Cell-based Computer Models in Developmental Biology

Pankaj Agarwal
September 1993

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Computer Science
New York University

Approved: _____

Jacob T. Schwartz
Research Advisor

Acknowledgements

This thesis would be incomplete without acknowledging the intellectual and moral contributions of mentors, friends, and family: my advisor, Jacob T. Schwartz for the introduction into the world of cells; Jerome K. Percus for encouragement and patient listening; Stuart A. Newman for his biological insight; Charles S. Peskin for his inspiring enthusiasm; and thesis committee members Benjamin Goldberg and Zvi M. Kedem for their interest.

I am indebted to my parents, Manjula Agarwal and Shivraj Agarwal, and grandparents, Satyavati Bansal and Shyam Sundar Lal Bansal for their unflinching trust and moral guidance, and to my sister Parool Agarwal for her love and support.

I am especially grateful to Laureen Treacy for her friendship, smile, dedicated proofreading, and instruction in the art of writing. I would also like to thank Michael Hind for encouragement and proofreading; Ajai Kapoor for his moral support and intriguing arguments; Alok Jain, Sushant Patnaik, Kabir Dutta, H. Ramesh, Simi Kedia, Joy Chhugani, Pushkal Kumar Pandey, Laurel Cooley, Madhusudan, and Manish Gupta for countless memorable occasions.

Abstract

In developmental biology, modeling and simulation play an important role in understanding cellular behavior. We suggest a simple language, the *Cell Programming Language* (CPL), to write computer programs to describe this behavior. Using these programs, it is possible to simulate and visualize cell behavior.

A genome is the program for the development of an organism. The genome, in conjunction with the environment, determines the behavior of each cell of the organism. The program for each cell (written in CPL) plays the role of its genome.

The program for an individual cell consists of a set of states. In each state, rules are specified which determine the cell properties (i.e. shape, motility, concentrations of various molecular species, etc.). Different states of the same cell signify different phases in the cell's life. Each cell has a tissue type associated with it. Cells of the same tissue type execute the same CPL program.

We use the discrete time simulation model. At every time step, each cell executes all the instructions in its present state sequentially. All cells are assumed to be executing in parallel, with synchronization performed after every time step.

The cells are two-dimensional. Each cell has a physical location comprising a collection of discrete connected points. This physical presence imparts to the cells the attributes of area, perimeter, and neighbors (other cells). The neighbor attribute forms the basis for all intercellular communication.

The language contains features for specifying:

- the location, area, and shape of the cells;
- the concentrations of various chemicals in each cell, the equations of their catalysis, and diffusion;
- the direction and speed of cell motion;
- the rates of cell growth and division;
- cell differentiation: the evolution of cell behavior during its lifetime.

We have employed CPL to model the following: aggregation in cellular slime mold in response to a chemotactic agent; the formation of skeletal elements in the vertebrate limb; and cellular segregation due to differential adhesion.

Contents

List of Figures

List of Simulation Images

List of CPL Programs

Chapter 1

Biological motivation

Developmental biology is the study of the process by which a single cell develops into an adult organism. A basic tenet of developmental biology, and for that matter of all science, is that structure and simplicity prevail at some level of organization. Though various phenomena may appear exceedingly complex, at some level of organization structure can be discovered. The processes involved in embryonic development are extremely complex and involve a great number of physical and chemical transformations which are not well understood at present. Amazingly the developmental processes and stages are strikingly similar in the entire animal kingdom (from hydra to humans). This similarity provides hope of deciphering the developmental plan. We have all wondered as to how a single small cell gives rise to such a complicated body plan with a myriad of tissues (neurons, muscle, cartilage, skin, etc.) and organs (heart, stomach, kidney, etc.) all organized intricately. An early school of thought believed that the sperm contained a miniature human fetus inside its head, and growth into an adult was by simple expansion. We now realize that the cell contains only a blueprint (genome) of the body plan. This blueprint unfolds with time, and in conjunction with the environment, physical laws and chemical laws gives rise to an adult organism.

All the cells of an organism are created due to repeated division from a single ancestor cell. While they divide they retain the genetic information; however, they come to occupy different regions. The cells determine their own function based on the genetic plan and their environment.

Cells are about halfway in the biological hierarchy, and constitute an important structure in the understanding of development. Much has been learned about the biochemistry of cells. It was realized that organisms are comprised of cells. All cells are basically similar. Schleiden and Schwann are credited with the first sound formulation of cell theory. They believed the cell to be a bag full of chemical substances that interact according to the laws of physics and chemistry. Today the cell is known to be a highly complex, but structured,

entity. Cells contain cytoplasm and a nucleus. Inside the cytoplasm are membrane-bounded organelles including the mitochondria. The nucleus carries the genetic material in the form of chromosomes. The chromosomes contain DNA (deoxyribonucleic acid). The DNA macromolecules are composed of four types of nucleotides: adenine (A), cytosine (C), guanine (G), and thymine (T). DNA is a double stranded structure with each strand composed of a sequence of these nucleotides joined together by phosphodiesterase molecules. Triplets of these nucleotides specify an amino acid. These 64 (4^3) possible triplets specify just 20 different amino acids (some amino acids are specified by more than one triplet). These amino acids combine to form proteins which are half the dry weight of a cell. The amino acid sequence determine both the three dimensional structure and function of the protein. Much of the cell biochemistry revolves around proteins. The molecular mechanisms of development are only now being discovered and are responsible for renewed excitement in the field.

We will not attempt to summarize developmental biology. Such summaries can be found in any introductory text on biology and in great detail in Gilbert's book on developmental biology [Gil91]. We instead consider a few developmental processes (in which the constituent cells play major roles) to provide motivation. In particular, we consider the following phenomena.

Slime mold aggregation

Dictyostelium discoidea[1] is a free-living amoeba and is considered by some to be the hydrogen atom of developmental biology. *Dictyostelium* is a bridge between unicellular and multicellular organisms, since *Dictyostelium* cells spend portions of their life in each mode. In their unicellular existence, they eat bacteria and reproduce by binary fission. Exhaustion of food supply causes tens of thousands of these amoebae to join together, to form moving streams of cells that converge into conical mounds. These conical aggregates modify their shape to form a slug (worm-like structure). The slug migrates in search of better environmental conditions, where the cells in the slug differentiate into stalk cells and spore cells, which together form a fruiting body. The spore cells disperse, each one becoming a new amoeba. This is a simple, yet intriguing, life cycle and a rich source for problems in development.

The cellular aggregation is randomly initiated. Cells do not move directly towards these random centers; rather, they join with each other to form streams; the streams converge into larger streams, and eventually all streams merge in the center. This motion has been shown to be due to chemotaxis[2], the chemical involved being cyclic adenosine monophosphate

[1] Also referred to as the cellular slime mold.

[2] Chemotactic movement is caused by the diffusion of chemical substances through a medium. Cells may detect and move along such chemical gradients.

(cAMP). There is no *dominant* cell or predetermined center; whichever amoebae happen to secrete cAMP first become aggregation centers. Other amoebae respond to the cAMP by initiating movement towards the cAMP source and by releasing cAMP of their own[3]. In chapter 5, we explore this aggregation in detail.

Limb skeleton formation

The organization of the bones in the limbs is regarded as a classical example of development. What kind of process would create an increasing number of progressively smaller bones in the limb along the proximo-distal axis (from the shoulder to the fingers): a single humerus; the radius and the ulna; and the five metacarpals-digits? This basic theme plays out in almost all vertebrates with minor differences. In addition, variations are observed often enough in the number of digits to conclude that the number of these bones is not preordained, but realized due to some process at a certain stage of development. In chapter 6, we explore this subject in greater detail.

Sponge reconstitution

Sponges are simple protozoans which possess a remarkable property. Wilson in 1907 observed that if a sponge is dissociated into its individual cells by passing through a sieve, the cells reaggregate to form a *functional* sponge [Wil07]. This reconstitution is species-specific; if cells from sponges of different species are mixed together, each of the reformed sponges contains cells only from its own species. This helped prove that cells can recognize other cells of their own kind. Later experiments by Moscona, Holtfreter, and Steinberg have established that cells from other organisms also recognize cells of their own kind, and tend to segregate when mixed with other cells. These experiments established that cells have an identity, and it became possible to visualize development in terms of the constituent cells. In section 3.2, we examine a model which attempts to explain this phenomena, and in chapter 7, we employ simulations to explore the theories in greater detail.

Thesis Organization

The preceding remarks refer to **Chapter 1: Biological motivation**. The first half of the thesis is organized as follows. **Chapter 2: Introduction to the model and language** provides an overview of *Cell Programming Language* (CPL). **Chapter 3: Cellular models** presents and explains some of the relevant models of developmental behavior that have influenced the design of CPL. **Chapter 4: Details of the Cell Programming Language** presents the instruction set of the CPL, along with some implementation details.

[3]This summary has been paraphrased from Gilbert [Gil91].

The second half of the thesis contains a discussion of the biological problems that we have modeled and explored using programs written in CPL, along with the simulation results obtained by running those programs. This includes **Chapter 5: Aggregation in slime mold**; **Chapter 6: Limb skeleton formation**; **Chapter 7: Segregation of tissues**; and **Chapter 8: Assorted applications**. **Chapter 9: Conclusions** presents an overall analysis of the thesis, including a variety of open problems. The grammar for CPL is included in the appendix.

Chapter 2

Introduction to the model and the language

Cells communicate with each other and organize themselves in useful patterns to exhibit extraordinary developmental behavior. The genome inside the cells plays a major role in directing cellular behavior. This thesis explores the possibility of describing developmental behavior in simple terms. The success of our attempt would show that it is possible to write short programs describing cellular behavior. It is our hope that most cellular behavior is similar enough that it will be possible to encode developmentally crucial phenomena using a simple programming language described below. These considerations motivate the design of the *Cell Programming Language (CPL)*. The idea of encoding each cell by a sequence of instructions has been proposed in various forms by Gordon [Gor66], Odell et al. [OOAB81] and Schwartz [Sch88]. Our proposed solution is inspired by these ideas, and also provides an implementation.

2.1 Structure of CPL programs

A CPL program for a single cell consists of a set of states. In each state, rules are specified which determine the cell properties (i.e. shape, motility, concentrations of various molecular species, etc.). Different states of the same cell signify different phases in the cell's life. Thus, at some point of time a cell could be in a state awaiting a signal, and once it receives a signal, it enters a different state in which chemotactic movement takes place.

Each cell has a tissue type associated with it[1]. Cells of the same tissue type execute the same program.

[1] The association of a tissue type with each cell may be regarded as a technical convenience, but we feel it is an important one, which lends to the understanding of the model and the language.

We use the discrete time simulation model. At every time step, each cell executes all the instructions in its present state sequentially. All cells are assumed to be executing in parallel, with synchronization performed after every time step.

CPL provides us with a mechanism for specifying operations on cell attributes. The main cell attributes are:

- **Tissue Type:** Each cell has a specific tissue type which dictates the cell's response to its environment. The tissue type determines what program the cell executes. The program may be thought of as representing the cell's genome, the effect of the environment on the cell, and the physical chemistry of the cell constituents.
- **Biochemicals:** The concentrations of all the biochemicals present in a cell, along with their equations of catalysis and diffusion, can be specified. These concentrations may represent either the interior or the surface concentrations. Cells are modeled to be homogeneous; therefore, the biochemical concentration are uniform inside the cell.
- **Physical presence:** A cell has the attributes of area, perimeter, and neighbors (other cells). Only cells in direct physical contact are treated as neighbors. This neighbor attribute forms the basis for all intercellular communication. Cells can sense the attributes of their neighbors and react accordingly. Biochemical diffusion also depends on the biochemical concentrations in the neighbors.
- **Neighbor's attributes:** A cell can sense its neighbor's attributes: tissue type, biochemical concentrations, area, perimeter, the contact length between the two cells, as well as the direction in which that neighbor lies.

In addition to the biologically motivated attributes listed above, other attributes are required to write programs for these cells. These are variables used to store information about the cell. The variables have specific types and may either be integer, real, or direction. Direction variables store values of the form (x,y), where x,y are integers or reals. Direction variables may be used to compute and store the direction of diffusion of a biochemical. Integer or real variables may store the number of divisions, time in a particular state, etc.

To access and modify the cell attributes described above, a set of instructions is provided; these are summarized in section 2.2 and explained in detail in chapter 4.

2.2 Introduction to CPL

In this section, we outline some of the CPL instructions to provide a flavor of the language. Complete details for each instruction are contained in chapter 4.

- **assignment:** In addition to the regular assignment statements, CPL provides a special assignment for biochemicals, which effects the biochemical value only at the next time step. Such assignments are prefaced with the reserved word `deriv`.
- **if-then-else:** The if-then-else instruction provides conditional execution of instructions.
- **move:** The move instruction causes the cell to move in a specified direction by exchanging the location of the cell with that of a neighboring cell in the given direction.
- **goto**: This instruction specifies a state switch.
- **for_each_neighbor_do:** This permits the execution of an instruction (or a block of instructions) using the parameters of each of the cell neighbors in sequence.
- **with_neighbor_in_direction:** This instruction is used to employ the attributes of a single specified neighbor.
- **divide:** The divide instruction causes the area of a cell to be split up equally between two daughter cells.
- **grow:** The grow instruction causes the cell to grow in area by the given size in the specified direction.
- **roundup:** The execution of this instruction rounds up the cell by modifying the cell boundary to form a more cohesive unit.
- **die:** This results in cell death; no more instructions of this cell are executed.

This thesis explores the power and limitations of this simple instruction set.

2.3 Physical representation of cells

Physically, an actual cell is a solid which may be approximated by a polygonal structure with a specified area. It is generally many-sided and not necessarily convex. A cell changes shape, grows in area, divides into two, and moves, depending on its own state and the environment. The chosen model should permit all these operations, and above all be flexible so as to be able handle additions to the set of operations.

A majority of the models that have been designed for cells so far, including the one CPL uses, model the cell as two dimensional[2]. One such model, used by a number of researchers, treats the cell as a rigid body of fixed size and shape. This model does not

[2]Three dimensional modeling is computationally expensive.

