize]");   /* Grow  */
addstmt("(TRUE)", split, "-state[split]");   /* Split */

/* just dilute down to 0 as we grow and divide */
addstmt("(1)", rundown, "-1");

/* initially, split if we aren’t stuck to too many neighbors */
/* (until rundown runs down) */
addstmt("((state[rundown] > 30) && (env[stuck] < 0.6)) ||
        (env[radius] > 1.0)", split, 4.3);
addstmt("(state[rundown] > 30) && (env[stuck] < 0.6)", fsize, 0.1);

/* later, split if there is not much of chemical 0 */
addstmt("(env[rvl] < 0.2) || (env[radius] > 1.0)", split, 4.3);
addstmt("(env[rvl] < 0.2)", fsize, 0.1);

/* Spew until we see enough of chem 0 in the local environment */
addstmt("(1)", "spew", "5*(0.5 - env[rvl])");
```

```
/* if we are not stuck to enough neighbors, move towards chem 0,
*  else if too many, move away */
addstmt("(env[stuck] < 0.2)", fx, "7*env[rdx]");
addstmt("(env[stuck] < 0.2)", fy, "7*env[rdy]");
addstmt("(env[stuck] > 0.6)", fx, "-7*env[rdx]");
addstmt("(env[stuck] > 0.6)", fy, "-7*env[rdy]");

/* set the concentration of adhesion molecule in the membrane */
addstmt("1", "homophilic", "0.4-state[homophilic]");

addcell("FirstCell", 5.0, -0.48); /* initial cell name and pos */

/* set initial value for state[rundown] */
/* changing values here makes different numbers of cells */
initcell("FirstCell", rundown, 1200);
```

Appendix B

Details on the Artificial Evolution

As discussed in Chapter 7, the evolution experiments we have performed to date are preliminary. This chapter serves as an informal record of some of the techniques we've tried so far.

B.1 Genetic Programming

We describe here the standard simple version of genetic programming as described by Koza [Koza, 1992], with some modifications for our application. Overview of entire algorithm:

```
population = Make initial population
Loop for n_generations or until best individual is good enough
  evaluate fitness of each individual in population
  population = Breed_Next_Generation(population)  (depends on fitnesses, too)
```

Once every organism in the current generation has undergone development, it is evaluated by an objective function and assigned a **raw fitness**. Depending on the problem, this may be a value to be minimized or maximized, and there may or may not be a limiting value (e.g., minimize to zero, or maximize unbounded).

After raw fitnesses are assigned, a new population is created from the old one. This is done by choosing individuals from the old population (using some selection method as described in Sec. B.1 below), and then processing them via some reproduction operations. These reproduction operations do not always produce exact copies; they may introduce mutations, perform crossover, or change the genomes in various ways, as described in Sec. B.1.

Here is the algorithm for making the next generation from the last:

```
Breed_Next_Generation(n_desired previous-population)
  Loop for i = 0 up to n_desired
    Choose a reproduction operation to perform
    Choose_individual (or pair) from previous-population
      using some selection method (see below)
    Perform the reproduction operation, putting new individuals in new population
```

Selection Methods

A selection method is a method of selecting which individuals from a population will reproduce. There are a variety of selection methods in popular use among genetic programming enthusiasts; we describe two here.

Tournament Selection:

```
Choose_individual()
  Pick two of the organisms at random
  Return organism which has better fitness
```

Here's another idea for modified tournament selection: choose A over B with probability $a/(a+b)$.

Fitness Proportionate Reproduction:

With probability $p_i = f_i / \sum_{j \in \text{organisms}} f_j$, where f_i is raw fitness of organism i.

```
Choose_individual()
  Choose randomly from population, but
   weight probability of each organism by its fitness
```

Simple Reproduction Operations

The reproduction operations introduce variation in the offspring. We first present some reproduction operations used in Koza's example code, and then in the next section we describe the operations we will use which are tailored to our particular genome and application.

Every organism in the new population (for Koza's example) is generated by one of the operations in this table.

Reproduction-with-variation operations used in Koza's example code.

Operation	Variable Name
crossover at a function (internal) point in the tree	*crossover-at-function-point-fraction*
crossover at any point in the tree (including terminals)	*crossover-at-any-point-fraction*
copy the existing organism	*fitness-proportionate-reproduction-fraction*
replace node in expression tree with random new subtree	"mutate" (default)

Reproduction operations using the Developmental Model

In our developmental model, we have a particular syntax for the genome. The reproduction operations are specially constructed to make only syntactically valid changes. In addition, we add some operations to change particular elements in (hopefully) more useful ways.

