A Simulation Testbed for the Study of Multicellular Development:

The Multiple Mechanisms of Morphogenesis

Kurt Fleischer
Alan H. Barr
Computation and Neural Systems
Mail Stop 350-74
California Institute of Technology
Pasadena, California, 91125

ABSTRACT

This paper presents a simulation framework and computational testbed for studying multicellular pattern formation. The approach combines several developmental mechanisms (chemical, mechanical, genetic and electrical) known to be important for biological pattern formation. The mechanisms are present in an environment containing discrete cells which are capable of independent movement (cell migration). Experience with the testbed indicates that the *interactions* between the developmental mechanisms are important in determining multicellular and developmental patterns.

Each simulated cell has an artificial genome whose expression is dependent only upon its internal state and its local environment. The changes of each cell's state and of the environment are determined by piecewise continuous differential equations. The current two-dimensional simulation exhibits a variety of multicellular behaviors, including cell migration, cell differentiation, gradient following, clustering, lateral inhibition, and neurite outgrowth (see color plates).

We plan to perform simulated evolution on developmental models as part of a long range goal to create artificial neural networks which solve problems in perception and control [Fleischer]. The testbed is a step on the path towards this goal.

1 Introduction

The testbed system described in this paper is part of a larger project to generate a new class of artificial neural networks. We are studying the problem of generating artificial neural networks that share some properties of biological neural networks, in particular:

[0] ©Kurt Fleischer 1993. To appear in **Artificial Life III**, Addison-Wesley, 1993.

- ◇ **problem-specific geometric structure:** Some biological neural circuits solve a problem largely by assuming a particular geometric configuration. For example, the owl auditory localization circuit [Carr & Konishi] uses intercalated axons as delay lines to compute the difference between time of arrival of auditory signals.
- ◇ **asymmetric topological connectivity:** Most artificial neural networks have a prescribed regular connectivity. Biological systems exhibit a variety of connection topologies.
- ◇ **heterogeneous neural types:** Most neural circuits involve multiple neural types which have different morphology and function.

These properties are evident in real neural networks, and are believed to be closely related to the functions they compute. To capture these properties in an artificial neural network, we have chosen to perform simulated evolution on a developmental model.[1]

Why evolve a *developmental* model? 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
- ◇ developmental gain – a small change in the genome can make a large change in the organism (eg. add another layer, add another segment)

The simulation testbed presented in this paper was created to find a simple developmental model which can create neural networks with arbitrary topological connectivity and a large degree of geometric complexity. The testbed must be flexible so that we can explore different strategies for development and evolution.

We use the term *modeling framework* to refer to our modeling abstraction, which we distinguish from the *testbed* (our implementation of the framework)[2]. For example, our framework contains the concept of an environment with diffusing chemicals. The testbed currently implements this as a discretized grid in two dimensions, and the cells' sensors are implemented as smooth interpolations on this grid. We may later decide to change the implementation, or extend it to three dimensions, but our abstraction (the modeling framework) would remain the same.

From developmental biology, we know that the gross structure of a multicellular organism arises from the expression in each cell of preprogrammed genetic information in a particular environment. Each cell inherits state from its parent, in the form of its genome as well as other cellular materials and structures (eg. organelles, mRNA, etc). Although each cell only has access to local information, the development process robustly creates successful organisms, even in the presence of fairly severe perturbations. What sort of local behaviors can account for this?

[1] A *developmental model* is a model that captures some aspect of the changes that occur as an organism grows to achieve its mature shape and function.

[2] This distinction between abstraction and implementation is a useful tool in applied mathematics and other fields [Barzel].

Previous mathematical models of morphogenesis have shown that chemical effects can account for some behaviors[Turing, Meinhardt], mechanical effects for others[Odell et al], and cell-lineage control of the geometry of cell division can account for yet other shapes[Lindenmayer & Prusinkiewicz, de Boer]. We combine these factors in one modeling system, to explore how the *interaction* between these factors can determine developmental patterns.

Our modeling framework consists of discrete cells which are capable of independent movement and are controlled by an artificial genome model. They move about in a simulated environment comprised of chemical, mechanical, and electrical elements. The individual cells are modeled as physical entities within this environment, subject to mechanical collisions, adhesive forces, and drag. Each cell's activities are determined by its internal state, and the expression of its genome within its local environment. The genome is a set of differential equations which depend on the cell's current state and its local environment. The changes in the environment are also governed by differential equations which implement the mechanical effects and the diffusion of extracellular chemicals. The current testbed is two-dimensional; extension to three dimensions is straightforward, although computationally more expensive.

Our first results show that the testbed is able to model several simple multicellular patterns. We show examples of following gradients, clustering, cell differentiation, pattern formation, and network generation (see color plates).

1.1 Overview of the paper

Several of the elements of our model are based on previous work in developmental biology. This work and other related work is briefly described in section 2.

Section 3 describes the modeling framework, and its relation to previous work. The testbed implementation is presented in in section 4 (more details appear in Appendix A). Sections 5 shows some experiments we have run with the testbed, and section 6 discusses what we have learned from our experience with the system.

2 Related Work

2.1 Previous Developmental Models

In this section we describe some previous models of development, each of which focuses on a particular developmental mechanism (chemical, mechanical, genetic, or electrical). Recently, some of these researchers have enhanced their models to include elements of the other mechanisms, as we advocate.