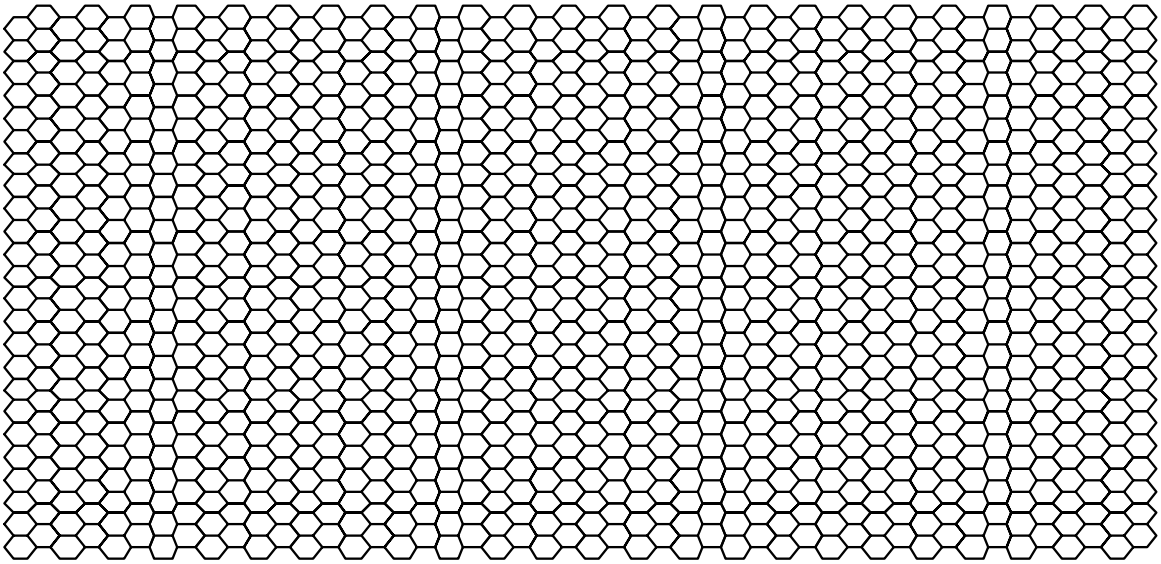


Figure 2.1: Hexagonal topology: each cell occupies one or more hexagons.

permit variable sized cells or cells with different shapes. An extension of this model, where each cell is modeled as an aggregate of a large number of discrete rigid objects, overcomes these deficiencies, as it permits cells to have arbitrary shape and size. This is the most general discrete representation. [3]

In CPL, we chose to represent each cell by a collection of discrete connected points. These points can be regarded as the points in a hexagonal lattice.[4] Figure 2.1 contains an hexagonal lattice structure. In hexagonal lattices, each point has six equidistant neighbors. A hexagon in the figure represents a lattice point. Each cell can occupy one or more of the lattice points. All the lattice points occupied by one cell should be connected i.e. a cell cannot be disjoint.

The physical representation of cells is mainly a technical issue and has not for the most part influenced the design of CPL. The representation of cells could be modified without significant alterations to CPL. The representation of cells is mostly influenced by the computational frontier. Polygonal representation of cells could be a viable computational alternative in the future.

[3] If all cells have the same shape and size, then each cell can be represented by a single point, providing an optimal representation.

[4] The hexagonal structure where each lattice point has six neighbors is better defined than the four or eight neighbor lattice structure, as the latter violate the Jordan Curve Theorem.

Chapter 3

Cellular models

A model for a real-world system is an abstract representation of the system. A computer model may be described as a sound scientific model which is amenable to computer simulation. Once a model is programmed on a computer, it becomes relatively simple to run simulations with different sets of data. In addition, if the model is not performing as well as expected, it is possible to fine-tune the model to achieve desired performance.

There is a body of literature on models in developmental biology. Pioneering work in the mathematical treatment of development is credited to D'Arcy Thompson and Waddington [Tho17,Wad66]. Ransom presented a good, non-mathematical survey of the field [Ran81]. Mostow's collection of papers involving on the *differential adhesion* hypothesis is also noteworthy [Mos75].

In section 3.1, we discuss the various generalized models of development that have been proposed including Turing's morphogen model. Section 3.2 examines Steinberg's differential adhesion hypothesis. Section 3.3 focuses on the choices various models have made regarding the physical representation of cells.

3.1 Generalized models of development

This section includes a discussion of Turing's morphogen model, which attempts to explain pattern formation; a mechanical model proposed by Odell et al., which examined cell sheet buckling; Gordon's general model; and cellular automata and lindenmayer systems, which are theoretical models of parallel computation. We also briefly describe two recent models: the connectionist model, which is used to predict the segmentation genes in the *Drosophila*; and a model that aims to grow artificial neural networks.

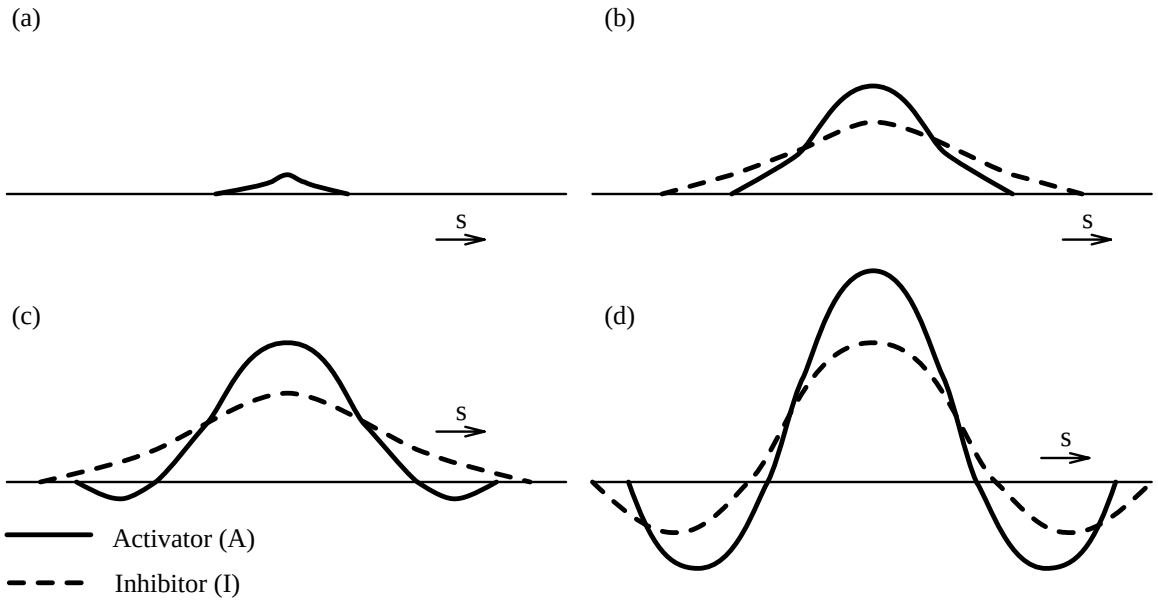


Figure 3.1: Regions of high and low activator concentration arise from a small perturbation in the initial uniform activator level. Each of the figures shows the activator and inhibitor values along the one-dimensional system. The subfigures a,b,c,d show the progression in time.

3.1.1 Turing's morphogen model

Turing, in a seminal paper in 1952, proposed a dynamic mathematical developmental model [Tur52]. He suggested that reaction-diffusion systems obeying physical laws could lead to various stable patterns. A system of chemical substances, called morphogens, reacting together and diffusing through the tissue may account for morphogenesis. Even if the morphogens have uniform concentrations, minor random fluctuation can lead to instabilities. One of Turing's findings was that under certain conditions periodic patterns arise: high and low peaks of morphogen concentration could occur, forming, for example, a series of stripes of regular width. The wavelength of these periodic patterns is determined by the diffusion constants and rates of reactions of the morphogens. Turing briefly considered real systems in terms of his model, including the *hydra* and the whorls of leaves of certain plants.

Let us consider a simple Turing system with two morphogens called the activator (A) and the inhibitor (I). The rate of change in these chemicals is given by the following equations:

$$\frac{\delta A}{\delta t} = k_a A - k_i I + \mu \frac{\delta^2 A}{\delta s^2}$$

$$\frac{\delta I}{\delta t} = k_a A - k_i I + K\mu \frac{\delta^2 I}{\delta s^2}$$

The $\frac{\delta^2}{\delta s^2}$ terms represent the diffusion in space. Consider the case:

- If k_a is positive, an increase in the concentration of the activator A leads to an increase in the concentration of both A and I.
- If k_i is positive, an increase in the concentration of I leads to a decrease in the concentrations of both A and I.
- If $K > 1$, then I diffuses faster than A.

To understand this system, let us consider a one-dimensional system with both A and I present in uniform concentrations. Figure 3.1 illustrates what happens when the activator value is perturbed upwards at a point. This may happen due to random noise. This increase in the activator A leads to a greater increase in A, and also an increase in I. But, since I diffuses faster than A it spreads out further as is shown in figure 3.1(b). Thus, the inhibitor diffuses to points in space where the activator value has not yet risen. This increase in inhibitor leads to the activator level falling below the equilibrium (c). Therefore, highs and lows of the activator are established throughout the system (d). The reactions themselves predict these extremes to be unbounded, but other physical constraints impose saturation values. In a two-dimensional system this activator-inhibitor system can lead to circular standing waves.

Meinhardt has explored in great detail various morphogen systems and the stable patterns arising from them [Mei82]. Such reaction-diffusion (activator-inhibitor) systems account for a significant share of the proposed models in developmental biology. Ouyang and Sweeney have recently discovered that stable patterns can arise in chemical systems from reaction diffusion [OS91]. These patterns formed spontaneously by varying a control parameter. They observed both hexagonal and striped patterns.

3.1.2 Mechanical model

Odell et al. presented a mechanical model to explain the folding of embryonic epithelia [OOAB81]. This was based on hypothesized properties of the cytoskeleton. They observed that if a cell that was part of a layer (or a ring) contracted it would stretch the other cells in the layer. They hypothesized that stretching cells beyond a point induced a contraction resulting in a smaller than original apical[1] surface with the volume remaining constant. This cascade effect of reduction in apical surface would cause a buckling in the cell sheet producing an invagination, which may be made to resemble gastrulation in *Sea Urchin*, ventral furrow formation in *Drosophila*, and neuralation in amphibians. Odell et al. also emphasized the importance of not assigning each cell an autonomous program of shape change:

[1] The outer surface of the epithelium is termed *apical*, and the inner is termed *basal*.

> "In this study we want to minimize both the number of complex instructions (e.g. morphogens and clocks) as well as the genetic programming required to generate morphogenetic patterns. It is, of course, conceivable that each individual cell in the blastula is genetically programmed to execute a sequence of instructions directing each movement that the cell performs as well as its precise timing. However, we regard such a view as evolutionarily implausible." [OOAB81, page 454]

3.1.3 Gordon's model

Gordon, in 1966, suggested a general model for development. According to Gordon, an organism may be regarded as an ensemble of cells, each cell capable of making decisions based on its own state and the environment. The environment would include the configuration of cells around it, and the chemical and electrical messages (surface interactions, hormones, nerve impulses, etc.) it receives. The internal state of a cell could include its state of differentiation, a limited memory, and an internal clock. A cell could take the following actions: do nothing; reset its internal clock; differentiate; communicate with other cells; divide; expand or shrink; fuse with a neighbor; move; and die.

The interactions between cells are probabilistic due to incomplete or inaccurate information about its own state and the environment, and varying initial states. Genetic control of development is indirect in this model. The genes presumably determine the next state function, albeit implicitly.

3.1.4 Cellular automata and Lindenmayer systems

A cellular automaton is a theoretical model of a parallel computer. It is an interconnection of identical cells. Each cell computes an output from local inputs. These inputs are received from a finite set of cells forming its neighborhood and possibly from an external source. Each cell houses a finite state machine, which is formally denoted by the 4-tuple (Q, Σ, δ, q_0). Q is a finite set of states, Σ is a finite input alphabet, δ is the transition function mapping $Q \times \Sigma$ to Q, and q_0 in Q is the initial state. The concept is the same as that of a finite automaton in computational theory, except that there is no final state.

A cellular automaton that allows cells to divide into daughter cells and allows the disappearance, or death, of cells is known as a dynamic cellular automaton or a *Lindenmayer* system. In other words, a Lindenmayer system is an array of interconnected automata, along with the provisions:

- The input to an automaton are the states of its neighboring automaton. Automaton on the border of the array may also have external (environmental) inputs. Therefore, $\Sigma \subset Q^k$, where k is the maximum number of neighbors providing input to the cell.

- The automaton array is not limited to its starting size. It can expand and contract by the reproduction or death of automata. This is modeled by modifying $\delta : Q \times \Sigma \rightarrow Q$ to $\delta : Q \times \Sigma \rightarrow (\epsilon, Q, Q*Q)$, where ϵ corresponds to the disappearance of an automaton and $Q * Q$ to the duplication of one.

This system is of interest to theoretical biologists as a model for the growth and development of organisms. Lindenmayer used cellular automata to grow branching and non-branching filaments, which exhibit various developmental patterns, such as a constant apical pattern, non-dividing apical zone, and banded patterns [Lin68]. He employed a one-dimensional cellular array with two neighbors for every automaton, and all the automata were identical. The state transition table for a Lindenmayer system that produces branching filaments is shown in figure 3.2. In this example, the only input each automaton takes is its own state.

Present state	1	2	3	4	5	6	7	8	9
Next state	2*3	2	2*4	2*5	6*5	7	8	9*[3]	9

Figure 3.2: State transition table for the Lindenmayer system. * indicates division. [] indicates a new branch.