Structure of the Genome:

Our genome consists of a list of statements of the form `condition index consequent`. The `index` identifies the state variable which this statement modifies, as described in Chapter 2. The list is divided into contiguous groups of statements which are called cell programs. At any given time, a cell's actions are governed by exactly one cell program. When cells split, the daughter cells are assigned their own cell programs which may differ from that of the parent (although generally they are the same). To recap:

Genome = list of cell-programs
Cell-program = list of stmts
Stmt = condition index consequent

Note: a stmt is a cell state equation, containing a condition, an index, and a consequent. The index indicates which state variable is modified by this equation.

Consider more restricted form for genome statements:

We may wish to consider restricting the form of expressions in our genome statements (rather than running simple GP on them, we can take into account 'reasonableness' criteria for our application).

Perhaps we should choose different forms for different types of state variables. For instance, we could classify vars as *motile-force, regulatory, chemical-spew-output*, etc., and restrict their form to "sensible" things. Also, imagine that f_{turn} can only depend on directions of gradients of chemicals...

This would require trickier evolution code (to keep correct syntax), but would make the search space smaller and, hopefully, better.

Danger: might restrict-away good solutions.

Reproduction Operations on our Developmental Genome:

There are two levels of operations: those on entire organisms, and those on individual **stmts** in the genome.

Organism reproduction-with-variation operations.

Crossover is between two genomes.

crossover stmt(s)	maybe do this as: choose a linear splitting point
crossover entire cell-program blocks	
copy organism	copy existing organism, no changes

stmt reproduction-with-variation operations
performed on one stmt of a genome
(or perhaps 2 stmts, for crossover operations)

mutate condition	replace (internal or leaf) expression tree with random new subtree
mutate index	change variable which this stmt affects
mutate consequent	replace (internal or leaf) expression tree with random new subtree
modify leaf node	e.g., add gaussian distributed noise to floating point number or add/subtract 1 to a stmt index (change variable it affects)
duplicate stmt(s)	
swap conditions	swap conditions with another stmt in this genome
swap indices	swap indices with another stmt in this genome
swap consequent	swap consequents with another stmt in this genome
Move stmts	move a block of stmts from one cell-program to another in this genome

Thoughts on the form of the artificial genome for evolution

We can sort of do introns/exons (with conditions = 0), but better would be having "start" and "stop" statements to comment out sections. The genome is linear (no hierarchy), as in natural genome (linear). What about pairs of genes? Should we have haploidy or diploidy? As ever, deciding what is important and how many parameters to twiddle is a big pain.

B.2 Example LISP Genome

Here is an example of the LISP version of a genome:

```
((FLOATPARMS (DIFF0_DEFAULT 1.0) (DIFF1_DEFAULT 1.0) (DISS0_DEFAULT 0.0)
  (DISS1_DEFAULT 0.1) (MAXTIME 20) (DT 0.5))