Chemical Factors. Turing's 1952 paper *The Chemical Basis of Morphogenesis* proposed a mathematical theory of cell-cell interaction via chemical substances (*morphogens*). 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], the chemotaxis models of *dyctyostelium* slime molds [Lin & Segel], and creating patterns such as those of the zebra coat [Bard] and some butterfly wings [Murray].

Mechanical Factors. In *The Mechanical Basis of Morphogenesis* (1981), 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]. In their system, the cell membrane is modeled as several springs with variable spring constant and rest length. These parameters are modified based on interactions between adjacent cells, and this gives rise to the various behaviors. Later work in this area uses more detailed mechanical models, and incorporates some chemical signalling to model cell intercalation [Oster] and other phenomena.

Genetic Factors: L-systems and Grammars. Grammar-based techniques such as L-systems[Lindenmayer] 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 & Lindenmayer, Lindenmayer & Prusinkiewicz], and have also been applied to modeling cell layers [Prusinkiewicz & Lindenmayer, de Boer, Fracchia and Prusinkiewicz, de Boer], topology of neural networks [Kitano],[3] and other systems.

Some grammar-based models incorporate environmental influences and cell-cell interactions using context-sensitive languages[Lindenmayer, Lindenmayer & Prusinkiewicz]. Other grammar-based models have been enhanced to include computation of physical forces for the modeling of cell layers [de Boer, Fracchia and Prusinkiewicz, Prusinkiewicz & Lindenmayer]. 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]), 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[Mjolsness et al, Prusinkiewicz, Hammel & Mjolsness].

Electrical 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 & Perkel], and that synapse formation can be strengthened in some cells when the firing of the input and output cells are correlated [Hebb].

[3]A more detailed comparison of our differential equation approach to evolving neural networks versus the grammar based approach of [Kitano] appears in [Fleischer].

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.

2.2 Other Related Work

Several researchers have independently determined that it is now computationally feasible to compute medium-scale developmental simulations for a variety of applications. We mention here two systems which bear some similarity to our model.

The Connectionist Model. Mjolsness et al. have noted the similarities between gene regulation and standard neural net dynamics [Mjolsness et al]. They use differential equations to describe dynamics of gene interactions at a short time scale, combined with a grammar-based model to describe cell state changes at a longer time scale (eg. mitosis[4], interphase, post-mitotic).

The connectionist model uses a fixed differential equation type to describe the detailed dynamics. The differential equations are of the standard connectionist form [Hopfield]: $\tau_a(\mathrm{d}v_i^a/\mathrm{d}t) = g_a(\sum_b T^{ab}v_i^b + h^a) - \lambda_a v_i^a$. 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.

This model is currently being successfully applied to the early developmental genetics of Drosophila [Reinitz et al], by optimizing the parameters 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] makes more simplifying assumptions (discrete time and space, and direct interactions between cells), which enable it to compute simulations with a few thousand cells. In the discretized spatial model, cells to exist only on grid points. A single cell can cover several adjacent grid points, creating a nontrivial shape. 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. Time is also 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. It has been applied to modeling cellular sorting by differential adhesion, aggregation in slime molds, and other phenomena.

[4]cell division

3 The Modeling Framework

We propose a multiple mechanism model for cellular development, based on the chemical, mechanical, genetic and electrical models discussed in the Previous Work section (section 2.1). Several others have been moving in this direction as well. The modeling of cell layers [Prusinkiewicz & Lindenmayer] incorporates a grammar-based model for cell lineage and a mechanical model for cell-cell interactions. The cell intercalation models of Oster et al. [Oster] combine detailed mechanical models with simple chemical signaling models. The retino-tectal model [Fraser & Perkel], the connectionist model [Mjolsness et al] and the Cell Programming Language [Agarwal] are all models which use a combination of mechanisms.

Modeling Framework (abstraction)	**Testbed** (current implementation)
Discrete cells (allows cell migration)	
cell geometry	2d circles
cell substructures	none
growth cones	modeled as small cells
neurites	path of growth cone and communication link between cell and growth cone
Genetic/Cell Lineage	
genetic control of cell operations	parallel ODEs w/conditions
inherit state from parent cell	yes
control over orientation of cell divisions	yes
asymmetric cell division	not implemented yet
Extracellular environment	
chemical	2d reaction-diffusion grid
mechanical	mechanical barriers, viscous drag
Cell-cell interactions	
mechanical	collisions and adhesion between cells
chemical (membrane proteins)	adhesion and contact recognition
electrical (gap junction, synapse)	not implemented yet
Cell-environment interactions	
chemical	emit, absorb, sense values in grid
mechanical	cell-env collisions and adhesion

Table 1: The modeling framework and its implementation.

Our modeling framework combines the multiple mechanisms within a single simulated environment (see Table 1). The environment contains diffusing and reacting chemicals, mechanical barriers, adhesive substances, etc. Cells move about within this environment, interacting with each other and with the environment.

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

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 mediated by the environment between the cells. 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 the state of the 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.