Time	Filament
1	1
2	23
3	224
4	2225
5	22265
6	222765
7	2228765
8	2229[3]8765
9	2229[24]9[3]8765
10	2229[225]9[24]9[3]8765
11	2229[2265]9[225]9[24]9[3]8765
12	2229[22765]9[2265]9[225]9[24]9[3]8765
13	2229[228765]9[22765]9[2265]9[225]9[24]9[3]8765
14	2229[229[3]8765]9[228765]9[22765]9[2265]9[225]9[24]9[3]8765
15	2229[229[24]9[3]8765][229[3]8765]9[228765]9[22765]9[2265]9[225]9[24]9[3]8765

Figure 3.3: A sample calculation for 15 time steps of the automaton. A diagrammatic representation is provided in figure 3.4.

Figure 3.3 displays a sample calculation for 15 time steps of the automaton. Figure 3.4 shows a diagrammatic representation of the final state of the filament. This models the developmental pattern for a particular red algae, *Callithamnion roseum*, and has the following

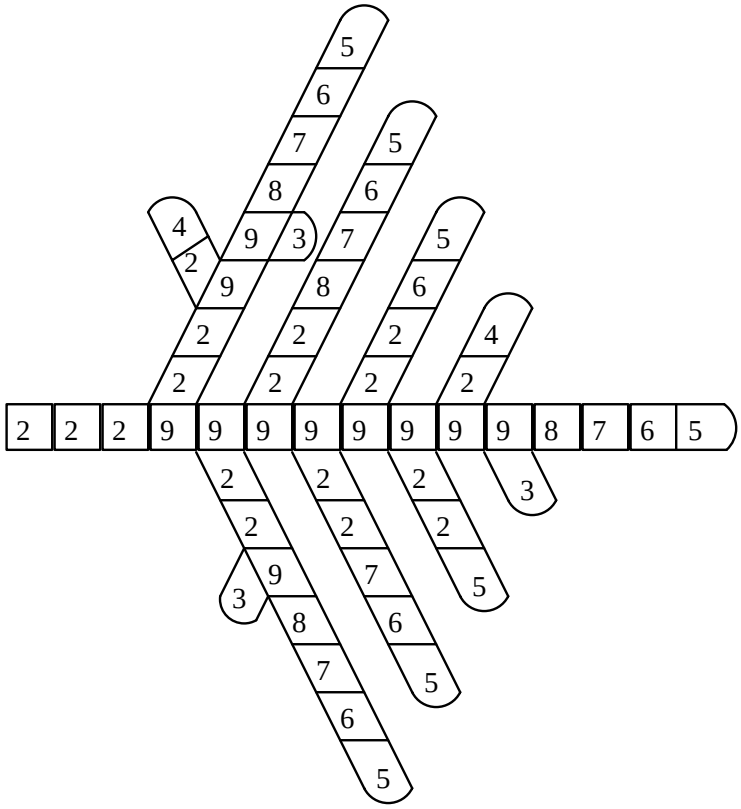


Figure 3.4: A branching filament obtained by using a cellular automaton after 15 time steps.

features:

- the main filament has at its base three cells that do not bear branches;
- each successive cell above these on the main filament bears one branch;
- in all stages four cells below the tip of the main filament have no branches;
- each primary or higher order branch repeats the pattern of the main filament.

Toffoli and Margolus have described various application of cellular automata in modeling [Tof87]. Prusinkiewicz has used Lindenmayer systems to design plants with intricate patterns [PL90].

3.1.5 Connectionist model

Mjolsness, Reinitz, and Sharp have provided a "systematic method for discovering and expressing correlations in experimental data on gene expression and other developmental processes" [RMS90,MSR91]. This *connectionist* model uses grammar rules to model state changes, and a fixed type of differential equation to model behavior within the state. The differential equations are of the form:

$$\frac{\mathrm{d}v_i^a}{\mathrm{d}t} = g_a(\sum_b T^{ab} v_i^b + h^a) - \lambda_a v_i^a$$

These equations describe the time dependence of the various protein (a) concentrations. v_i^a are the protein concentrations for object i (which may be a cell, nucleus, or synapse), g_a are sigmoidal threshold functions, T^{ab} are connectionist threshold elements, and h_a are the threshold terms. Both T^{ab} and h_a are adjusted (trained) to fit the experimental data. This allows them to compute the coupling between the various genes (which produce the proteins), and thus predict the concentration profiles for similar systems (mutants). They have tested the system on the segmentation genes in the *Drosophila* blastoderm.

3.1.6 A model for growing artificial neural networks

Fleischer and Barr have designed a simulation system for studying multicellular pattern formation [FB93]. This system has been designed to simulate evolution, so that eventually they can create artificial neural networks to solve problems. The cells have a genome represented by a set of differential equations. The differential equations code for the cell's chemical, mechanical, genetic, and electrical activity. This representation eases evolution, since cells can modify genetic information (i.e. exchange some of their differential equations) by either mutation or during cell meiosis. The choice of differential equation to model cell

activity permits fine control of their behavior; however, the computational cost associated with solving them restricts the simulations to a few hundred cells. Currently, the cells are modeled as two dimensional disks with possibly different radii. They have conducted simulations which exhibited cell migration, cell differentiation, gradient following, clustering, lateral inhibition, and neurite outgrowth.

3.2 Differential adhesion hypothesis

The *differential adhesion* model aims to explain general cell movement, with segregation and engulfment in particular. Along with differential adhesion, we also examine the various biological mechanisms which may explain differential adhesion. Some criticisms of this model are presented, and alternatives are mentioned.

Cell adhesions play a major role in the process of development. Adhesion is the force of attraction between heterogeneous or homogeneous cells, and cohesion is the force of attraction between homogeneous cells. Adhesion includes cohesion.

Early experiments by H.V. Wilson in 1907 revealed that dissociated single sponge[2] cells reaggregate into a functional Sponge. Holtfretter, in 1944, conducted experiments on reaggregation of cells from different embryonic tissues demonstrating that cells could find their proper locations, and that reaggregation was not due to redifferentiation[3]. Cells were not modifying their behavior depending on the location to which they moved; rather, they were moving to a location where they could exhibit their characteristic behavior. In other words, cells segregate according to their tissue type.

Steinberg observed that cell aggregates from embryonic tissue have a tendency to round up *in vitro*. In fact, there are quantitative differences in this rounding up; limb bud tissue rounds up more than heart tissue, which in turn rounds up more than liver tissue [Ste78]. If limb bud cells and heart cells are intermixed, the limb bud cells migrate centrally. If heart cells and liver cells are intermixed, the heart cells migrate centrally (i.e. the liver cells envelop the heart cells). A transitivity in this central migration is also observed; thus, if limb bud cells and liver cells are intermixed, then the limb bud cells migrate centrally.

Cell segregation, rounding up, central migration, and transitivity of migration are properties observed in non-living liquids too. Liquids tend to round up in the absence of external forces. Some liquids, when intermixed, segregate. This segregation may involve one of the liquids being suspended as a droplet inside the other (central migration). In liquids this behavior is known to be due to surface tension forces, which arise due to differences in adhesion between molecules of different liquids. This led Steinberg to postulate the *differential*

[2] A sea animal of the phylum *Porifera*.

[3] Differentiation is the process (often irreversible) by which cells modify themselves for a special function or purpose.

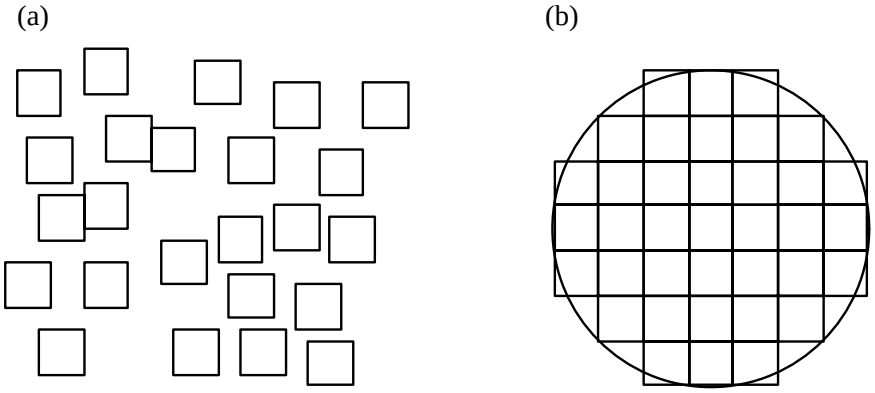


Figure 3.5: Homogeneous cells with (a) zero adhesive energy (no clumping) (b) positive adhesive energy (rounding up).

adhesion hypothesis which states that cells adhere differently to one another. Accordingly, the arrangement of embryonic tissues *in vitro* can be explained by the forces generated by the surface tension due to differential adhesion [Ste70,Ste75].

Surface tension acts to minimize the surface area of liquids/cell aggregates. Surface free energy is the effect of this force (tension) per unit area, and is equal to the reversible work required to expand the surface by unit area (also known as work of adhesion per unit area). Adhesive cells have a greater work of adhesion per unit area.

Cell surfaces have a fixed amount of surface available for binding to other cells. One can associate a free energy with this cell surface, which decreases as adhesions form. The cells move around till this free energy (V) of the aggregate is minimized, or equivalently, the adhesive energy (E) is maximized. Thus, $E+V$ can be regarded to be a constant. This holds for both homogeneous and heterogeneous cell aggregates.

To formalize the notion of adhesive energy for a cell aggregate, let us consider cell types: $1, 2, \ldots n$. Let E_{ij} be the adhesive energy per unit area between cell types i and j. E_{ii} is the *cohesive* energy per unit area for cell type i. E is symmetric, i.e. $E_{ij} = E_{ji}$. If A_{ij} is the contact area between cell types i and j, then the total adhesive energy of an aggregate of the cells is $\sum_{i<=j} E_{ij}A_{ij}$. The maximization of this adhesive energy leads to the cell aggregate being thermodynamic equilibrium.

There are two equilibrium cell configurations for the single tissue case ($n = 1$). For the sake of simplicity we assume that all the cells have the same size and shape. Examples of these two configurations are given in figure 3.5.

(a) $E_{11} = 0$. The cells do not clump together; the energy of the aggregate is 0, irrespective of the cell configuration.

(b) $E_{11} > 0$. The adhesive energy is maximized when cell contacts (A_{11}) are maximized

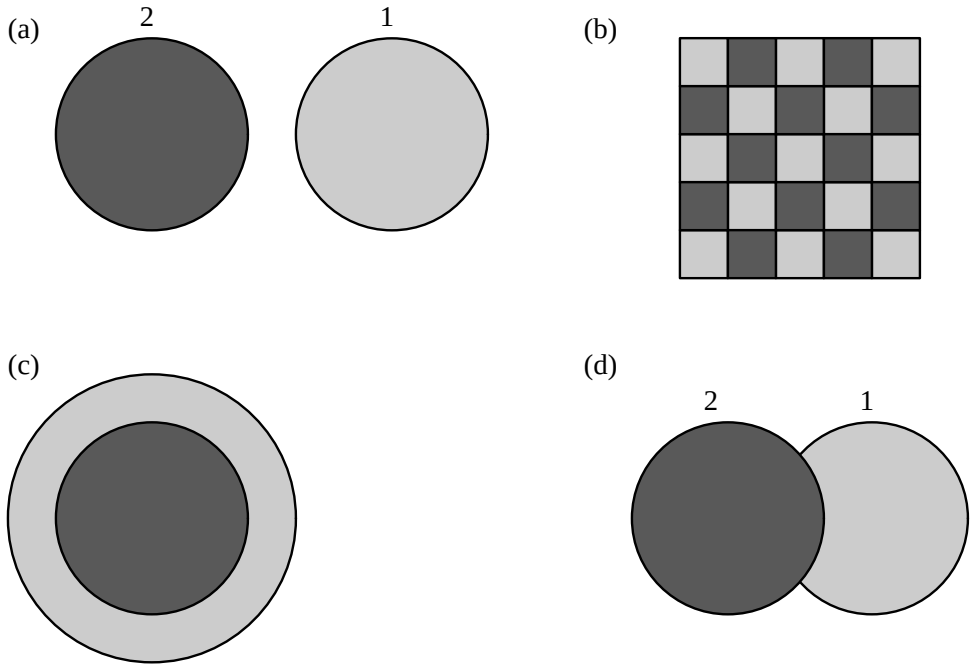


Figure 3.6: Two types of cells with different adhesive energies (a) $E_{12} = 0$ round up separately (b) $E_{12} > (E_{11} + E_{22})/2$ checkerboard pattern (c) $E_{11} \leq E_{12} < (E_{11} + E_{22})/2$ *onion* pattern (d) $0 < E_{12} < E_{11} < (E_{11} + E_{22})/2$.

or when free cell surface is minimized. The cells thus form clumps which are shaped, so that they have minimum exposed surface area for the given volume (sphere in three dimensions and disk in two dimensions).

For the two tissue case ($n = 2$), without loss of generality, we assume that $E_{11} < E_{22}$. The interesting configurations occur only when $0 < E_{11}$. Four interesting equilibrium cell configurations are shown in figure 3.6.

(a) $E_{12} = 0$. Contacts between different cell types do not increase adhesive energy. This results in cells of each type behaving as if the cells of the other type were absent, and each cell type forms spherical aggregates with cells of its own kind. Thus, segregation into separate spheres/disks occurs.

(b) $E_{12} > (E_{11} + E_{22})/2$. In this case, heterogeneous contacts (between different cell types) are preferred to homogeneous contacts. The adhesive energy is maximum for a checkerboard pattern, with the two cell types alternating.

(c) $E_{11} \leq E_{12} < (E_{11} + E_{22})/2$. Since adhesions ($E_{12}$) are quite strong, the entire cell aggregate will shape into a sphere to minimize free surface area. On the surface type 1 cells are preferable, since $E_{11} < E_{22}$. Segregation occurs because the average cohesion

is less than the average adhesion ($E_{12} < (E_{11} + E_{22})/2$). This results in a spherical ball of type 2 cells covered by a shell of type 1 cells. This pattern is often observed when two cell types are mixed in experiments and is referred to as the *onion* pattern.

(d) $0 < E_{12} < E_{11} < (E_{11} + E_{22})/2$. The adhesive interaction are the weakest but non-zero. In one limiting case, $E_{12} = E_{11}$, we have a type 2 sphere enclosed by a type 1 shell (case (c)). In the other limiting case, $E_{12} = 0$, we have the cells separating into disjoint spheres (case (a)). Between the two limiting cases, type 1 cells do not completely surround type 2 cell but recede slightly, exposing some type 2 cell surface. The closer E_{12} is to zero, the more they recede.