 (PROGRAM GCPROG0 (STMT 1 FX (* -0.5 FX)) (STMT 1 FY (* -0.5 FY))
  (STMT 1 DIE (* -0.05 DIE))
  (STMT (> CONTACT+4 9.9999994e-5) FX (* -10.0 FX))
  (STMT (> CONTACT+4 9.9999994e-5) FY (* -10.0 FY))
  (STMT 1 FSIZE (* -1.0 FSIZE))
  (STMT 1 SPLIT (* -1.0 SPLIT))
  (STMT 1 FSIZE (* 0.001 RVL))
  (STMT 1 SPLIT (* 9.9999994e-5 GVL))
  (STMT 1 SURF+4 (- 0.5 SURF+4))
  (STMT 1 SPEW+0 (- 0.3 RVL))
  (STMT 1 SPEW+1 0.0)
  (STMT (AND (> (SAFE_SQRT (+ (* GDX GDX) (* GDY GDY))) 5.0000006e-4)
   (< CONTACT+4 1.0e-5)) FX
   (- (* -1.0 (/ GDX (SAFE_SQRT (+ (* GDX GDX) (* GDY GDY))))) (* 3 FX)))
  (STMT (AND (> (SAFE_SQRT (+ (* GDX GDX) (* GDY GDY))) 5.0000006e-4)
   (< CONTACT+4 1.0e-5)) FY
   (- (* -1.0 (/ GDY (SAFE_SQRT (+ (* GDX GDX) (* GDY GDY))))) (* 3 FX)))
  (STMT (AND (> (SAFE_SQRT (+ (* RDX RDX) (* RDY RDY))) 5.0000006e-4)
```

```
  (< CONTACT+4 1.0e-5)) FX
  (- (* 1.0 (/ RDX (SAFE_SQRT (+ (* RDX RDX) (* RDY RDY))))) (* 3 FX)))
 (STMT (AND (> (SAFE_SQRT (+ (* RDX RDX) (* RDY RDY))) 5.0000006e-4)
  (< CONTACT+4 1.0e-5)) FY
  (- (* 1.0 (/ RDY (SAFE_SQRT (+ (* RDX RDX) (* RDY RDY))))) (* 3 FX)))
 (STMT (AND (AND (< FX 0.003) (< FY 0.003)) (< CONTACT+4 0.05))
  DIE (* 0.05 5.5)))
(PROGRAM PROG0
 (STMT 1 FX (* -0.5 FX)) (STMT 1 FY (* -0.5 FY))
 (STMT 1 FSIZE (* -1.0 FSIZE))
 (STMT 1 SPLIT (* -1.0 SPLIT)) (STMT 1 DIE (* -1.0 DIE))
 (STMT 1 FX RDX) (STMT 1 FY (SIN SPLIT))
 (STMT 1 FSIZE (* 0.001 RVL)) (STMT 1 SPLIT (* 0.01 DIE))
 (STMT (AND (< LOCAL+0 0.5) (> RADIUS 1.0)) SPLIT 1.0)
 (STMT 1 SURF+5 (- 0.5 SURF+5))
 (STMT 1 SPEW+0 0.0)
 (STMT 1 SPEW+1 0.5))
(INITIAL-CONDITIONS
 (ADDCELL A GCPROG0 (0 0 0 0.1 0 0.3 0 0 0) (3.0 -0.4 0.5 0.1))
 (ADDCELL B PROG0 (0 0 0.1 0.2 0 0 0 0 0) (-0.3 4.0 0.5 -0.3)))
(INTPARMS
  (DISHNVALS 3) (DISHNPTS 50) (RANDOM_SPLIT_ANGLE 1)
  (RANDOM_SENSOR_LOCATION 1) (NCELL_LOCAL_VARS 2) (NCELL_SURF_VARS 6))
```

Bibliography

[Agarwal, 1993] Agarwal, P. (1993). The cell programming language. Technical Report TR630, NYU CS Dept, Courant Institute of Mathematical Sciences. Available via ftp or WWW from `cs.nyu.edu`.

[Alberts et al., 1989] Alberts, B., Bray, D., Lewis, J., Raff, M., Roberts, K., and Watson, J. D. (1989). *Molecular Biology of the Cell*. Garland Publishing, New York, NY, 2 edition.

[Angeline et al., 1994] Angeline, P. J., Saudners, G. M., and Pollack, J. B. (1994). An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(1):54–65.

[Armstrong, 1989] Armstrong, P. B. (1989). Cell sorting out: The self-assembly of tissues *in vitro*. *CRC Critical Reviews in Biochemistry and Molecular Biology*, 24:119–149.

[Arvo and Kirk, 1988] Arvo, J. and Kirk, D. (1988). Modeling plants with environment-sensitive automata. In *Proceedings of Ausgraph '88*, pages 27–33.

[Back et al., 1991] Back, T., Hoffmeister, F., and Schwefel, H.-P. (1991). A survey of evolution strategies. In Belew, R. K. and Booker, L. B., editors, *Proc. of the Fourth Int'l. Conf. on Genetic Algorithms*, pages 2–9. Morgan Kaufmann.

[Bard, 1981] Bard, J. B. L. (1981). A model for generating aspects of zebra and other mammalian coat patterns. *Journal of Theoretical Biology*, 93.

[Barzel, 1992] Barzel, R. (1992). *Physically-Based Modeling: A Structured Approach*. Academic Press, Cambridge, MA.

[Beer and Gallagher, 1991] Beer, R. D. and Gallagher, J. C. (1991). Evolving dynamical neural networks for adaptive behavior. *Adaptive Behavior*, 1.