Genetic factors such as cell lineage are also important in forming developmental patterns. Some simple grammar-based systems which only model cell lineage are capable of making biologically relevant patterns [Prusinkiewicz & Lindenmayer]. The capabilities of these models motivate us to include genetic mechanisms, although our representation is somewhat different from that of L-systems. L-systems generally have pre-defined cell types and determine the types of the children directly from the parent via a production rule. Our cells inherit state from their parent, and then cells with different behavior or composition can be interpreted as different types. Our model is thus at a slightly lower level of abstraction than the L-systems models, with cell type derived rather than specified.

The *connectionist model* mentioned previously [Mjolsness et al] is a hybrid system which uses grammar rules to model state changes within individual cells, and differential equations to describe the continuous behavior within a state. This system is similar to ours in many ways, the major differences being:

- ◇ they use a fixed type of differential equation, we have a more arbitrary form (user-specified function, sections 4 and A.2.1)
- ◇ they have grammar rules for state changes, we use conditional terms on our ODEs (sections 4 and A.2.1)
- ◇ goals: they are matching biological data, we are making artificial NNs (although both systems are probably general enough to be applied in either domain)

4 Testbed Implementation

In this section, we describe in broad terms the implementation of each feature of the model sketched in Table 1. The current implementation is a subset of the entire modeling framework. The electrical model is as yet incomplete, so we leave its description to a future paper. Appendix A contains a more detailed description of the data structures, mathematical methods, and numerical solutions.

Design goals for the testbed:

◇ faithfully implement the modeling framework
◇ show the ability to generate neural networks with asymmetric structure and heterogeneous cell types
◇ the language used for genome should be amenable to simulated evolution (as discussed in the introduction)
◇ scale: hundreds of cells
◇ flexible and extensible

We model discrete cells as containing two classes of proteins: those in the cytoplasm of the cell, and those in the membrane. These proteins form the *state* of our cells. Each protein is represented by a floating point state variable ($state[i]$) describing the amount of that protein currently in the cell (or membrane). Conversion between amounts and concentration is done using cell volume.

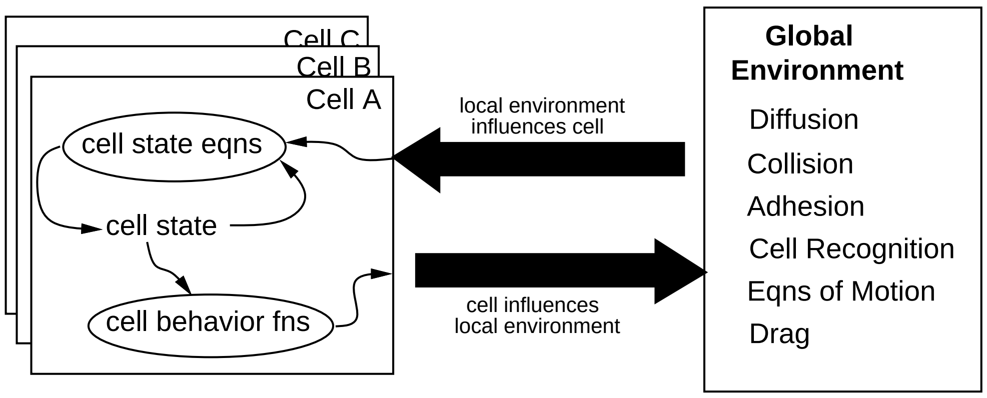

Diagram 1: Each cell's state is a modified by the *cell state equations* (the genome), which have access to the cell's local environment. The local environment information is computed from the processes in the global environment. The cell's state determines the cell's behavior via the *cell behavior functions*. The behavior can then affect the global environment (eg. by releasing a diffusable chemical).

The cells change state continuously via an artificial genome (the *cell state equations*, see Diagram 1 and section A.2.1), which encodes the change of amount of protein over time ($dstate[i]/dt$). This derivative depends on the current state of the cell and on information available in the cell's local environment (locally measured values of diffusable chemicals, amount of contact with neighbors, etc). To allow the artificial genes to regulate each other (turn each other on and off), we include a conditional term to each differential equation. For a cell c, a single artificial gene for a state variable i has the form:

$$\textbf{if } \mathsf{Condition}_c(state_c, env_c) \textbf{ then } \frac{dstate_c[i]}{dt} = \mathsf{Consequent}_c(state_c, env_c)$$

Several artificial genes may code for the same artificial protein, and their contribution is summed compute the total change in state. We implement the conditions as continuous functions from which vary from zero to one in a sigmoidal fashion, which are multiplied times the consequent to give the actual contribution. Using a continuous function avoid signalling a discontinuity every time a condition fires.

$$\frac{dstate_c[i]}{dt} = \sum_S (\mathsf{ContinuousCondition}_c^S(state_c, env_c))(\mathsf{Consequent}_c^S(state_c, env_c))$$

The state of a cell determines its behavior via the *cell behavior functions* (section A.2.2) 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 use a mathematical expression (the *cell behavior function*) to describe how a protein or group of proteins acts to perform some function, eg: produce a locomotor force, release a diffusable chemical, adhere to another molecule, etc.

Cells also exhibit discontinuous behaviors (events), such as cell division, emitting a growth cone, and dying. We determine the timing of each event with a cell behavior function; when the function crosses zero, the specified event occurs (cell divides, emits a growth cone, or dies).