There are numerous configurations involving the intermixing of more than two tissue types. Goel et al. [GCG+75] provide a detailed description of all possible configurations of two and three types of cells.

3.2.1 Mechanisms responsible for type specific adhesions

What type of microscopic cell properties are consistent with observed macroscopic cell adhesion behavior? The adhesive transitivity observed suggests that quantitative differences in the microscopic properties may account for differential cell adhesion.

Steinberg has proposed a theory in which all cells (even of different types) have the same kind of immobile adhesive sites [Ste75]. However, the number of adhesive sites depends on the cell type. The adhesive energy is proportional to the number of sites per unit area in apposition. Let the probability of an adhesive sites at a given location be f_l for cell type $l = 1, 2, \ldots n$. Without loss of generality, we assume that if $i < j$ then $f_i \leq f_j$; we are simply arranging the cell types in increasing order of number of contact sites. The probability of apposition of sites in the cell pair i, j is given by $f_i f_j$ (since the location of sites are independent events). Consider $i < j$,

$$f_i \leq f_j$$

Multiplying both sides by f_i:

$$f_i^2 \leq f_i f_j \tag{3.1}$$

Also

$$\begin{aligned} (f_i - f_j)^2 &\geq 0 \\ \Rightarrow 2 f_i f_j &\leq f_i^2 + f_j^2 \\ \Rightarrow f_i f_j &\leq \frac{f_i^2 + f_j^2}{2} \end{aligned} \tag{3.2}$$

Combining equations 3.1 and 3.2, we obtain

$$f_i^2 \leq f_i f_j \leq \frac{f_i^2 + f_j^2}{2}$$

Introducing the proportionality constant k, the adhesive energy being $E_{ij} = k f_i f_j$, the above equation can be rewritten as

$$E_{ii} \leq E_{ij} \leq \frac{E_{ii} + E_{jj}}{2} \tag{3.3}$$

which is recognized (from page 18 case (c)) as the adhesive relationship that leads to the *onion* pattern with tissue $i = 1$ enveloping tissue $j = 2$. This microscopic explanation explains the onion pattern which is frequently seen in biological experiments concerning mixing and segregation of tissues. In this system, segregation and engulfment always occur (unless $f_i = f_j$), and the engulfment is transitive. Consider a third tissue k, and if $j < k \Rightarrow f_j \leq f_k$, then tissue j would envelop tissue k. But since $i < j$ and $j < k \Rightarrow i < k \Rightarrow f_i \leq f_k$, tissue i would envelop tissue k, yielding the transitivity property in central migration.

A different microscopic model has been suggested by Childress and Percus [CP78]. This has different assumptions but makes similar predictions. Steinberg considered the probability of apposition of two adhesive sites from distinct cell pairs to be proportional to $f_i f_j$. Childress and Percus consider mobile adhesive sites; thus, the number of contact sites in their model is proportional to $\min(f_i, f_j)$. We assume $f_i \leq f_j$. The sites being mobile, each of the type i sites would move until it found a type j site to itself. The expression for the adhesive energy is

$$E_{ij} = (k_i + k_j) \min(f_i, f_j) \tag{3.4}$$

k_i can be viewed as the potential adhesion energy at a single site of cell type i. Therefore, an i,j contact site gives rise to $k_i + k_j$ adhesive energy. This model predicts the transitive property of central migration, like the immobile receptor site model of Steinberg. It can be also shown that if tissue k engulfs tissue j and tissue j engulfs tissue i, then tissue k must engulf tissue i.

Both the Steinberg and Childress & Percus explanations for the microscopic behavior fail to account for partial engulfment, which requires $E_{ij} < E_{ii}$, and separation, which requires $E_{ij} = 0$. Yet each presents reasonable explanations for the most common cell configurations.

3.2.2 Critique

Harris has raised some objections to Steinberg's differential adhesion hypothesis [Har76]. There are a number of crucial distinctions between intermolecular attraction (in liquids)

and intercellular adhesion (in cell aggregates), namely[4]

- Liquid drops are closed systems thermodynamically whereas aggregates of living cells can generate an indeterminate amount of metabolic energy capable of altering cell positions and adhesions.
- Intercellular adhesions are more than just close range interactions since cells can be held together by forces (for example, enzyme-substrate, antibody-antigen etc.) in addition to those which originally pulled them together.
- The breakage of intercellular adhesions need not be simply the reverse, thermodynamically, of the formation of those adhesions.
- Intercellular adhesions being concentrated at relatively small foci such as desmosomes, a maximization of the intercellular adhesion does not necessarily require a maximization of intercellular contact area, or vice versa.

Harris also proposes various alternative hypotheses, most of which are minor variants of Steinberg's hypothesis. In addition,

> "a differential surface contraction hypothesis is proposed, according to which cell sorting and related phenomena are the results of cell surface contraction induced to occur over those portions of the cell surface, which are exposed to the surrounding culture medium. There is reason to believe that various invagination type movements of embryonic epithelium are caused by cell surface contractions, which suggests the feasibility of the differential surface contraction hypothesis for cell sorting etc."[Har76]

3.2.3 Models and simulations

In spite of the objections of Harris, the differential adhesion hypothesis remains popular, and various researchers have built models and conducted simulation to further explore the hypothesis.

Goel et al. have conducted numerous simulations modeling cells as rigid objects with preassigned adhesive properties [LG71,GCG+75,GR78,RG78]. These cells form a regular square tessellation of the plane. The medium is modeled as just another cell type. Neighboring pairs of cells are allowed to interchange positions, if this interchange increases the total adhesive energy. Their model assumes differential affinities between cells of different types and local motility of cells.

[4] Quoted from [Har76].

Goel et al. considered two-dimensional rectangular aggregates of cells of two to four different types [GCG+75]. They derived all the absolutely stable patterns (*maximum* adhesive energy) in such aggregates, by considering the different adhesive strengths between cell types. According to Goel et al., "none of those configurations was found to be significantly histologically interesting." They ran computer simulations of cell movement, from various randomly chosen starting cell distributions, of two cell types. They used different plausible motility rules in a bid to reach the maximum adhesive energy configuration. Most of these trials failed to reach global maxima and were trapped in locally stable configurations. However, some locally stable configuration (*maximal* adhesive energy) were found to be histologically significant (rings and sheets of cells).

Antonelli et al. [ARW75, Appendix A] observed that Goel et al. [GCG+75] ignored the effects of boundary conditions on the simulations they conducted. The calculations of Goel et al. assume that the cell array size is large enough to be able to ignore boundary effects; yet their experiments are conducted on small cell arrays, where the boundary effect is substantial.

Goel and Leith considered anisotropic cells [GL75]. Different sides of the cells have different adhesive properties. Some of the maximal adhesive energy patterns obtained were histologically interesting; they were able to produce open and closed rings which can be looked upon as the two dimensional analog of vesicles, closed shell and epithelial patterns. They still had problems with the segregation not proceeding to completion and the simulations ending at local maxima.

Leith and Goel investigated the effects of different motility rules on segregation experiments with two cell types [LG75]. They tested the effects of 21 different motility rules, which varied in the selection, direction, distance and type of the motile cell, and the minimum change in energy required for movement. Most of these rules had little effect on the final results; the only factor that seemed significant was the range of the interaction between the cells.

Antonelli et al. examined simulations involving hexagonal cells. Their exchange principle only allowed moves between nearest neighbors that increased total adhesive energy [ARW75]. Three or more cell types (one being medium) were used to model tissue engulfment. They considered a hexagonal tessellation of the plane. Each cell occupies a cavity in the hexagonal lattice. Two neighbors are allowed to exchange positions if that exchange does not lead to a decrease in total surface adhesion. Their simulations did not yield the central clumping reported in other experiments. Their model achieved relatively rapid formation of small internally segregating clumps.

Goel and Rogers investigated non-local exchange procedures for cells [GR78]. They overcame the difficulty of tissue segregation ending at a local energy maxima and were able to successfully simulate, among other things, tissue engulfment. Movement was considered

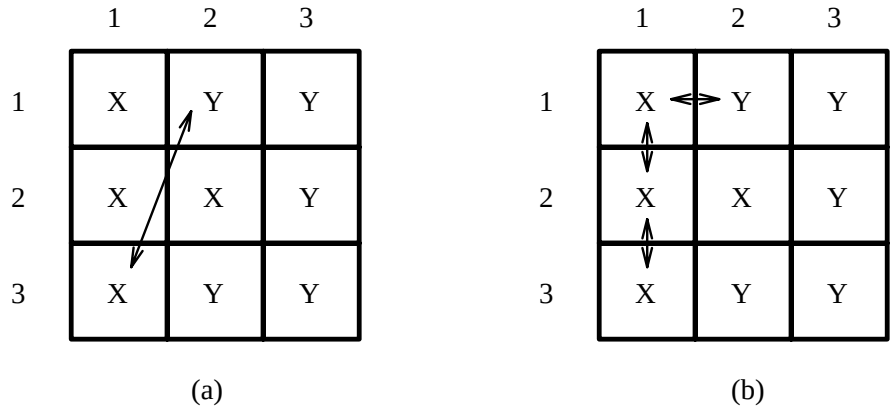


Figure 3.7: An example of Goel and Rogers's distant exchange being viewed as a series of local exchanges. $E_{XX} = 1, E_{XY} = E_{YY} = 0$. Consider 4 neighbors for each cell (left,right,top, and bottom) (a) The maximum positive $\Delta E = 2$ is for cell (1,2). The smallest negative $\Delta E = -1$ is for cell (3,1). (b) The exchange of (1,2) and (3,1) visualized as a series of local exchanges.

only for cells on interfaces between two different cell types, since these moves are the only ones that change the global energy. For every cell position on an interface, the change in energy ΔE resulting from switching the cell type was computed. The lists of cells on the interface were split into positive ΔE (sorted in non-increasing order) and negative ΔE (sorted in non-decreasing order). If the sum of the ΔE's in position $m = (1, 2, \ldots)$ in both the lists was greater than zero, then the corresponding cells were switched. The switch at position 1 corresponds to the swap that results in the greatest increase in adhesive energy of all heterogeneous pairs at the tissue interface. This procedure was repeated for every interfacial boundary in turn[5].

The swaps dictated by the above rule may not be local. They may correspond to the shift of a whole cell aggregate by an array position. Goel et al. believe that this distant exchange of cell types along the interface does not violate the basic motility rule that a cell can move only one cell diameter per simulated time step. Rather, a single distant exchange may be visualized as a series of simultaneous neighborly exchanges within the interfacial areas (illustrated in figure 3.7). In spite of this observation, the distant exchange is still a cause for concern, since non-local information is used to make the exchanges.

Goel and Rogers also considered differing neighborhoods for computing the adhesive energy E_i of the cell at location i [GR78]. In general, the energy for a cell, considering interactions up to D layers, would be

$$E_i = \sum_{j=1}^{D} \sum_{k=1}^{8j} K_j E_{i,jk}$$

[5]In CPL, we use a generalized version of this procedure to implement the round up instruction.

The j^{th} layer has $8j$ cells, K_j is a weight given to interactions with cells in the j^{th} layer, and $E_{i,jk}$ is the value of the adhesion parameter between the type of the i^{th} cell and the type of the k^{th} cell in the j^{th} layer. Long range interaction (effect of cells in layers $j = (2, 3, \ldots D)$) may be justified by the extension of processes (such as filopodia, pseudopodia) over multiple cell diameters.

They successfully simulated engulfment of two and three tissues, of equal size, starting from an initial configuration of rectangular clumps of each cell type in contact. The overall behavior of this model agrees quite well with the experimental observations.

Rogers and Goel extended this work by simulating segregation of mixtures of two different cell types into separate phases [RG78]. Whether the two cell types sorted out or not and the final configuration reached were found to be dependent on the initial concentration of the cell types and the range of the interaction.

Matela and Fletrick have also conducted simulation cellular segregation using a graph to model the cell structure. This is discussed at length in section 3.3.4.

In summary, the differential adhesion hypothesis has not gained universal acceptance, but it does seem to be the most popular one. The simulations have produced mixed results. Our simulations in chapter 7 shed some more light on the differential adhesion model; in particular, we were able to obtain improved segregation and engulfment results by introducing some random motion.

3.3 Representation of cells

The array is an obvious data structure; most of the early cell growth and division models revolve around it. In the previous section, we have already seen the use of the array in cell segregation simulations by Goel et al. The one dimensional filaments (which simulated algae growth) by Ransom, Conway's game of life, and Gordon's stochastic spirals preceded Goel in his use of arrays to represent cells [Ran81]. In fact, all these models are special cases of the cellular automata model that we have examined earlier. However, these do present models of varying biological activity.

More recently, researchers have employed voronoi diagrams, graphs, and polygons to represent cells.

3.3.1 Simple array models

Ransom has employed one dimensional arrays to simulate algae filament growth [Ran81]. If "1" represents a cell and "0" an empty location, then filaments grows by replacing each successive 0 by a 1. Thus, successive stages of the filament are:

1000000000
1100000000
1110000000
⋮
1111111110

In most two dimensional models each cell has four neighbors, and the structure is a square tessellation of the plane. The possible implementations of cell division are worth investigating.

Only cells on the boundary, with a free space in the neighborhood, are allowed to divide. One can study the various patterns formed, as a result of choosing different rules which determine the order of division of the boundary cells, and the neighboring location that the cell divides into, in case of multiple vacancies.

If all cells are allowed to divide, the internal cells divide by placing the daughter cell into the nearest empty location they can find. However, this leads to the daughter cell not being adjacent to the parent. Alternatively, the daughter cell are placed next to the mother cell, and space is created for the new cell by pushing the other cells out. Various strategies may be used for pushing the cells. We can either push all the cells on the straight line path from the dividing cell to the nearest boundary out. Alternatively, we can replace a neighbor of the parent cell by the daughter cell, and then try to place this cell, in one of its neighboring locations. This strategy bubbles cells out, not necessarily in a straight line.