[Belew, 1993] Belew, R. K. (1993). Interposing an ontogenic model between genetic algorithms and neural networks. In *Proceedings of Neural Information Processing Systems 5*. Morgan Kaufmann.

[Berg, 1983] Berg, H. C. (1983). *Random Walks in Biology*. Princeton University Press, New Jersey.

[Booch, 1991] Booch, G. (1991). *Object Oriented Design with Applications*. Benjamin/Cummings Publishing Co., Inc.

[Brown et al., 1991] Brown, M. C., Hopkins, W. G., and Keynes, R. J. (1991). *Essentials of Neural Development*. Cambridge University Press, Cambridge, England.

[Buss, 1987] Buss, L. W. (1987). *The evolution of individuality*. Princeton University Press, Princeton, NJ.

[Cangelosi et al., 1995] Cangelosi, A., Parisi, D., and Nolfi, S. (1995). Cell division and migration in a 'genotype' for neural networks. *Network: Computation in Neural Systems*, pages 497–515. In press. Also available via FTP at `kant.irmkant.rm.cnr.it:pub/econets/cangelosi.migration.ps.Z`.

[Carr and Konishi, 1988] Carr, C. E. and Konishi, M. (1988). Axonal delay lines for time measurement in the owl's brainstem. *Proceedings of the National Academy of Science, Neurobiology (USA)*, 85:8311–8315.

[Cook, 1984] Cook, R. L. (1984). Shade trees. In Christiansen, H., editor, *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 223–231.

[Crawford and Stocum, 1988] Crawford, K. and Stocum, D. L. (1988). Retinoic acid coordinately proximalizes regenerate pattern an blastema differential affinity in axolotl limbs. *Development*, 102:687–698.

[Crow, 1982] Crow, F. C. (1982). A more flexible image generation environment. In *Computer Graphics (SIGGRAPH '82 Proceedings)*, volume 16, pages 9–18.

[Darnell et al., 1990] Darnell, J. E., Lodish, H., and Baltimore, D. (1990). *Molecular cell biology*. Scientific American Books, New York, 2 edition.

[Dawkins, 1976] Dawkins, R. (1976). *The Selfish Gene*. Oxford.

[Dawkins, 1982] Dawkins, R. (1982). *The Extended Phenotype: the gene as the unit of selection*. Oxford.

[Dawkins, 1986] Dawkins, R. (1986). *The Blind Watchmaker*. W.W. Norton, New York.

[de Boer, 1989] de Boer, M. (1989). *Analysis and computer generation of division patterns in cell layers using developmental algorithms*. Ph.D. dissertation, University of Utrecht.

[de Boer et al., 1992] de Boer, M., Fracchia, D., and Prusinkiewicz, P. (1992). Analysis and simulation of the development of cellular layers. In Langton, C. G., Taylor, C., Farmer, J. D., and Rasmussen, S., editors, *Artificial Life II*. Addison-Wesley.

[Dellaert and Beer, 1994] Dellaert, F. and Beer, R. D. (1994). Toward an evolvable model of development for autonomous agent synthesis. In Brooks, R. A. and Maes, P., editors, *Artificial Life IV*, Cambridge, MA. A Bradford Book, MIT Press.

[Duncan, 1994] Duncan, J. (1994). The making of a rockbuster. *Cinefex: the Journal of Cinematic Illusions*, 58:34–65.

[Edelstein-Keshet and Ermentrout, 1990] Edelstein-Keshet, L. and Ermentrout, G. B. (1990). Models for contact-mediated pattern formation: cells that form parallel arrays. *Journal of Mathematical Biology*, 29.

[Fleischer, 1994] Fleischer, K. (1994). Cells: Simulations of multicellular development. In *Siggraph Video Review, Number 101. A video presented at Siggraph 94.*

[Fleischer, 1995] Fleischer, K. W. (1995). *A Multiple-Mechanism Developmental Model for Defining Self-Organizing Structures*. PhD dissertation, Caltech, Department of Computation and Neural Systems.

[Fleischer and Barr, 1994] Fleischer, K. W. and Barr, A. H. (1994). A simulation testbed for the study of multicellular development: The multiple mechanisms of morphogenesis. In Langton, C. G., editor, *Artificial Life III*. Addison-Wesley.

[Fleischer et al., 1995] Fleischer, K. W., Laidlaw, D. H., Currin, B. L., and Barr, A. H. (1995). Cellular texture generation. *To appear in Proceedings of ACM SIGGRAPH '95, Los Angeles, CA*.