The continuous and discontinuous behaviors of the cells form a system of piecewise continuous ordinary differential equations [Barzel]. These 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 (eg. during cell division or cell death). Thus the solver must also do root finding on the event functions while it is integrating the ODEs forward in time.[5]

The testbed supports a broad range of experiments, in which a simulated organism is allowed to undergo development according to its artificial genome and the given environemental conditions. An experiment is described in a file containing the cell state equations (artificial genome), cell behavior functions, parameter settings, boundary conditions, and initial state (an example file appears in Appendix B). Thus the user can change both the state equations (genome) and the behavior equations. In practice, we tend to change only the genome and use the same behavior functions between experiments.

5 Results

Snapshots from the animations generated by the testbed are shown in the Figures 1-6 (color plates). Two of them are also included in black and white with the text, for convenience (Figures 5d and 6d). These simulations show the ability of our testbed to exhibit some basic multicellular behaviors. The cells are

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

represented as circles, and the diffusing chemicals are rendered as graded colors. Neurites are shown as white lines (in Figure 6).

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

Figure 1: Neurite path finding. In this experiment, a growth cone from a cell on the left climbs the gradient of the red chemical, 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 2: Cell differentiation. The initial cell divides under the control of a cytoplasmically inherited factor (a protein which the cell inherits from its parent). When this is diluted beyond a threshold, the cells stop dividing, and begin to emit diffusable chemicals. One of the chemicals is a fast-diffusing 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, as in Meinhardt's work [Meinhardt]. Those turned 'on' are emitting both red and green. The other cells' activity is suppressed by the inhibitory green chemical. Initial conditions and environmental differences account for which cells are selected, creating a heterogeneous population.

Figure 3a-b: Cyclic Behavior. Size regulation (creating and maintaining the proper number and types of cells) is an important function in multicellular organisms. In every experiment we conduct, we must deal with size regulation, avoiding explosive non-terminating growth (which simply fills our simulated petri dish).

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 Figures 3a and 3b, we see an attempt to regulate size using diffusable chemicals. Instead of regulating the size, this experiment unexpectedly resulted in cyclic behavior. There were initially two cell types: one emits and seeks red, the other emits and seeks green. Growth and splitting for each type limited by the presence its respective chemical. The amount of chemical emission is regulated by the presence of the chemical. This specification forms clusters,h 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 lead the cells out from the center, forming a ring. The ring pattern is not stable either, and the cells tend to cluster together again, cyclically.

Figure 4a-d: Chains of Cells. 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 due to a diffusable chemical. The cells are held together by the adhesive force, but are attempt to move away from each other due to the repulsive 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, so the ones at the end of a chain tend to divide more frequently. This also serves to regulate the size of the chains. (The interpreted code for this experiment appears in Appendix B).

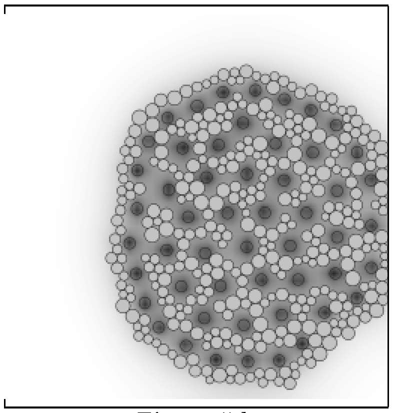
Figure 5d.

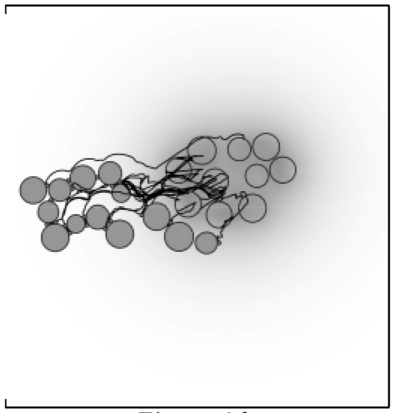
Figure 6d.

Figure 5a-d: Skeleton. This sequence begins with a single cell which divides for several generations. These first cells emit a diffusing chemical (shown as yellow in the color figure), and move about until they sense at a certain level of the chemical. This forms a pattern of roughly equally spaced cells. When this pattern is stable, some of the cells change state and begin to exhibit the chaining behavior (as in Figure 4). The second wave cells avoid large concentrations of the (yellow) diffusing chemical, but are attracted to it if the concentration is too small. Thus they intercalate between the original cells, but do not move to far from the cluster.

Figure 6a-d: Network. Two cells of different types give rise to a small network in this simulation run. One cell divides a few times to create a small cluster of green emitting cells. At a later time (determined by the dilution of an inherited factor), these cells emit growth cones[6] which seek the red chemical. Meanwhile, the other cells have been dividing and emitting the red chemical, and are waiting to be contacted by the growing neurites.

When a growth cone initially contacts a red-emitting cell, it adheres via a certain surface factor (an artificial membrane protein). Upon recognizing the contact, both the cell and the growth cone undergo a state change. They both begin to express a different adhesive surface factor, and stop expressing the first. In addition, the cell stops emitting red, and the growth cone stops moving. Thus the cell will no longer attract growth cones (since it has stopped emitting red), and later arriving growth cones will no longer adhere to it (since it has stopped expressing the first surface factor).

Growth cones which have not made contact continue to search for red-emitting

[6]A growth cone is a structure at the tip of a growing neurite, which is "the driving force behind neurite extension."[Purves et al].

cells. Eventually, the growth cones which fail to find a target will die, as will their neurites. The process of excess neurites dying off can be seen in the difference between the number of white neurites in figures 6c and 6d. This "pruning" of excess neurites is a phenomenon observed in biological networks[Purves et al, Brown et al].