The final suggestion was to use a variable size cell, occupying either 2 or 3 array locations. The cell grows from 2 to 3 locations, and then grows into 4 locations dividing immediately into 2 cells occupying 2 locations each. These variable sized cells do not result in visible improvements in the simulation.

3.3.2 The game of life

The *game of life* was devised by J.H. Conway and made popular by Martin Gardner. This model is worth mentioning more for its popularity than for its biological significance. It consists of a two dimensional array, with each array position designated a cell. Each cell has 8 neighbors (4 orthogonal, 4 diagonal). Each cell can have one of two states: full or empty. The rules of the game are:

- Each full cell with 0 or 1 full neighbor dies (from isolation).
- Every full cell with 2 or 3 full neighbors survives for the next generation.
- Each full cell with 4 or more full neighbors dies (from overpopulation).

- Each empty cell adjacent to exactly 3 full neighbors is born.

Birth is the process that changes an empty cell to full, while death is the reverse process. These simple rules lead to various periodic patterns that move both in space and time.

3.3.3 Gordan's spirals

Richard Gordon, in 1966, developed a model to simulate the growth of a spiral by stochastic means [Gor66]. Using a two dimensional array, Gordon defined two types of cells, I and S. Growth of these cells occurs into the orthogonal neighboring empty array positions. Growth is essentially division into an empty neighboring cell. Only cells on the boundary are permitted to grow. One of the S cells is designated the leader and is the only S cell which may divide. (It corresponds to the tip of the spiral.) The cell it produces becomes the new leader. Growth of S cells is directional; the leader cell grows into the site to its left (relative to the vector from its parent, directed to itself). In the case where the site on its left is occupied, it attempts to occupy either the forward or the right site, in that order of preference.

If S cells were the only ones present, it would result in a tight square spiral. The I cells are used to open up the spiral. The I cells grow randomly, akin to tumor cells.

If the I cells divide too often they may totally surround the S cells, killing the spiral. On the other hand, if they grow too slowly the spiral will get very tight, since the S cells would completely surround the I cells. If each cell grows at a specific, intrinsic rate, independent of the rest, it is termed *local* growth. If the leading S cell could count the number of active I cells (for example, by being sensitive to a hormone diffusing from them), and the S cell grows at a rate proportional to this count, it results in an Archimedean spiral.

The growth of S cells is reminiscent of apical meristem in plants, which only grows at the tip of a shoot, inhibiting growth below. Also, vines tend to spiral in one direction.

3.3.4 Topological model

The advantages of the discrete cell models discussed in the previous sections, in terms of speed and ease of manipulation, are obvious. However, these models ignore the fact that cells have shape, their geometry need not remain fixed, and that cells move, while slowly changing their contacts with their neighbors. Polygonal cell models avoid these drawbacks; however, most operations on them are not so easily defined. For example, the issue of adjusting the surrounding polygons, so as to give a growing polygon some additional space, is not resolved.

Matela and Fletterick have used graphs to represent cell maps [RM84,MR85]. They represent each cell by a vertex. If two cells share a common boundary, there is an edge

between the two respective vertices. The graph so obtained is planar and is the dual of the cell map. In most of these simulations they restrict the graph to be triangulated, which implies that in the cell map no corner has a degree more than three (i.e. no four cells meet at a point).

The advantages of this model are that it permits cells with different shapes (any n-gon), and graphs are computationally well studied structure. However, representing the cell map by its dual graph has its own share of problems. The graph does not define a unique cell map, nor does it specify the size of each cell or the contact length with other cells.

In the graph model, it is possible to demonstrate segregation of tissues using only local information [MF80]. Consider a graph containing two different types of vertices corresponding to different cell types. Initially the graph is quite random, in the sense that the vertices are connected by edges picked at random to form a triangulated graph. An edge represents the boundary between the cells. The cells are sorted by adopting an exchange rule. Each quadrilateral in this graph is inspected in turn. Every quadrilateral has exactly one diagonal (since the graph is triangulated). Replacing this diagonal by the other one leads to a new triangulated graph. In terms of the dual (the cell map), we have replaced the contact edge between two cells by a contact edge between two other cells. If this replacement (exchange) is guided to ensure only profitable exchanges, then enough iterations of this mechanism would yield the cell map with the highest adhesive energy.[6]

Their experiments have shown that a large degree of isotypic cell association may be produced by using a topological model. However, it is obvious that this exchange technique would lead to the simulation being stuck in local minima, and not lead to complete segregation.

Ransom and Matela have also addressed the problem of cell division in terms of the graph model [RM84].

3.3.5 Voronoi diagrams

The *voronoi* diagram model is computationally feasible. This cellular model was first suggested by Hisao Honda in 1978 [Hon78] and was also employed by Sulsky [Sul82]. A voronoi diagram enables one to represent a cell by a single point called the nucleus. The cells are modeled as convex polygons in the voronoi diagram model.

The nuclei have coordinates $s_1, s_2, \ldots, s_N$ in a region $\Omega \subset \mathcal{R}^2$. Define the half spaces

$$H_{ij} = \{s \in \Omega : |s - s_i| \leq |s - s_j|\}$$

[6] Any triangulated graph can be transformed into any other triangulated graph on the same number of vertices with a finite number of edge exchanges. This guarantees that the highest adhesive energy graph is, in theory, reachable from every starting configuration.

H_{ij} is the set of all points closer to s_i than to s_j. Define the polygon or cell corresponding to the nuclei s_i as $P_i = \bigcap_{j=1}^{N} H_{ij}$. Therefore, P_i is the set of all points closer to s_i than to any other nuclei s_j. P_i is convex and polygonal, since it is an intersection of a finite number of half spaces. The edges of P_i are made up of points s such that $|s - s_i| = |s - s_j|$ for some neighbor j. These edges represent the cell boundaries.

The advantage voronoi polygons have over arbitrary polygons is that each voronoi polygon can be defined by a single point, its nucleus. Thus, N cells can be specified by exactly N points. In the case of arbitrary polygons one has to specify each of the corner points, and there may be infinite corner points in an arbitrary planar graph with N faces.

Not all polygonal subdivisions of a space correspond to voronoi diagrams. In fact, voronoi diagrams form a small specific class, and even their own projections at an angle, or a scaling along one axis, are not voronoi. Honda studied in detail what kind of patterns occurring in nature are voronoi [Hon78]. He devised a metric that measures the deviation of a polygonal subdivision from the closest voronoi subdivision. Based on this metric Honda categorized patterns occurring in nature. Pattern of alveoli (lung cells), the coenabial algae —*Pediastrum Boryanum*, epithelial cells of chick embryo, and endothelial cells of rabbit aorta were all categorized as roughly voronoi.

Sulsky carried out simulations using voronoi regions to represent polygons [Sul82]. Often, columnar cells are observed in embryos, whose height is inversely proportional to their cross-sectional area. These cells can be regarded as right prisms with polygonal cross-sections determined by voronoi polygons. Faces of these polyhedra represent cell membranes.

Forces, responsible for cell movement, act along cell membranes and are equivalent to surface tensions at these interfaces. The model generates motion in the cell configuration by maximizing the rate of free energy decrease, subject to a dissipation constant. The form of this constraint is suggested by continuum mechanics, resulting in aggregate motion that resembles a viscous fluid driven by surface tension forces.

The model was used to simulate cell reaggregation and sorting experiments. In addition, the embryological process of *neuralation*, in which a circular monolayer of cells changes to a keyhole shape, was modeled. These simulations were accomplished by choosing a regular grid of cells and the initial distribution of interfacial surface tensions. Cell movement was directed by the variational principle.

3.3.6 Polygonal cell division models

D'Arcy Thompson considered the problem of cell division — especially when patterns are produced as a direct consequence of the cellular divisions [Tho17]. Thompson suggested that the following rules should hold during cell division:

- The dividing partition should cut across the longest axis of the cell.
- Each partition must run at right angles to its immediate predecessor.

He hoped that a partition chosen with the above rules would be the smallest that generates equal volume daughter cells.

Cowan and Morris have used similar rules in their simulations [CM88]. They considered polygonal cells and examined various rules for choosing the division line. They postulated an imaginary organism, *Tessellata elegans*, which grows in a monolayer on a planar medium. It starts with a single convex polygonal cell. It divides by connecting two sides of its membrane by a line segment, giving rise to two daughter cells, both again convex polygons. Uniform expansion of the organism accommodates cell growth. Cell division is synchronous, but division lines are chosen independently. A division line does not meet a vertex, and neither does it distort the membrane that it meets (i.e. the angle in the neighboring cell remains at 180°). Their experimental results are mainly concerned with the statistical distribution of cell types. The type of the cell is the number of edges in the cell. They investigate the effects of choosing dividing lines through different sides of the cell.

To apply these concepts to generate epithelial patterns they suggest:

- reducing the junction angles from 180° by repositioning the cell membranes,
- choosing the dividing line of a cell to have one end on the line that partitioned its mother, and
- having rough equality in sibling areas.

Section 8.2.2 contains a simulation image generated by CPL which is similar to that obtained by Cowan and Morris.

In this section, we have examined a variety of physical representations for cells, which have been employed at various points of time. CPL has borrowed ideas from all these models, and to a greater extent from the cellular models discussed in the earlier sections of this chapter.

Chapter 4

Details of the Cell Programming Language

This chapter discusses the syntax and the semantics for the instructions in CPL. The implementation issues are discussed in the last section of this chapter.

CPL reserved words appear in this thesis in the `typewriter type style`. Names between "<" and ">" represent templates. <*expression*> refers to any expression, <*integer-expression*> refers to any expression evaluating to an integer, <*direction-expression*> refers to any expression evaluating to a direction, <*instruction*> refers to any single instruction, or a block of instructions enclosed in curly braces. CPL uses curly braces in the style of the programming language C++ to mark the beginning and end of blocks, and semicolons to separate instructions [Str91].

In CPL, all non-reserved words (with the first character being a letter and the following characters alphanumeric or underscores) are *valid identifiers* (used for biochemical, tissue, state, and variable names). Distinctions are made between upper and lower case letters, and all the reserved words are in lower case.

4.1 Expressions

CPL expressions follow the rules of C or C++ [Str91]. In particular, expressions evaluating to zero are regarded as false and those not evaluating to zero as true.

CPL expressions use a subset of the language C's operators. CPL operators are listed in figure 4.1.

The possible operands in these expressions may be:

- constants: integer, real, and direction (of the form point(integer,integer));

[0]Much of the contents of this chapter have appeared earlier in a technical report [Aga93].

Operator	Use/Meaning
!, &&, ||	logical not, and, or
^	exponentiation
*, /, %	multiplication, division, modulo
+, −	addition, subtraction
==, != , >, <, >=, <=	relational operators
=	assignment operator

Figure 4.1: CPL operators

- variable names;
- biochemical names;
- cell attributes: perimeter, area, cell number, location;
- neighbor attributes: neighbor.direction, neighbor.contact_length, neighbor.perimeter, neighbor.area, neighbor.cell_number, neighbor.state and neighbor.tissue_type[1];
- simulation attributes: time, time_interval, steps (steps = time/time_interval); and
- functions: sqrt:real→real, int:real→int, and random:(integer×integer) →integer[2].

The non-trivial operands are discussed in section 4.6.

4.2 Instructions

Examples of CPL instruction sequences are highlighted by placing them in rectangular boxes in this thesis.

The programs written for cells are called *tissue definitions* because the same program is used by all the cells of the same kind or tissue. In addition, there are *cell definitions*, which contain information about the starting location (and chemical concentrations) of the initial cells.

The repertoire of instructions that the cell may choose to execute is listed in this section, along with descriptions of the syntax and semantics of each of them.

[1] neighbor.state and neighbor.tissue_type may only be tested for equality or inequality to name strings to check the state or tissue type of the neighbor.

[2] random returns a random integer uniformly distributed between the two given integers.

- **assignment:** The assignment instruction is used to assign new values to variables. CPL has the simple assignment statement,

  $\boxed{clock = 1}$

  The positive and negative accumulator assignments (as in the language "C") are also permitted.

  $\boxed{clock \mathrel{+}= 1}$

  $\boxed{clock \mathrel{-}= 1}$

  In addition, derivatives of biochemicals may be specified, and that assignment takes the form:

  $\boxed{\texttt{deriv } BcOne = k * BcTwo * BcThree}$

  `deriv` is a reserved word and indicates that the expression on the right hand side (RHS) is *added* onto the previous BcOne value (and not assigned). The RHS is also implicitly multiplied by the size of the time step[3] in the simulation. Moreover, the addition to biochemical values is only effected at the end of the time period. The above equation assumes the following reaction:

  $$BcTwo + BcThree \rightarrow BcOne$$

  with the rate of production of BcOne (d $BcOne/\mathrm{d}t$) being given by

  $$\frac{deriv\ BcOne}{\Delta t} = k * BcTwo * BcThree$$

  $$deriv\ BcOne = (k * BcTwo * BcThree) * \Delta t$$

  The biochemical assignment statement in CPL resembles the above equation, though the multiplication of the right hand side by Δt is implicit. Δt should be small for the equation to be valid.

[3] The time step is explained in section 4.3

In addition, the biochemical concentrations can also be set to a specified value by using the simple assignment (without using the `deriv` reserve word). In that case, there is no implicit multiplication of the RHS by the time interval.

BcOne = 1.0

This sets the concentration of BcOne to 1.0 at the end of the next time step. This statement is primarily used to set initial concentrations.

Thus the assignment statement has the following possible forms:

`deriv` <biochemical> = <expression>
<biochemical> = <expression>
<variable> = <expression>
<variable> += <expression>
<variable> −= <expression>

- **move:** The move instruction causes the cell to move by exchanging the location of the cell with that of a neighboring cell in the specified direction. The move is well defined if the two cells exchanged are of equal size; however, the move is not as well defined when cells of unequal sizes are involved.[4]

  `move` *direction1*

  The general form of the move instruction is:

  `move` <direction-expression>

  A `move` changes the neighborhood and location of both the cells involved. This may require recomputation of values for some of the cell variables, particularly the biochemical gradients.