[Fogel et al., 1990] Fogel, D. B., Fogel, L. J., and Porto, V. W. (1990). Evolving neural networks. *Biological Cybernetics*, 63.

[Fowler et al., 1992a] Fowler, D. R., Meinhardt, H., and Prusinkiewicz, P. (1992a). Modeling seashells. In Catmull, E. E., editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 379–388.

[Fowler et al., 1992b] Fowler, D. R., Prusinkiewicz, P., and Battjes, J. (1992b). A collision-based model of spiral phyllotaxis. In Catmull, E. E., editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 361–368.

[Fraser and Perkel, 1990] Fraser, S. and Perkel, D. (1990). Competitive and positional cues in the patterning of nerve connections. *J. Neuro.*, 21(1).

[Gear, 1971] Gear, C. W. (1971). *Numerical Initial Value Problems in Ordinary Differential Equations*. Prentice-Hall, Englewood Cliffs, NJ.

[Gell-mann, 1990] Gell-mann, M. (1990). The santa fe institute. Transcript of a talk given January 9, 1990, at SFI. This may be available from the Santa Fe Institute, Santa Fe, NM.

[Getting, 1989] Getting, P. (1989). *Ann. Rev. Neuroscience*, 12.

[Gilbert, 1991] Gilbert, S. (1991). *Developmental Biology*. Sinnauer Associates, 3 edition.

[Glazier and Graner, 1993] Glazier, J. A. and Graner, F. (1993). Simulation of the differential adhesion driven rearrangement of biological cells. *Physical Review E*, 41(3).

[Goldstein, 1980] Goldstein (1980). *Classical Mechanics*. Addison-Wesley.

[Goodwin, 1994] Goodwin, B. (1994). *How the Leopard Changed Its Spots*. Charles Scribner's Sons.

[Goodwin et al., 1993] Goodwin, B. C., Kauffman, S. A., and Murray, J. D. (1993). Is morphogenesis an intrinsically robust process? *Journal of Theoretical Biology*, 163.

[Greene, 1989] Greene, N. (1989). Voxel space automata: Modeling with stochastic growth processes in voxel space. In Lane, J., editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 175–184.

[Grillner and Wallen, 1985] Grillner, S. and Wallen, P. (1985). *Ann. Rev. Neuroscience*, 8.

[Gruau, 1993] Gruau, F. (1993). Genetic synthesis of modular neural networks. In Forrest, S., editor, *Proc. of the Fifth Int'l. Conf. on Genetic Algorithms*. Morgan Kaufmann.

[Harp et al., 1989] Harp, S. A., Tariq, S., and Guha, A. (1989). Towards the genetic synthesis of neural networks. In Schaffer, J. D., editor, *Proc. of the Third Int'l. Conf. on Genetic Algorithms*. Morgan Kaufmann.

[Harrison, 1993] Harrison, L. G. (1993). *Kinetic theory of living pattern*. Cambridge University Press, Cambridge ; New York.

[Hebb, 1949] Hebb, D. O. (1949). *The organization of behavior*. Wiley, New York.

[Holland, 1992] Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems*. MIT Press/Bradford Books, 2 edition.

[Honda, 1978] Honda, H. (1978). Description of cellular patterns by dirichlet domains: the two-dimensional case. *Journal of Theoretical Biology*, 72:523–543.

[Honda, 1983] Honda, H. (1983). Geometrical models for cells in tissues. *International Review of Cytology*, 81:191–248.

[Hopfield, 1984] Hopfield, J. J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Science (USA)*, 81:3088–3092.

[Kaandorp, 1994] Kaandorp, J. (1994). *Fractal modelling : growth and form in biology*. Springer-Verlag, Berlin ; New York.

[Kajiya, 1985] Kajiya, J. T. (1985). Anisotropic reflection models. In Barsky, B. A., editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 15–21.

[Kajiya and Kay, 1989] Kajiya, J. T. and Kay, T. L. (1989). Rendering fur with three dimensional textures. In Lane, J., editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 271–280.

[Kauffman, 1993] Kauffman, S. A. (1993). *Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press.

[Kay, 1988] Kay, T. (1988). Cli: A configurable command line interpreter. A nice software system for getting things running quickly, with utilities for peeking and poking values in the symbol table. Runs on lots of architectures, too. Assorted maintenance, extensions, porting and bug fixes by John Snyder and Kurt Fleischer.