6 Discussion

6.1 What have we learned from these experiments?

◇ Size regulation (creating and maintaining the size and shape of a multicellular organism) is critical, and non-trivial.
◇ Using the combination of multiple mechanisms to specify a pattern can be more robust than using a single mechanism.
◇ Specifying cell lineage (via cytoplasmically inherited factors) is a useful way of describing a developmental pattern (this is the mechanism used by L-systems, section 2.1).
◇ It is difficult to design an artificial genome which develops into a particular pattern, or conversely, to predict the pattern that a particular artificial genome will create.
◇ Developmental models are adaptive (robust). They tend to generate similar patterns under perturbations of environment or initial conditions.

Size Regulation. 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 two methods:
◇ using a threshold on the concentration of a diffusable chemical to turn cell division on and off, and
◇ using cytoplasmically inherited factors. Cells keep subdividing 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 (as in Figures 4, 5 and 6).

Relation to L-systems Control via cytoplasmically inherited factors is an effective way to regulate a developmental process. The combination of asymmetric cell division and inherited factors can emulate L-systems ([Prusinkiewicz & Lindenmayer], section 2.1). Each cell division is like a production rule ($parent \rightarrow child_a \ child_b$), with the children differing as specified by the asymmetric cell division. As discussed in section 2, this approach can be seen as a model at a lower level of abstraction than that of L-systems. Asymmetric cell division was not used in the simulations shown in this paper, however we did use inherited factors to control cell division and state changes.

Hard to make the genomes by hand As we mentioned in the Results section (5), it is difficult to construct by hand an artificial genome which will give rise to a particular pattern. This is not surprising, since we are writing a specification in a very indirect language, and the resulting simulated organism will be affected by many factors during its development. However, it is not our intention that the system be easily programmed by humans. A more pertinent issue is whether the artificial genome will be appropriate for simulated evolution.

Is this genome representation appropriate for simulated evolution? The experiments reveal that developmental models adapt well to changes in environment and initial conditions. It seems likely that that they will also have robustness with respect to genomic variation (as mentioned in the introduction). Hence our initial experiments lead us to believe that our artifical genome is well suited to simulated evolution. We have started performing simulated evolution on these developmental models, and will report those results when they are complete (the preliminary results seem promising).

6.2 Useful Techniques for Developmental Simulation

During the construction of the testbed, we focused on getting qualitatively biological behavior while aiming for computational efficiency. The following techniques were effective in achieving this balance:

- ◇ Artificial Genome: (sections 4 and A.2.1)
  - ◇ 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 one sensor with a time-varying location instead of many fixed sensors (section A.3.1)
- ◇ use simplest stable solver (adaptive euler solver, section A.6)
- ◇ use viscous dynamics ($F = kv$) for cell motion (section A.2.3)
- ◇ use penalty method for sloppy collisions (section A.3.2)

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. 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 more nutrient. We incorporate noise by adding it in at the cell's sensors (user can specify the amount of noise).

7 Conclusion

We have presented a developmental model which captures sufficient biological detail to produce patterns with asymmetric structure and heterogeneous cell types. In fact, asymmetry is the rule rather than the exception. Yet in figure 5, we see a fairly regular global pattern which can arise despite local disorder. Heterogeneity is also evident in the multiple cell types.

The modeling framework is based on several mechanisms known to be important in biological development (chemical[Turing], mechanical[Odell et al]. electrical[Fraser & Perkel], and genetic/cell lineage[Lindenmayer]). The interaction between mechanisms can lead to heterogeneous patterns; also, patterns can be redundantly specificied via different mechanisms.

This is work in progress towards a long range goal of performing simulated evolution on developmental models. The experiments performed to date suggest that developmental models can be robust to changes in the environment or initial conditions. Despite perturbations, they tend to form similar patterns. This property is likely to be important for simulated evolution, where adaptation to small changes in the genome may help the organism survive mutations which lead to new structures.

8 Acknowledgements

The authors would like to thank all members of the Caltech Graphics Group for collaboration, support, and encouragement. Some of the code used in the testbed was written by Ronen Barzel, Tim Kay, David Laidlaw, Mike Meckler, John Snyder, and Adam Woodbury. Thanks also to Dan Fain, David Kirk, David Laidlaw, Erik Winfree, and an anonymous reviewer for proofreading and helpful suggestions.

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/DARPA STC for Computer Graphics and Scientific Visualization. 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.

A Appendix A: Detailed Implementation

We begin with definitions of variables and objects, then proceed with an explanation of the equations which control the behaviors of the cells and environment. The last two subsections describe the implementation of growth cones and neurites, and the numerical solution methods.

- ◇ Definitions
- ◇ Cell Equations (State Equations, Behavior Functions, Equations of Motion)
- ◇ Environment Equations (Diffusion, Collision)
- ◇ Model of Membrane Proteins and Cell Adhesion
- ◇ Neurites and Growth Cones
- ◇ Numerical Computation

A note on the protein model

The model was initially implemented as described in section 4. Since the cell state variables encoded protein amounts, they were constrained to be greater than or equal to zero. However, we found that our cell state equations and cell behavior functions often used a difference of the state values to create a signed value ($state[i] - state[j]$). This suggested a computational speedup by doing a change of variables, and allowing our state variables to represent the difference of a *pair* of proteins. This removes the constraint that the state variables be non-negative, and reduces the number of variables by half.