  The direction specified is relative to the location of the cell.

- **goto:** A `goto` instruction specifies a state switch. A cell executes all the instructions in its present state at each time unit; the `goto` provides a mechanism for switching this set of instructions. Typically, gotos would be used to cycle between a set of states (such as *waitForSignal, readyToSignal*, and *signal*).[5] These states consist of a set of instructions specified by the user. There are no predefined states.

[4] In the unequal area case, the `move` conserves the areas of all the cells involved; however, their shape need not be conserved.

[5] These states are employed in the cellular slime mold example in chapter 5.

The general form of the goto instruction is:

`goto` <state-name>

`goto` *waitForSignal*

The execution of the `goto` ceases execution of the rest of the program for that time step. In other words, the `goto` instruction, if executed, is the last instruction executed for a cell at any time step. At the next time step, instructions belonging to the new state are executed.

- **differentiate_to:** This instruction is a form of the `goto` instruction. The `differentiate_to` instruction can be used to specify the type of tissue the cell should differentiate into. This is employed to specify a major change, often irreversible, in the cell's life history. From the programmer's perspective, it is possible to eliminate one of the two instructions: `goto` or `differentiate_to`; however, they serve different biological purposes (cycling between a set of states and irreversible change).

  The general form of the `differentiate_to` instruction is:

  `differentiate_to` <tissue-name>

  `differentiate_to` *epithelial*

  The `differentiate_to` instruction changes the CPL program that the cell executes.

  The execution of the `differentiate_to` ceases execution of the rest of the program for that time step. At the next time step, the program for the new tissue is executed.

- **for_each_neighbor_do:** This permits the execution of an instruction (or a block of instructions) using the parameters of each of the neighbors in sequence. This instruction takes another instruction as its argument. Some extra read-only variables are available for in the scope of the `for_each_neighbor_do`, specifically, the area of the neighbor; the direction to the neighbor; contact length with the neighbor; the tissue type and state of the neighbor; the biochemical concentrations inside the neighbor; and the variable values inside the neighbor.

  The `for_each_neighbor_do` executes the block of instruction with each neighbor in turn. It randomly picks up the first neighbor and then cycles through the remaining neighbors by going around the cell boundary. The `exit` instruction may be used to exit the loop prematurely. Nested `for_each_neighbor_do`'s are not allowed.

  The general form of the `for_each_neighbor_do` instruction is:

`for_each_neighbor_do` <instruction>

```
for_each_neighbor_do
  deriv BcOne = (neighbor.BcOne − BcOne)/Drate;
```

The above instruction sequence illustrates a possible representation of diffusion. An amount proportional to the difference in the concentration of BcOne between the cell and the neighbor is added to the concentration of BcOne.[6] In a similar fashion, diffusion for the other biochemicals in the cell could be represented. The variable *Drate* determines the rate of diffusion. A larger *Drate* would indicate slower diffusion.

```
dirBcOne = point(0,0);
for_each_neighbor_do {
  deriv BcOne = (neighbor.BcOne - BcOne)/Drate;
  dirBcOne += neighbor.direction * (neighbor.BcOne - BcOne)/Drate;
}
```

The above instruction sequence demonstrates the accumulation of the direction of the biochemical diffusion, which is the sum of the direction to its neighbors weighted by the amount of the biochemical diffusing from the respective neighbor.

- **if-then-else:** The if-then-else instruction provides conditional execution of instructions. The conditions can depend on any of the cell attributes, including the neighbor attributes (only when `if-then-else` is used inside a `for_each_neighbor_do` or `with_neighbor_in_direction`).

  The general forms of this instruction are:

  `if` <expression> <instruction>
  `if` <expression> <instruction> `else` <instruction>

  If the expression evaluates to a non-zero value, the condition is taken to be true.

```
if (area > 100) divide horizontal
```

[6] The user should ensure that the cells do not cheat in diffusing biochemicals. If a cell receives some amount of biochemical due to diffusion, some other cell must lose an equal amount of it; therefore, the language user must write code ensuring the conservation of biochemicals during diffusion. The language facilitates this since all the cells access the same biochemical values at any time. It is impossible to change the biochemical value inside a cell until the end of a time step.

- **exit:** The `exit` may only be used inside a `for_each_neighbor_do`, and the remaining instructions inside the `for_each_neighbor_do` are not executed. The execution resumes at the instruction succeeding the `for_each_neighbor_do`. `exit` may be used to help determine a neighbor with a particular property. The neighbor is undefined on an exit from the loop; therefore, some variable (indicating the direction to the neighbor) must be set to determine the neighbor to use (if any) after exiting the loop. The general form of the exit instruction is:

  ```
  exit
  ```

- **with_neighbor_in_direction:** This instruction is used to employ the attributes of a single specified neighbor. It takes two parameters: a direction variable and an instruction (or block of instructions). It finds the neighbor in the given direction and executes the instruction(s) using the attributes of this neighbor. Frequently, `for_each_neighbor_do` and `with_neighbor_in_direction` are used in conjunction. The `for_each_neighbor_do` may cycle through all the cell's neighbors to determine the direction to a specific neighbor, such as a neighbor with a given tissue type. `with_neighbor_in_direction` then enables accessing the attributes of this specific neighbor.

  The general form of the `with_neighbor_in_direction` instruction is:

  `with_neighbor_in_direction` <direction-expression> <instruction>

  The net flow of *BcOne* can be determined, as in the `for_each_neighbor_do` example; `with_neighbor_in_direction` can then be used to move in that direction.

  ```
  with_neighbor_in_direction dirBcOne {
    if (neighbor.tissue_type == tissueA) move dirBcOne;
  }
  ```

  If the neighbor in the direction of the diffusion is of tissue type A, the cell swaps locations with this neighbor.

- **divide:** The divide instruction causes the area of a cell to be split up into equal halves. The instruction has an option specifying the direction of the cell division line. The choices are horizontal, vertical, perpendicular to last division, and random (any) dividing lines.[7] In addition to the area division, the values of the variables of the

[7] Each of these choices is represented by a reserved word.

parent (except biochemicals which are split equally) are copied to the children. One of the two daughter cells then finishes executing the state definition[8].

```
divide perpendicular
```

This causes the cell to divide `perpendicular` to its last axis of division.

- **grow:** The grow instruction causes the cell to grow in area by the given size in the specified direction. The size can be any expression evaluating to an integer. A negative size reduces the area of the cell (the cell shrinks). The direction can either be a specified by a variable (perhaps representing a biochemical gradient, in which case the cell grows preferentially along that direction), or `random_direction`, in which case the direction of cell growth is randomly chosen.

  The general form of the grow instruction is:

  grow <integer-expression> <direction-expression>

  ```
  grow 5 random_direction
  ```

  The above instruction causes the cell to grow in area by a unit in each of 5 randomly chosen directions[9].

- **roundup:** The execution of this instruction rounds up the cell by examining the cell boundary and exchanging boundary points so as to for a more cohesive unit. This instruction does affect the shape of the neighboring cells, since rounding up this cell requires modifying the boundaries of the neighbors. Repeated executions of roundup may lead to varying boundaries. The general form is:

  ```
  roundup
  ```

- **die:** As the name suggests, this results in cell death; no more instructions of this cell are executed. The general form is:

  ```
  die
  ```

This concludes our discussion on the instructions in CPL that are directly influenced by the cell biology.

[8] We still cannot determine if it is necessary to be able to distinguish between the daughter cells; therefore, we have adopted this arbitrary solution that one daughter cell finishes execution, since its implementation is the easiest.

[9] A cell is represented as a collection of discrete points. Growth by 5 units is equivalent to adding 5 lattice points to the cell.

4.3 Meta-instructions

The instructions discussed in this section enable us to control the simulation, aid program readability, and help visualize the results.

- **simulation_size:** Since the cells are modeled as a collection of discrete integral points, the user has to specify the maximum size (area) the simulation may occupy. This declaration must be the first statement in the program.

  `simulation_size` (<integer>, <integer>)

  `simulation_size` *(25, 30)*

  In the above example, the simulation is carried out for x coordinates ranging from 0 to 25, and y coordinates ranging from 0 to 30. The lower left hand corner is always (0,0), and the user specifies the upper right hand corner. Enough simulation space should be provided to ensure that the cells do not grow past any boundary.

- **time_interval:** The user can specify the time interval at which the program for each cell is executed. The right hand sides of the difference equations of biochemical catalysis and diffusion are implicitly multiplied by this quantity. Thus, if `time_interval` = 5, the cells execute their program at time 5,10,15..., which makes the simulation run 5 times faster. However, `time_interval` = 0.5 slows down the simulation by half. Only in the assignments to the derivatives of the biochemicals (`deriv`) is the time_interval implicit; the other instructions in the program have to use the `time_interval` explicitly. Thus, if some variable is monitoring the elapsed time in a cell, it should be incremented by `time_interval`. It is emphasized that the time interval should be a small number; otherwise, the validity of the discrete time simulation is questionable. The default value of the `time_interval` is 1. It may be overridden by a declaration following the `simulation_size` declaration.

  `time_interval` = <real>

- **echo** takes a string in double quotes as an argument and prints out this string on standard output when this instruction is executed. '\n' in the string is treated as a newline character.

  `echo` "<string>"

- **write** takes an expression as an argument and prints out the expression value followed by a space on standard output.

`write` <expression>
`write state`
`write neighbor.state`
`write tissue_type`
`write neighbor.tissue_type`

The last four forms of this instruction print out the numerical representation of the corresponding state or tissue type. These are useful for printing out cell motion information.

- **image** takes a biochemical or variable name as an argument and prints for each lattice point the particular biochemical/variable's value in the cell located at that lattice point. This can be used to visualize simulation results in image form. Alternatively, it can be used to print out a matrix of numerical representations of the tissue types or the states of various cells.

  `image` <biochemical-name>
  `image` <variable-name>
  `image state`
  `image tissue`

- **save:** This instruction saves the system state in a file; the simulation can be restarted from the last saved state.

  `save`

- **constants:** The language permits C style #define's to define constants.

- **comments:** In addition to instructions, the language also permits *comments*. It ignores everything on the line after encountering a //.

The meta-instructions aid us both in writing readable CPL programs and in producing results that may be easily visualized.

4.4 Variable declarations

CPL has simple variables, and it has biochemicals.

4.4.1 Biochemicals

Each biochemical appearing in a CPL program must be declared as such. The general form of the declaration is:

`biochemical` <id-list>

<id-list> is a list of identifiers separated by commas.

Biochemicals can also be of either the integer or floating point variety, and their storage type is declared akin to the storage type declaration for the other variables.

4.4.2 Other variables

CPL variables have two characteristics: storage type (integer, float, or direction) and scope (static or local).

- **Storage type**: The variables must be declared to be either integer, float, or direction. The general forms for the declarations are:

  `integer` <id-list>
  `float` <id-list>
  `direction` <id-list>

- **Scope:** Each cell has a different copy of most user defined variables, including biochemicals, and it retains this copy throughout its lifetime. Such variables are termed `local` variables. This is the default condition of all variables. In addition, `static` variables are made available, whose values are shared by all the cells. Thus, a `static` variable has the same value no matter from which cell it is accessed, as opposed to a local variable which has different values in different cells. `static` variables are useful for collecting statistics about the cell aggregate, such as the count of cells of a specified tissue type, or the total amount of a biochemical present in the entire tissue. In fact, it is difficult to justify using `static` variables for any purpose other than data collection, since in the biological context `static` variables permit non-local communication. The general form of the static declaration is:

  `static` <id-list>

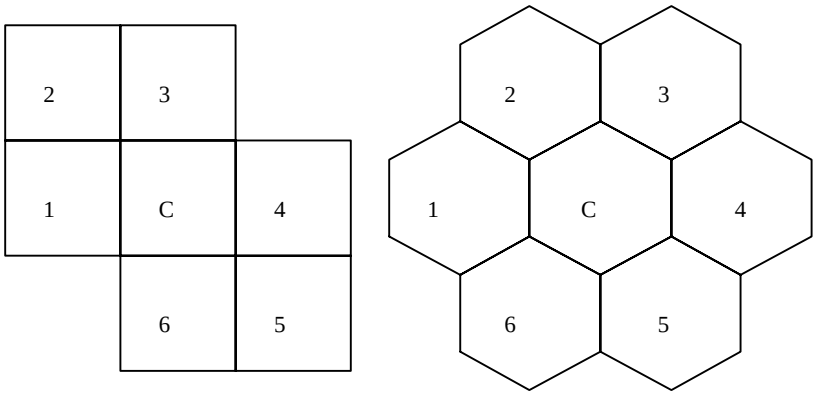


Figure 4.2: The map transforming the square lattice to a hexagonal lattice.

4.5 Cell declarations

Before running a simulation, the locations of the cells that are initially present have to be specified. Here is an example of an initial cell definition:

```
cell {
  type generic; // Tissue type of the cell
  start_up_area rectangle(20,20,21,21); // starts up with area 4
}
```

The following instructions may be used inside a cell definition and serve to elucidate the above example:

- **type** declares the tissue type of the cell. Its general form is:

  `type` <tissue-name>

- **assignment** statement may be used to initialize the biochemical concentrations, as well as other variables. It has the same structure as the assignment previously discussed. Assignments to derivatives of biochemicals are not useful in initialization of cells.

- **start_up_area** indicates the starting size and location of the cell. For the purpose of defining starting locations of cells, the user can specify the points on a square lattice. The CPL implementation uses the map given in figure 4.2 to convert this square lattice to a hexagonal lattice (since cells contain points on the hexagonal lattice). Six of the eight neighboring point on the square lattice map to the neighbors for the hexagonal lattice.

The general form of the `start_up_area` is:

`start_up_area` <object> [`union` <object>]$^+$

where <object> is either `rectangle`, `circle`, `triangle`, or `hexagon`. The union operator enables the specification of composite shapes for the cell.