[Kernighan and Ritchie, 1978] Kernighan, B. W. and Ritchie, D. M. (1978). *The C Programming Language*. Prentice-Hall.

[Kitano, 1990] Kitano, H. (1990). Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, 4.

[Kitano, 1994a] Kitano, H. (1994a). Evolution of metabolism for morphogenesis. In Brooks, R. A. and Maes, P., editors, *Artificial Life IV*, Cambridge, MA. A Bradford Book, MIT Press.

[Kitano, 1994b] Kitano, H. (1994b). A simple model of neurogenesis. Technical Report SCSL-TM-94-4, Sony Computer Science Laboratory, Inc., 3-14-13 Higashi-Gotanda, Shinagawa, Tokyo 141 Japan.

[Koch and Segev, 1989] Koch, C. and Segev, I. (1989). *Methods in Neuronal Modeling : from Synapses to Networks*. MIT Press, Cambridge, MA.

[Konishi, 1993] Konishi, M. (1993). Listening with two ears. *Scientific American*.

[Koza, 1992] Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA.

[Lin and Segel, 1974] Lin, C. C. and Segel, L. A. (1974). *Mathamatics Applied to Deterministic Problems in the Natural Sciences*. Macmillan Publishing, New York. Dictostelium (slime mold) modeling has been described by many authors; this applied mathematics textbook includes a nice description.

[Lindenmayer, 1968] Lindenmayer, A. (1968). Mathematical models for cellular interaction in development, parts i and ii. *J. Theo. Bio.*, 18.

[Lindenmayer and Prusinkiewicz, 1989] Lindenmayer, A. and Prusinkiewicz, P. (1989). Developmental models of multicellular organisms: A computer graphics perspective. In Langton, C. G., editor, *Artificial Life*. Addison-Wesley.

[Lyon and Mead, 1988] Lyon, R. F. and Mead, C. A. (1988). An analog electronic cochlea. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 36.

[Marx, 1994] Marx, J. (1994). Stalking the start of colon cancer. *Science*, 263. Research News (summary).

[Maynard-Smith, 1986] Maynard-Smith, J. (1986). *The Problems of Biology*. Oxford.

[Mead, 1989] Mead, C. A. (1989). *Analog VLSI and Neural Systems*. Addison-Wesley.

[Mead and Mahowald, 1988] Mead, C. A. and Mahowald, M. (1988). A silicon model of early visual processing. *Neural Networks*, 1.

[Meinhardt, 1982] Meinhardt, H. (1982). *Models of Biological Pattern Formation*. Academic Press, London.

[Michaelis and Menten, 1913] Michaelis, L. and Menten, M. I. (1913). Der kinetik der invertin-wirkung. *Biochem. Z.*, 49.

[Miller and Pearce, 1989] Miller, G. and Pearce, A. (1989). Globular dynamics: A connected particle system for animating viscous fluids. *Computers and Graphics*, 13(3):305–309.

[Miller et al., 1989] Miller, G., Todd, P., and Hegde, S. (1989). Designing neural networks using genetic algorithms. In *Proc. of the Third Int'l. Conf. on Genetic Algorithms*, pages 379–384. Morgan Kaufmann.

[Mittenthal and Baskin, 1992] Mittenthal, J. E. and Baskin, A. B., editors (1992). *Principles of Organization in Oranisms : Proceedings of the Workshop held June 1990, in Santa Fe, New Mexico*, Santa Fe Institute Studies in the Sciences of Complexity. Addison-Wesley.

[Mjolsness et al., 1991] Mjolsness, E., Sharp, D., and Reinitz, J. (1991). A connectionist model of development. *J. Theo. Bio.*, 152.

[Murray, 1981] Murray, J. D. (1981). On pattern formation mechanisms for lepidoptera wing patterns and mammalian coat markings. *Philosophical Transactions of the Royal Society (B)*, 295.

[Murray, 1993] Murray, J. D. (1993). *Mathematical Biology*. Springer-Verlag, New York, 2nd edition.

[Nagorcka et al., 1987] Nagorcka, B. N., Manoranjan, V. S., and Murray, J. D. (1987). Complex spatial patterns from tissue interactions – an illustrative model. *Journal of Theoretical Biology*, 128:359–374.