A.1 Definitions

cell 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*

continuous cell behaviors Cells exhibit several continuous behaviors, determined by the cell behavior functions (section A.2.2):

- ◇ 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)

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 timing of events is determined by cell behavior functions which are described in section A.2.2 below.

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

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.

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 ($\mathbf{env}_c$[ ]) An array of variables which represent the local environment of a cell. The values 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.

Local Environment Variables

The local environment of a cell can be accessed via the array of local environment variables. These include:

- ◇ amount and gradient of diffusable chemicals at a local sensor (with noise)
- ◇ amount of cell membrane proteins in a bound state (for detecting contact with other cells – see section below on Model of Membrane Proteins and Cell Adhesion).
- ◇ cell size

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. A cell does not have access to its absolute location, but it does know its size.

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 or non-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 2d gradient ($\partial chem_i / \partial x, \partial chem_i / \partial y$).

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 A.4 below for details).

At present the local environment array looks like:

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

A.2 Cell Equations

The cell state variables indirectly control a cell's behavior within its environment. This is accomplished via three categories of functions, the *cell state equations* (genome), *cell behavior functions* (protein structure to function model), and the *cell equations of motion* (see Diagram 1).

cell state equations (the genome): The state equations modify the cell's internal state variables based on the local environment and the cell's current state.

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.

cell equations of motion: given the cell's behavior functions which describe what the cell is *trying* to do, these equations will compute the end results. For example, the cell may be applying forces to move right, but the collision forces may counteract that movement, producing a net movement to the left or right.

Cells are only able to access local information such as the local concentration of chemicals (see **Local Environment Variables** section A.1). 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 (eg. collision with a wall).

A.2.1 Cell State Equations (the genome)

The cell state equations are the model of the cell's genome (see section 4). These equations encode how the cell changes state based on its local environment and its current state. The state equations implement a crude model of protein sythesis. In biological cells, a gene encodes a protein. In our model, a conditional differential equation determines the change of a variable related to protein amount. (Examples of cell state equations appear in Appendix B).

This artificial genome was designed to be amenable to simulated evolution. We focus on the following properties:

- ◇ allow regulation of genes by other gene products (to switch on and off single equations)
- ◇ allow groups of genes to be regulated together (to switch on and off groups of equations)
- ◇ if there are multiple genes for the same protein, just make more of it

The conditional element models the regulation of a gene or group of genes, enabling us to switch on or off groups of equations based on the state of the cell or its local environment.

It is possible to have multiple contributions to the same state variable (multiple genes for the same protein). The differential equation for state variable $state[i]$ is formed from the contribution of all of the consequents pertaining to that state variable. The condition is implemented as a continuous function (rather than a zero to one step), which is multiplied times the consequent. This is both more "biological" (rates of protein production turn on and off with some probability), and more efficient (no need to do root finding for the exact time of the condition changing).

$$\frac{dstate_c[i]}{dt} = \sum_S (\mathsf{ContinuousCondition}_c^S(state_c, env_c))(\mathsf{Consequent}_c^S(state_c, env_c))$$

A.2.2 Cell Behavior Functions

The behavior functions compute the cells' attempted behaviors based on their current state. Both continuous and discontinuous behaviors are handled; continuous behaviors are simply continuous functions of the state variables (eg. MotiveForce, GrowthForce, Spew), and discontinuous behaviors are events triggered by the zero-crossing of a behavior function (eg. TimeToSplit, TimeToDie).

This example illustrates the behavior functions used to compute the simulations shown in section 5. These may be changed to arbitrary C mathematical expressions which depend on the state and local environment.

$$\begin{aligned}
\mathsf{MotiveForce}_c(state_c, env_c) &\equiv (state_c[0], state_c[1]) \\
\mathsf{GrowthForce}_c(state_c, env_c) &\equiv state_c[2] \\
\mathsf{TimeToSplit}_c(state_c, env_c) &\equiv \min(\mathrm{thresh}(\mathrm{r}_0, env[\mathrm{radius}]), \\
&\qquad \mathrm{thresh}(\mathrm{split}_0, state_c[\mathrm{split}])) - 0.5)
\end{aligned}$$

The definition of TimeToSplit() is a continuous version of the condition ((radius $> r_0$) & ($state_c$[split] $> s_0$)), ie. a cell splits when it is large enough (bigger than r_0) and has accumulated enough of the $state_c$[split] protein (more than s_0). The other event functions TimeToDie() and TimeToEmitNeurite() are defined similarly.

For each chemical $a \in 0, 1, \ldots, nchems$, this function defines the amount of a diffusable chemical a being emitted into or absorbed from the environment by cell c. In the simulations shown, the rate at which chemical a is emitted by the cell is determined a single state variable $state_c[i_a]$, where i_a is an index into the state array.

$$\mathsf{Spew}_{a,c}(state_c, env_c) \quad \equiv \quad state_c[i_a]$$

As mentioned before, this an example of the equations used in the simulations shown; other functions of the state and environment variables can easily be defined. For instance, a useful alternative to defining the components of the motive force directly via state variables is to use state variables for the magnitude and direction of motion, then transform them to obtain the xy components of the force: $\mathsf{MotiveForce}_c(state_c, env_c) \equiv (state_c[0] \cdot \cos(state_c[1]), state_c[0] \cdot \sin(state_c[1]))$.