- **rectangle** declares a rectangular region of points as part of a cell. It takes four integers as parameters (x1, y1, x2, y2), where (x1, y1) are the coordinates of the lower left hand corner, and (x2, y2) are the coordinates of the upper right hand corner. This instruction if present must be part of the `start_up_area` instruction. It takes 4 integer parameters:

  `rectangle` (<integer>, <integer>, <integer>, <integer>)

- **circle** declares a disk of points as part of the cell. It takes three integers as parameters (x, y, r), where (x, y) is the center of the circle, and r is its radius. This instruction if present must be part of the `start_up_area` instruction.

  `circle` (<integer>, <integer>, <integer>)

  The circle produced is a circle on the square lattice. Mapping it to the hexagonal lattice distorts it.

- **hexagon** declares a hexagon of points as part of the cell. It takes three integers as parameters (x, y, r), where (x, y) is the center of the hexagon, and r is its radius. This does not define a unique hexagon, but a family of hexagons. The hexagon with corners on $(x-r, y+r), (x, y+r), (x+r, y), (x+r, y-r), (x, y-r), (x-r, y)$ is chosen because under the map in figure 4.2, it maps onto a regular hexagon. This instruction if present must be part of the `start_up_area` instruction.

  `hexagon` (<integer>, <integer>, <integer>)

  The hexagon on a hexagonal lattice has the property that all the points on its perimeter are equidistant from its center. It corresponds to a circle in the continuous domain.

- **triangle** declares a triangle of points as part of the cell. It takes six integers as parameters, namely the coordinates of the three corner points $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ of the triangle. This instruction if present must be part of the `start_up_area` instruction.

  `triangle` (<integer>, <integer>, <integer>, <integer>, <integer>, <integer>)

- **unit_area** A typical simulation may have thousands of cells. To avoid repeating the initial code for each cell, CPL provides notation for declaring arrays of cells of unit area (i.e. single point cells). If the first statement in a cell definition is the reserved word `unit_area`, then the entire area of the cell is split up into distinct cells, each possessing

the same tissue type and unit area. This enables the definition of arbitrary shaped cellular array (with the constraint that each have unit area). Thus, the following example declares an array of $10 \times 10 = 100$ cells, starting from point (11, 11) to point (20, 20).

```
cell {
  unit_area; // Declares it to be an array of unit area cells
  type generic; // Tissue type of the cell
  start_up_area rectangle(11, 11, 20, 20);
}
```

Cells are initialized with their areas in the order of their definition; thus, if their areas overlap, the cell declared last receives the area in question.

4.6 Accessing cell attributes

In section 2.1, the cell attributes were briefly listed; in this section each of them is discussed in greater detail, and both syntax and semantics are provided.

- tissue type: Although the issue type is a primary attribute, it is not necessary to use its value in the program for the tissue; as a program is written for each specific tissue type, the tissue type is implicitly coded in each instruction of the program.

- `biochemical`: All the biochemicals declared are assumed to be present in all the cells in the simulation space, though possibly in different concentrations. Any valid identifier can be chosen to represent a biochemical concentration. That identifier may be used as a legal variable in all situations, with the caveat that all assignments to this variable are deferred until the end of the simulation time step. All biochemical concentrations are updated simultaneously and synchronously at the end of each time step. Thus, when cells access biochemical concentrations of their neighbors, they are all consistent and independent of the order of execution of the cell programs. It is recommended that assignments to biochemicals use the special `deriv` assignment to highlight this difference.

- `area`: The cell area may be used in different contexts, such as determining if the cell has grown to a sufficient size for division. The cell area is a non-negative integer. Its value indicates the number of discrete points that form the cell's internal representation. It is possible to ignore the internal meaning of the area and use it as a

representation of the relative areas of different cells. `area` is a reserved word and may only be used as a read-only variable. Its value cannot be changed by the assignment statement. There exists a separate instruction called `grow` that can modify this variable.

- `perimeter`: The cell perimeter is the number of discrete points (in the internal representation) on the boundary of the cell. It may be used to determine the fraction of the cell boundary that may be in contact with a particular cell. `perimeter` is a reserved word and may only be used as a read-only variable, whose value is affected by the instructions `grow`, `divide`, and `roundup`. A positive growth would tend to increase the perimeter; negative growth, division, and rounding up tend to decrease the perimeter.

- `location`: The physical location of a cell may be accessed for programming purposes, such as setting up initial conditions. This is guaranteed to be a lattice point belonging to the cell. For cells with unit area, this provides the unique x and y coordinates of the cell's location in direction form (i.e. `point(x,y)`).

- `cell_number`: This provides a unique integer identifying the cell, and images of its value may be used for identifying cell boundaries.

- Neighbor attributes: The physical layout implicitly defines the group of immediate neighbors of each cell. The attributes of a neighbor may be accessed in the cell's tissue program. These attributes are read-only and available only inside the `for_each_neighbor_do` and `with_neighbor_in_direction` instructions.

  - `neighbor.area` is the area of the neighboring cell.
  - `neighbor.contact_length` is the size of the interface with the neighboring cell.
  - `neighbor.perimeter` is the perimeter of the neighboring cell.
  - `neighbor.direction` is the direction to the neighboring cell. This is the direction perpendicular to the boundary between the two cells and is determined by a walk along the boundary with the neighbor. The magnitude of this vector represents the size of the contact.
  - `neighbor.`<biochemical_name> is the biochemical concentration in the neighboring cell of the given biochemical. The biochemical concentration accessed is the concentration in the neighbor at the previous time instance. This may be used to compute the biochemical gradient at a given time, which would help determine the biochemical concentrations at the next time instance.

  - `neighbor.`<variable> accesses the value of the neighbor's variable. Cells may have a variable, say age, that tracks their lifetime in a certain state, and its neighboring cells may access this information (neighbor.age) so as to synchronize their own life cycle.
  - `neighbor.tissue_type` is the tissue type of the neighboring cell. This may only be tested for equality or inequality with an identifier representing a tissue name (i.e. neighbor.tissue_type == Dictyostelium).
  - `neighbor.state` is the current state of the neighboring cell. This may only be tested for equality or inequality with an identifier representing a state name (i.e. neighbor.state == waitForSignal).

- `time`: It is reasonable to assume that cells have some idea of their lifetime. This variable captures the notion of lifetime, by storing the current running time of the simulation. The time attribute is useful for deciding when to switch cell states or perform other actions. It has a real value that starts from 0 and is automatically incremented by the `time_interval` at the start of each time cycle. This is a read-only variable; cell programs cannot modify its value.

- `time_interval`: This is a user defined constant, and it sets the pace of the simulation. A description of its significance is provided in section 4.3.

- `steps`: This is a system variable that provides the number of simulation time steps taken so far. It is equal to the `time` divided by `time_interval`.

- `random`: This provides random numbers, which may be used to take probabilistic actions. $random(x, y)$ is a function call that provides a random integer between x and y. For example, this may be used to differentiate a tissue into two different types with roughly half the cells of each type. Actual cells do not have random number generators, but some of their processes are stochastic, and this probabilistic nature is captured by providing a random number generator.

```
if (random(0,1) == 0) differentiate_to tissueA;
else differetiate_to tissueB;
```

4.7 Other CPL features

In this section, we describe some additional CPL features and provide modeling suggestions that enable the writing of CPL programs.

- The first state in the tissue is the default state. Cells belonging to the tissue start by executing the code of the first state in the tissue. If a tissue has just one state, it need not be named. This is accomplished by omitting the state name declaration.

```
tissue simple {
  // state simpleOne { not needed
  <instruction-list>
  }
```

- A tissue *environment* is defined by default. The lattice points in the simulation space that are not defined explicitly as belonging to a tissue, belong to this environment tissue.

- A cell of tissue type *observer* is defined by default; however, a program for the tissue *observer* has to be defined. This special cell always executes its instructions at the end of every time cycle and may be used to collect results (or display images). In addition, the static variables that are collecting data over the entire cell aggregate may be reinitialized after each time step in the code for the *observer*, since the *observer* always executes after all the other cells have executed for the current time value, and before they execute for the next time value.

- Cells of the same tissue type exhibit similar behavior (with minor differences) even in their different states. CPL contains a feature *like* that enables us to specify if a state is similar to another. This is accomplished by:

```
tissue amoeba {
  state One: like amoebaCore {
      ⋮
  }
  state amoebaCore {
      ⋮
  }
}
```

In effect, this executes the code for the state *amoebaCore* before executing the code for state *One*. Normally, each multi-state tissue should have a core state (like *amoebaCore*) containing all the biochemical interaction and diffusion equations, because

these are, for the most part, invariant during a cell's history.

- We do not model intercellular space implicitly; instead, intercellular space may be modeled by specifying dummy cells which act as intercellular space.
- The chemical concentration of the biochemicals in the environment is assumed to be zero at all times. Control over the concentration of biochemicals in the environment can be achieved by surrounding the cell aggregate by some other tissue type and specifying a program for this user defined tissue type.
- Even though the time step of the simulation can be specified by the user, the discrete simulation, by definition, assumes a "small" step size. In particular, the largest time step for running the simulation is limited by the fact that diffusion in a time step should at best equalize the concentration between neighbors. The biochemical should not fluctuate in a cell between extremes with every time step. In that case, diffusion is being modeled incorrectly.

4.8 Implementation

Our exercise of writing programs for developmental behavior began with writing special purpose programs for various developmental phenomena (notably, cell segregation and engulfment). However, we soon realized that these programs were for the most part similar, and it should be possible to write simpler descriptions (in program form) of this developmental behavior. A small lexical analyzer (using lex) and a parser (using yacc) were written for CPL. The initial CPL programs were converted into quadruples, which were in turn interpreted. The interpreter was written in C++. Eventually, the interpreter was discarded in favor of a CPL to C++ translator to improve execution speed. The current implementation provides a library of functions for CPL instructions. The parser replaces CPL instructions with calls to C++ functions. These are then compiled and linked with the library to produce fast optimized code for the specific application.

4.8.1 Function libraries

Two libraries are provided: the first library (called UNIT) is optimized for CPL programs that only use cells with unit area, and the second library (called MULTI) provides functions equipped to handle cells with non-unit area. UNIT provides greater simulation speed and has been extensively tested; MULTI is more general and still awaits serious biological applications.

For unit area cells (UNIT), the instructions grow, divide, and roundup are not needed. Each cell has a fixed perimeter, which makes it trivial for cells to identify their neighbors.

If a cell moves, it is easy to identify (and tag) what other cells it may have effected, and only those cells need to recompute their neighborhoods.

In the general case (MULTI), a single cell operation (move, grow, divide, or roundup) may affect the geometry of a large number of cells. Thus, each operation may necessitate recomputation of a large number of cell boundaries. Overhead is also involved in ensuring that the cells remain connected, and that their areas are correct, since operations on other cells may change a cell's area.

4.8.2 Cell execution order

The implementation of CPL is sequential. At each time instant the code for all the cells is executed in random order. This reduces any effects that may be introduced by executing the code for the cells in a specific order.

4.8.3 Stability of neighborhoods

Before executing the code at any time instant, a cell recomputes its neighborhood to account for any changes that may have taken place. This is reasonable, if a sizable number of cells move at each time instant. If only a few of them move at each time instant, a more efficient scheme is utilized. In this case, neighborhoods are only recomputed if they may have been modified. Each time a cell moves, it tags all its old and new neighbors, informing them of possible changes in their neighborhood. Only such tagged cells recompute their neighborhood. A compiler directive enables the user to choose between the two alternatives. The default is to recompute the neighbors at each time instant.

The recomputation of the neighborhood is accomplished by selecting a neighbor at random, and then traversing the boundary and forming a list of neighbors along the boundary. Thus, unless the neighborhood is recomputed, the `for_each_neighbor_do` instruction will access the neighbors in the same order at every time step.

4.8.4 Choice of topology

The hexagonal topology is the default. For multiple area cells, this is the only well defined topology. However, for unit area cells, the user may choose the eight neighbor topology with a compiler directive.

4.8.5 Cell shape and area modification

In this subsection, we discuss the implementation of CPL instructions that may effect the shape or area of some cells. The implementations are difficult only when at least some cells have non-unit area.

Move

A cell can only move to a location occupied by another cell. [10] The move instruction is implemented as a complete exchange of the lattice points occupied by the two cells. The implementation is trivial if both the cells have equal area. If they have unequal area, then the move temporarily changes their area; however, the original area is restored by growing (shrinking) the cells by the requisite amounts. This may modify the shapes of the cells involved, and possibly other cells[11].

A move, where cells of unequal area are involved, is computationally expensive, since it requires recomputation of neighborhoods for a potentially a large number of cells.

Divide

The divide instruction should split the lattice points that comprise the cell equally between the two daughter cells. Thus, each daughter cell should end up with half the lattice points. The division should also ensure that the two daughter cells are connected. A cell is connected if and only if between each pair of lattice points (l_0 and l_n) belonging to the cell:

- There is a path of lattice points $(l_0 l_1 \ldots, l_{n-1} l_n)$ belonging to the cell, where $l_1, \ldots, l_{n-1} \in$ cell.
- On the path of lattice points, l_i and l_{i+1} (for all $i = 0, 1, \ldots, n-1$) should be neighbors on the hexagonal grid.

The divide is implemented by finding the two extreme points on the cell in a direction perpendicular to the choice of dividing line. Thus, if a horizontal dividing line is desired, the topmost and the bottommost points on the cell serve as the seeds for the daughter cells. Lattice points from the mother cell are added to the daughter cell alternatively. Only lattice points which are neighbors of the lattice points already belonging to the daughter cells may be considered for addition to the daughter cell. This strategy ensures that the two daughter cells are connected, and halts when no more lattice points may be added to one of the daughter cells. The procedure may halt due to two reasons. All the lattice points in the mother cell may have been divided (which ends the divide procedure), or one of the daughter cells may have cut off the path (for the other daughter cell) to the remaining lattice points in the mother cell. In the second case, the remaining lattice points in the mother cell are added to the daughter cell that has access (by means of a path) to them. Subsequently, the grow instruction is employed to correct the imbalance in area between the two daughter cells (by reducing the area of one and increasing the area of the other).

[10] The space between cells is modeled as just another cell type.

[11] Other cells are affected, since the grow instruction has a non-local effect.