[Nolfi et al., 1995] Nolfi, S., Miglino, O., and Parisi, D. (1995). Phenotypic plasticity in evolving neural networks. In Gaussier, P. and Nicoud, J.-D., editors, *Proceedings of the First Conference From Perception to Action (at Lausanne)*. IEEE Press, Los Alamitos, CA. Available via FTP at `kant.irmkant.rm.cnr.it: pub/econets/nolfi.perac.ps.Z`.

[Odell et al., 1981] Odell, G. M., Oster, G., Alberch, P., and Burnside, B. (1981). The mechanical basis of morphogenesis. *Developmental Biology*, 85.

[Oster et al., 1990] Oster, G., Weliky, M., and Minsuk, S. (1990). Morphogenesis by cell intercalation. In Jen, E., editor, *1989 Lectures in Complex Systems*, Sante Fe Institute Studies in the Sciences of Complexity. Addison-Wesley.

[Page, 1994] Page, T. L. (Mar 18, 1994). Time *is* the essence: Molecular analysis of the biological clock. *Science*, 263. Perspectives (summary/review).

[Pedersen, 1994] Pedersen, H. K. (1994). Displacement mapping using flow fields. In Glassner, A., editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, pages 279–286. ACM Press.

[Platt, 1989] Platt, J. (1989). *Constraint Methods for Neural Networks and Computer Graphics*. Ph.D. dissertation, Caltech, Department of Computer Science, Pasadena, CA, 91125.

[Prusinkiewicz et al., 1993] Prusinkiewicz, P., Hammel, M., and Mjolsness, E. (1993). Animation of plant development using differential l-systems. *Computer Graphics*, 27. (Proceedings of Siggraph '93).

[Prusinkiewicz et al., 1994] Prusinkiewicz, P., James, M., and Měch, R. (1994). Synthetic topiary. In Glassner, A., editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, pages 351–358. ACM Press.

[Prusinkiewicz and Lindenmayer, 1990] Prusinkiewicz, P. and Lindenmayer, A. (1990). *The Algorithmic Beauty of Plants*. Springer-Verlag, New York.

[Purves and Lichtman, 1985] Purves, D. and Lichtman, J. W. (1985). *Principles of Neural Development*. Sinauer Associates, Sunderland, MA.

[Reeves, 1983] Reeves, W. T. (1983). Particle systems – a technique for modeling a class of fuzzy objects. *ACM Trans. Graphics*, 2:91–108.

[Reeves and Blau, 1985] Reeves, W. T. and Blau, R. (1985). Approximate and probabilistic algorithms for shading and rendering structured particle systems. In Barsky, B. A., editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 313–322.

[Reinitz et al., 1992] Reinitz, J., Mjolsness, E., and Sharp, D. H. (1992). Model for cooperative control of positional information in *drosophila* by *bicoid* and maternal *hunchback*. Research Report YALEU/DCS/RR-922, Yale University, Computer Science Department, New Haven, CT. Also available as Los Alamos Unclassified Report 92-2942.

[Reynolds, 1987] Reynolds, C. W. (1987). Flocks, herds, and schools: A distributed behavioral model. In Stone, M. C., editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 25–34.

[Shampine et al., 1991] Shampine, L., Gladwell, I., and Brankin, R. (1991). Reliable solution of special event location problems for odes. *ACM Transactions on Mathematical Software*, 17(1).

[Sims, 1990] Sims, K. (1990). Particle animation and rendering using data parallel computation. In Baskett, F., editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 405–413.

[Sims, 1991a] Sims, K. (1991a). Artificial evolution for computer graphics. *Computer Graphics*, 25(4). (Proceedings of Siggraph '91).

[Sims, 1991b] Sims, K. (1991b). Artificial evolution for computer graphics. In Sederberg, T. W., editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 319–328.

[Sims, 1991c] Sims, K. (1991c). Interactive evolution of dynamical systems. In *Proceedings of the First European Conference on Artificial Life*, Paris. MIT Press.

[Smith, 1984] Smith, A. R. (1984). Plants, fractals and formal languages. In Christiansen, H., editor, *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 1–10.

[Snyder, 1992] Snyder, J. (1992). *Generative Modeling for Computer Graphics and CAD: Symbolic Shape Design using Interval Analysis*. Academic Press.

[Snyder and Barr, 1987] Snyder, J. M. and Barr, A. H. (1987). Ray tracing complex models containing surface tessellations. In Stone, M. C., editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 119–128.