A.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 $F = kv$ with the applied forces ($k = k_{drag} \times \text{area}(c)$). The CollisionForce and AdhesionForce are computed in the global environment, and are described in detail below (under Environment Equations).

$$\begin{aligned} \sum_{\mathsf{forces}} &= \mathsf{CollisionForce}_c + \mathsf{AdhesionForce}_c + \mathsf{MotiveForce}_c - kv \\ &= 0 \end{aligned}$$

Unlike the drag force, the collision forces do not depend on velocity (section A.3.2). 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.

A.3 Environment Equations

A.3.1 Diffusion

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

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

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:

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

where the location of cell c is (c_x, c_y), and $\delta(x, y)$ is the Dirac delta function (a spike at $x = 0, y = 0$).

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.

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 the randomly moving sensor to be the most effective of the three. The randomly moving sensor computes a 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 sensor and source which are both located at the center of the cell.

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 A.4.

The diffusion equation for moving point sources/sinks can be solved in closed form using an integral expression. It is possible that this approach could speed up the computation, although the dependence of the integral on the history of cell movement may make this undesirable. Also, using a closed form solution would put more restrictions on the extracellular reaction function R().

A.3.2 Collision

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

In the case of colliding circles, the forces are very simple, but more complex collisions are possible within our framework. 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. Then we have:

$$\mathsf{CollisionForce}_c = \sum_{b \in objects} k \ (\mathsf{d}(b,c) - \mathit{offset})^n \ \mathbf{p}$$

This method of collision computation is somewhat inaccurate and requires choosing parameters k, n, and *offset* arbitrarily. However, accurate collisions are not critical to our application, and the cells we are modeling are not rigid objects so some amount of overlap is acceptable. In practice, object interpenetration has only occasionally been troublesome, and the penalty method has been computationally efficent when combined with our solution methods (section A.6).

A.4 Model of Membrane Proteins 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)
◇ cell adhesion (cells physically bind together)

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

A single state variable z_j directly controls the amount a given surface factor in the cell's membrane. The surface factor which is bound to its complement on an adjacent cell is reported in the mem_j variables in the local environment array. This suffices for cell contact recognition; cells can determine that they are in contact with another cell that expresses a certain surface factor. For cell adhesion, there is a force computed which depends on the size of the region of contact, and the amounts of surface factors available on the contacting cells.

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, not 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.

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 amount of the complementary factors in their membranes. Let ($\mathsf{Conc}_c(a)$ return the concentration of a on cell c). For a particular surface factor a with complement a', we have:

$$\mathsf{AmountBound}_c(a) = \sum_{b \in \text{cells}} \mathsf{ContactArea}(c, b) \times \mathsf{Conc}_c(a) \times \mathsf{Conc}_b(a') \times p_{bind},$$

where p_{bind} is a multiplicative constant that roughly corresponds to the probability that two proteins a and a' will bind. $\mathsf{ContactArea}(c, b)$ estimates the contact area between two cells as the chord length of their overlap. The recognition factors report $\mathsf{AmountBound}$ in the appropriate local environment variable (the mem_i mentioned in section A.1).

The adhesion force on a cell is the sum of the forces from all the adhesive surface factors on all of the cells in contact:

$$\mathsf{AdhesionForce}_c = \sum_{j \in \text{adhesion factors}} \mathsf{AmtBound}_c(j) * stickness_j$$

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 directly measure/change the chemicals in the local environment (discussed above in the **Diffusion** section, A.3.1). This is done for efficiency.

A.5 Neurites and Growth Cones

We model growth cones as small cells which are connected to the parent cell by a neurite. They have the same capabilities as cells, 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. 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 (Figure 6). 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.

A.6 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]. 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 step-size, 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.

The solver rejects differential equation steps that do not meet the following criteria:

- ◇ per ODE step, cells cannot move more than a specified distance (typically 5% of a cell diameter). This prevents tunneling of cells 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.
- ◇ diffusion concentrations cannot change by more than a specified amount per ODE step on the grid elements. Thus, we avoid wasting computation time at near-zero concentrations. (This can cause some oscillation of values at low concentrations in the diffusion grid.)

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.

Another reason for using the adaptive Euler solver is that it copes better with discontinuities such as random noise added to the system. More sophisticated solvers do not handle this as well.

B Appendix B: Example Experiment File

This is an example of the interpreted file used to generate the simulation shown in Figure 4.

```
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);
```

References

[Agarwal] *The Cell Programming Language*, presented at Symposium on Pattern Formation, Feb 12, 1993, Harvey Mudd College, Claremont, CA.

[Bard] Bard, Jonathan B. L., *A model for generating aspects of zebra and other mammalian coat patterns,* Journal of Theoretical Biology, 93:501–531, 1981.

[Barzel] Barzel, Ronen, *A Structured Approach to Physically-Based Modeling,* Ph.D. Thesis, Caltech, 1992. Also: *Physically-Based Modeling: A Structured Approach*, Ronen Barzel, Academic Press, Cambridge MA, 1992.