[Steinberg, 1964] Steinberg, M. S. (1964). The problem of adhesive selectivity in cellular interactions. In Locke, M., editor, *Cellular Membranes in Development*. Academic Press.

[Steinberg, 1970] Steinberg, M. S. (1970). Does differential adhesion govern self-assembly processes in histogenesis? equilibrium configurations and the emergence of a hierarchy among populations of embryonic cells. *J. Exp. Zool.*, 173.

[Steinberg and Takeichi, 1994] Steinberg, M. S. and Takeichi, M. (1994). Experimental specification of cell sorting, tissue spreading and specific spatila patterning by quantitative differences in cadherin expression. *Proceedings of the National Academy of Science (USA)*, 91. Developmental Biology.

[Stork et al., 1992] Stork, D. G., Jackson, B., and Walker, S. (1992). Non-optimality via preadaptation in simple neural systems. In Langton, C. G., Taylor, C., Farmer, J. D., and Rasmussen, S., editors, *Artificial Life II*. Addison-Wesley.

[Strang, 1986] Strang, G. (1986). *Introduction to Applied Mathematics*. Wellesley-Cambridge Press.

[Szeliski and Tonnesen, 1992] Szeliski, R. and Tonnesen, D. (1992). Surface modeling with oriented particle systems. In Catmull, E. E., editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 185–194.

[Terzopoulos et al., 1989] Terzopoulos, D., Platt, J., and Fleischer, K. (1989). From goop to glop: Heating and melting deformable models. In *Graphics Interface 89*.

[Torreele, 1991] Torreele, J. (1991). Temporal processing with recurrent networks: An evolutionary approach. In Belew, R. K. and Booker, L. B., editors, *Proc. of the Fourth Int'l. Conf. on Genetic Algorithms*. Morgan Kaufmann.

[Turing, 1952] Turing, A. (1952). The chemical basis of morphogenesis. *Phil. Trans. B.*, 237.

[Turk, 1991] Turk, G. (1991). Generating textures for arbitrary surfaces using reaction-diffusion. In Sederberg, T. W., editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 289–298.

[Turk, 1992] Turk, G. (1992). Re-tiling polygonal surfaces. In Catmull, E. E., editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 55–64.

[Vaario, 1994] Vaario, J. (1994). Modeling adaptive self-organization. In Brooks, R. A. and Maes, P., editors, *Artificial Life IV*, Cambridge, MA. A Bradford Book, MIT Press.

[Waddington, 1968] Waddington, C. H., editor (1968). *Towards a theoretical biology (An International Union of Biological Sciences Symposium)*. Edinburgh University Press and Aldine Publishing Co., Chicago. Three volumes, from symposia in 1966, 1967 and 1968.

[Watts, 1993] Watts, L. (1993). A tour of neuralog and spike, tools for simulating networks of spiking neurons. For more information contact `lloyd@pcmp.caltech.edu`.

[Weliky and Oster, 1990] Weliky, M. and Oster, G. (1990). The mechanical basis of cell rearrangement. i. epithelial morphogenesis during *fundulus* epiboly. *Development*, 109.

[Westin et al., 1992] Westin, S. H., Arvo, J. R., and Torrance, K. E. (1992). Predicting reflectance functions from complex surfaces. In Catmull, E. E., editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 255–264.

[Whitley and Hanson, 1989] Whitley, D. and Hanson, T. (1989). Optimizing neural networks using faster, more acurate genetic search. In *Proc. of the Third Int'l. Conf. on Genetic Algorithms*. Morgan Kaufmann.

[Wilson, 1970] Wilson, B. F. (1970). *The growing tree*. University of Massachusetts Press, Amherst, MA.

[Witkin et al., 1987] Witkin, A., Fleischer, K., and Barr, A. (1987). Energy constraints on parameterized models. In Stone, M. C., editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 225–232.

[Witkin and Kass, 1991] Witkin, A. and Kass, M. (1991). Reaction-diffusion textures. In Sederberg, T. W., editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 299–308.

[Witkin and Heckbert, 1994] Witkin, A. P. and Heckbert, P. S. (1994). Using particles to sample and control implicit surfaces. In Glassner, A., editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, pages 269–278. ACM Press.

[Yao, 1993] Yao, X. (1993). Evolutionary artificial neural networks. *International Journal of Neural Systems*, 4(3).