[Brown et al] Brown, M.C., Hopkins, W.G., and R.J. Keynes, *Essentials of Neural Development*, Cambridge University Press, 1991.

[de Boer] de Boer, Martin, *Analysis and computer generation of division patterns in cell layers using developmental algorithms*, PhD thesis, University of Utrecht, Nov 20, 1989.

[de Boer, Fracchia and Prusinkiewicz] de Boer, Martin, Fracchia, David and Prusinkiewicz, Przemyslaw, *Analysis and Simulation of the Development of Cellular Layers*, Artificial Life II, Addison Wesley (1991), pp 465–483.

[Carr & Konishi] Carr, Catherine E., and Konishi, Masakazu, *Axonal delay lines for time measurement in the owl's brainstem*, Proc. Natl. Acad. Sci. USA, Volume 85, pp 8311-8315, Nov 1988 (Neurobiology).

[Fleischer] Fleischer, Kurt W., *Generating Artificial Neural Networks using Simulated Evolution of a Developmental Model*, Caltech PhD Thesis, expected 1993.

[Fraser & Perkel] Fraser, Scott and Donald Perkel, *Competitive and Positional Cues in the Patterning of Nerve Connections*, J. Neuro. **21** 1: 51-72 (1990).

[Hebb] Hebb, Donald O., *The organization of behavior*, Wiley (New York, 1949).

[Hopfield] Hopfield, J. J., *Neurons with grade response have collective computational properties like those of two-state neurons*, 1984 Proceedings of the National Acadademy of Science, USA, **81** 3088-3092.

[Kitano] Kitano, Hiroaki, *Designing Neural Networks Using Genetic Algorithms with Graph Generation System*, Complex Systems **4**:461-476, 1990.

[Lin & Segel] Dictyostelium (slime mold) modeling has been described by many authors, among them Lin, C.C., and Segel, L. A. *Mathematics Applied to Deterministic Problems in the Natural Sciences*, SIAM, Philadelphia (1988).

[Lindenmayer] Lindenmayer, Aristid, *Mathematical Models for Cellular Interaction in Development, Parts I and II,* J. Theo. Bio 18, 280-315 (1968).

[Lindenmayer & Prusinkiewicz] Lindenmayer, Aristid, and Prusinkiewicz, Przemyslaw, *Developmental Models of Multicellular Organisms: A Computer Graphics Perspective,* Artificial Life 1221-249 (1989)

[Meinhardt] Meinhardt, Hans, *Models of Biological Pattern Formation*, Academic Press Inc (London) Ltd 1982.

[Mjolsness et al] Mjolsness, Eric, Sharp David and Reinitz, John, *A connectionist model of development,* Journal of Theoretical Biology (1991), **152**, 429-453.

[Murray] Murray, J. D., *On pattern formation mechanisms for lepidoptera wing patterns and mammalian coat markings,* Philosophical Transactions of the Royal Society (B), 295:473–496, 1981.

[Odell et al] Odell, Garrett M., Oster G., Alberch P., and B. Burnside, *The Mechanical Basis of Morphogenesis*, Developmental Biology **85**, 446-462 (1981).

[Oster] Oster, G., Weliky M., and Minsuk, S., *Morphogenesis by Cell Intercalation*, 1989 Lectures in Complex Systems (SFI Volume II, ed: Erica Jen), Addison-Wesley, 1990

[Platt] Platt, John, *Constraint Methods for Neural Networks and Computer Graphics,* Ph.D. Thesis, California Institute of Technology, 1989.

[Prusinkiewicz & Lindenmayer] Prusinkiewicz, Przemyslaw and Lindenmayer, Aristid, *The Algorithmic Beauty of Plants*, Springer-Verlag, New York, 1990.

[Prusinkiewicz, Hammel & Mjolsness] Prusinkiewicz, Przemyslaw, Hammel, Mark and Mjolsness, Eric, *Animation of Plant Development using Differential L-systems*, to appear in Computer Graphics **27**, (ACM Siggraph July 1993).

[Purves et al] Purves, Dale, and Lichtman, Jeff W, *Principles of Neural Development*, Sinauer Associates (1985), Sunderland, MA.

[Reinitz et al] Reinitz, John, Mjolsness, Eric, and Sharp, David H., *Model for co-operative control of positional information in* Drosophila *by* bicoid *and maternal* hunchback*,* Yale University Research Report YALEU/DCS/RR-922 or Los Alamos Unclassified Report 92-2942, Sept 1992.

[Sims91a] Sims, Karl, *Interactive Evolution of Dynamical Systems*, Proceedings of the First European Conference on Artificial Life, Paris, 1991 (publ by MIT Press).

[Sims91b] Sims, Karl, *Artificial Evolution for Computer Graphics*, Computer Graphics **25**(4), pp 319–328, (ACM Siggraph 1991).

[Snyder] Snyder, John, *Generative Modeling: An Approach to High Level Shape Design for Computer Graphics and CAD,* Ph.D. Thesis, California Institute of Technology, 1991.

[Stork et. al.] Stork, David G., Jackson, Bernie, and Walker, Scott, *Non-Optimality via Pre-adaptation in Simple Neural Systems*, Artificial Life II, Addison-Wesley 1992.

[Turing] Turing, Alan, *The Chemical Basis of Morphogenesis*, Phil. Trans. B. **237**, 37-72 (1952).