Grow

An implementation of growth is accomplished by walking from the cell's center of mass in the chosen direction until its boundary is encountered; the boundary point so encountered (belonging to another cell) is stolen. This cell in turn steals a point from its neighbor by performing a walk in the same direction from its center of mass. With this implementation, if a rectangular cell is grown horizontally, the cell does not remain a rectangle; rather, only its center line expands. This does not have the desired effect, since we would like to be able to grow cells so that they maintain their shape. We also note that this implementation of grow is an extension of the implementations proposed in section 3.3.1.

An alternative improved implementation[12] of growing a cell in a direction is by computing its boundary and randomly picking the requisite number of points on the boundary. The chosen point should be such that their neighboring point in the direction of growth belongs to a different cell. This cell then steals these neighboring points. The neighboring cells then steal from their neighbors in turn until the effect ripples out of the cell aggregate. With this implementation, if a rectangular cell is grown horizontally, all the points on the vertical edges would have an equal chance of being chosen, resulting in a rectangular cell.

This implementation is not perfect either, since the outward ripple effect alters the shape of the exterior cells.

Roundup

The implementation of `roundup` is based on an extension of a algorithm proposed by Goel and Rogers, in 1978 [GR78]. Their procedure for non-local exchange of cells was designed to cause segregation and engulfment of tissues[13]. We have modified it to cause cells to round up. This is not surprising, since the same adhesive force is responsible for cells rounding up, and tissue segregation and engulfment.

Description of the rounding up procedure:

1. Consider the boundary lattice points on a cell that are shaded dark in figure 4.3. These points have a varying number (P) of adjoining points belonging to the cell. Since we use a hexagonal topology, P varies between 1 and 5[14]. Order these points in increasing order of P. Observe that points with low P are responsible for jagged cells, and removing them from the cell would round up the cell. However, this would reduce the area of the cell.

2. Now consider the boundary lattice points on a cell that are shaded light in figure 4.3).

[12] This is the currently available implementation.
[13] This is discussed in section 3.2.3
[14] Points on the boundary cannot have 6 cell point neighbors.

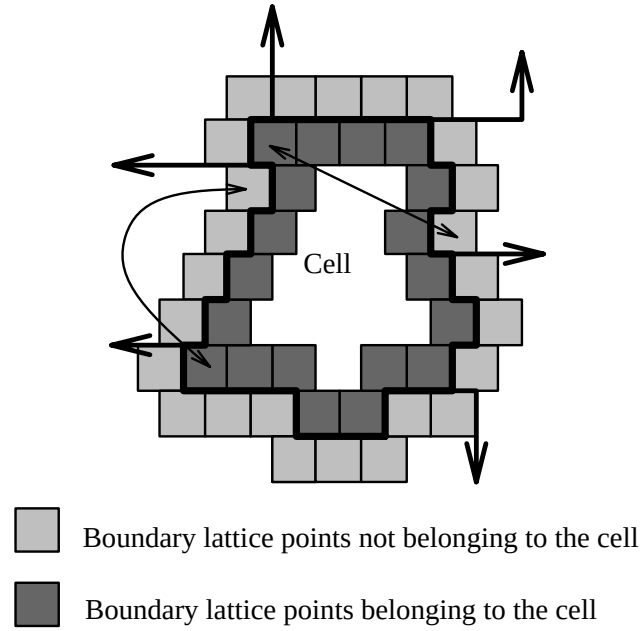


Figure 4.3: Cell rounding up. The thick line is the cell boundary. The cell is comprised of the dark shaded cells on the boundary and the light interior. The thin double-edged lines indicate possible lattice point exchanges that will round up the cell.

Consider their P number (the number of adjoining lattice points belonging to the cell to be rounded up), which again varies between 1 and 5. Order these points in decreasing order of P. Observe that points with high P are responsible for incisions into the cell under consideration, and adding them to the cell would round up the cell. However, this would increase the area of the cell and reduce the area of a neighboring cell.

We combine the previous two steps to balance the additions and deletions of lattice point to the cell. This rounds up the cell under consideration. The lattice points deleted from the cell are added to a neighboring cell. We keep track of additions and deletions to neighboring cells, and if they don't cancel out, we invoke the grow procedure for the cell to correct its area. This grow invocation can undo the rounding up, and thus repeated invocation of round up may be required to achieve cohesive cells. In practice, the rounding up performs well.

In this chapter, we have discussed the details of the *Cell Programming Language*. In the next few chapters, we explore its power through various biological applications.

Chapter 5

Aggregation in slime mold

5.1 Theory

Dictyostelium discoidea, a free-living amoeba, is often called the hydrogen atom of developmental biology. It has an intriguing life cycle. In its vegetative cycle, solitary amoebae eat and multiply. Upon exhausting their food supply, tens of thousands of these amoebae join together forming moving streams of cells that converge at a central point. There they form a conical mound, which eventually absorbs all the streaming cells. This amoeba aggregate bends over to produce a migrating slug. The cells in the slug differentiate into two varieties: stalk cells and spore cells, which together form a fruiting body. The spore cells disperse, each one becoming a new amoeba.

The cellular aggregation is not due to a simple radial movement. Rather, cells join with each other to form streams; the streams converge into larger streams, and eventually all streams merge in the center. Sometimes amoebae will even move away from the center to join a stream. This directed amoeba motion has been shown to be due to chemotaxis (movement along a chemical gradient), the chemical involved being cyclic adenosine monophosphate (cAMP). There is no dominant cell or predetermined center. Neighboring cells respond to the cAMP in two ways: they initiate a movement towards the cAMP pulse, and they release cAMP of their own. Following this stage the cell is unresponsive to further cAMP pulses for several minutes [Gil91].

This life cycle has been well researched, and there is a wealth of experimental data available [Bon67,AM74,New77,Rap84]. Tomchik and Devreotes observed the cAMP wave patterns, and accumulated quantitative data on them [TD81]. Wessels et al. have conducted experiments on measuring *Dictyostelium* response to cAMP waves [WMS92]. They concluded that *Dictyostelium* amoebae respond primarily to the leading edge of the cAMP

wave (temporal gradient), rather than to the absolute value of cAMP or the spatial gradient of cAMP[1]. Martiel and Goldbeter have presented a model based on cAMP-receptor desensitization, which explains both the relayed and autonomous pulses of cAMP [MG87]. Tyson and Murray have built mathematical models and conducted numerical simulations on the cAMP wave patterns [TM89]. Cohen and Robertson have mathematically analyzed the aggregation process suggesting that signalling delay (and not intercellular diffusion) limits wave velocity [CR71]. They also show that the amoeba concentration has a critical density, below which cAMP waves cannot propagate, and thus aggregation cannot occur.

Parnas and Segel simulated the aggregation of 40 amoebae arranged in a row (one-dimensional) [PS77,PS78]. Novak and Selig conducted simulations on a two-dimensional (49×49) aggregate of slime mold cells [NS76]. Mackay has conducted extensive simulations on various aspects of slime mold aggregation [Mac78], with images that closely mimic *Dictyostelium* aggregation. Our results are similar to those obtained by Mackay, but are achieved using a versatile discrete model.

5.2 Aggregation: a quantitative examination

Dictyostelium amoebae are typically $10\mu m$ in diameter, and their aggregation may involve up to 100,000 amoebae from as far as 20 mm. The initial stimulus needed to initiate aggregation is starvation. This induces in some cells the ability to produce slow rhythmic pulses of cAMP with an initial frequency of approximately 1 pulse every 7–10 minutes. Meanwhile, the rest of the starving population produce cAMP receptors on their surface which enables them to receive the pulsed signal. The cAMP signal does not diffuse far from the centers of its production but is destroyed within $57\mu m$ ($\approx$ 6 cell diameters) by acrasinase (a phosphodiesterase enzyme) also produced by the starving amoebae.

An amoeba receiving a cAMP signal responds by moving in the direction of the signal source for 100 seconds covering $20\mu m$, and the amoeba itself emits a pulse of cAMP approximately 12 seconds after receiving the signal (signalling delay). The amoebae respond to the leading edge of the cAMP wave (they are positive edge triggered). The trailing edge elicits no response from the amoebae. The amoeba's own cAMP signal bolsters the cAMP wave. Eventually the wave diffuses away. The amoeba then waits for the arrival of a new wave. By this system of relay, a series of waves of cAMP production, destruction and response, move outward from the center as the amoebae move inward. Due to the relay of the pulse by the responding amoebae, the amoebae tend to gather into streams. The streams increasingly act as strong local sources of attraction. Amoebae can, at times, even be observed to move outward from the center in order to join a stream that happens to run

[1] In an earlier technical report [Aga93], we have reported the aggregation results assuming the amoebae respond to the spatial gradient.

behind them. Eventually, the amoebae in these moving streams all reach the aggregation center, and a conical mound of cells is formed. This summary of the aggregation procedure has been taken from Newell[2] [New77].

5.3 Model

We wrote programs in CPL to model this aggregation. We considered each amoeba to be a square with $10\mu m$ sides. We modeled each *Dictyostelium* amoeba by a single point in our discrete space. Typically, we used a discrete space of 100×100, which translates to $1mm \times 1mm$ of real space. The number of amoebae is about 10% of the total points, about 1000, giving them a density of $10^3\ mm^{-2}$, or $10^5\ cm^{-2}$, which is also a typical density for laboratory experiments.

The concentration of cAMP varies between $10^{-8}M$ and $10^{-6}M$. The signal duration and the strength of the signal are not known; however, the cAMP waves experienced by the ameobae have been described by Tomchik and Devreotes [TD81]. In fact, the cAMP waves (also termed pulses) resemble the positive half of a sine wave with a width of 1 to 3.3 minutes, and a period of 7 minutes. The waves used in the CPL program are shown in figure 5.1. The cAMP signal duration and strength, and phosphodiesterase activity for the simulation, were selected to produce cAMP waves with the same high and low concentrations (between $10^{-8}M$ and $10^{-6}M$), and duration (1 to 3.3 minutes). Phosphodiesterase removed a constant fraction (1/12) of the cAMP above the base value ($10^{-8}M$) in an amoeba[3]. The cAMP signal strength of each amoeba was $33 \times 10^{-8}M$, and it lasted for 60 seconds. This allowed the concentration of cAMP to build up to about $10^{-6}M$. The exact peak concentration depends upon the density of the amoebae[4].

Figure 5.1 exhibits the typical cAMP concentrations within an amoeba that has just begun releasing cAMP, and at a distance of $60\mu m$ from the amoeba. The cAMP pulse is observed to weaken in intensity, due to diffusion, with increasing distance from the firing amoeba.

Only amoebae within $57\mu m$ from a firing amoeba can detect and react to the cAMP pulse. This translates to approximately 6 cell diameters; thus, amoebae within 6 discrete points of a firing amoeba in our simulation space can react. Figure 5.2 exhibits the maximum cAMP concentration, spatial gradient, and temporal gradient as a function of cell distance from a single firing (cAMP releasing) amoeba. From the temporal gradient graph, we

[2]Some additions and subtractions have been made to the summary.

[3]If we assume a *Dictyostelium* amoeba to be a cube with sides of $10\mu m$, then a concentration $10^{-8}M$ corresponds to about 6000 molecules per cell. Since all the amoebae have identical volume in our model, we can use concentrations instead of number of molecules for diffusion purposes.

[4]In our representation, we scaled the concentration of cAMP by 10^8, thus instead of varying between $10^{-8}M$ and $10^{-6}M$ it varies between 1 and 100.

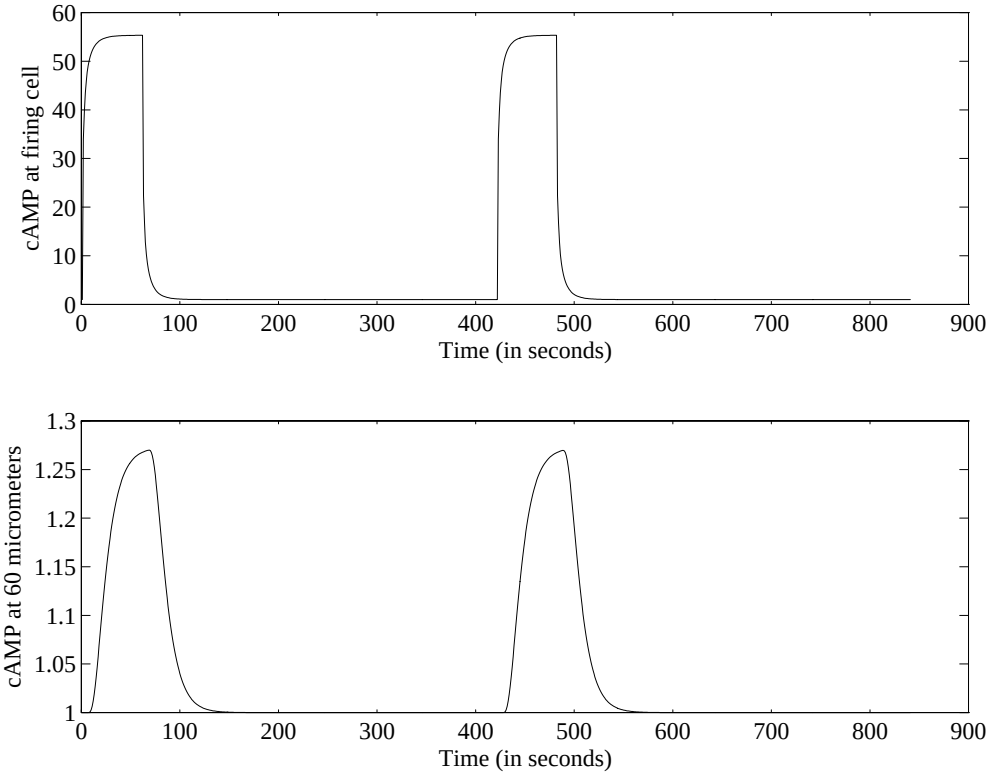


Figure 5.1: cAMP signal strength in $10^{-8}M$ as a function of time at the firing amoeba (top), and $60\mu m$ from the firing amoeba (bottom). The y-axis scale is different for the two graphs.

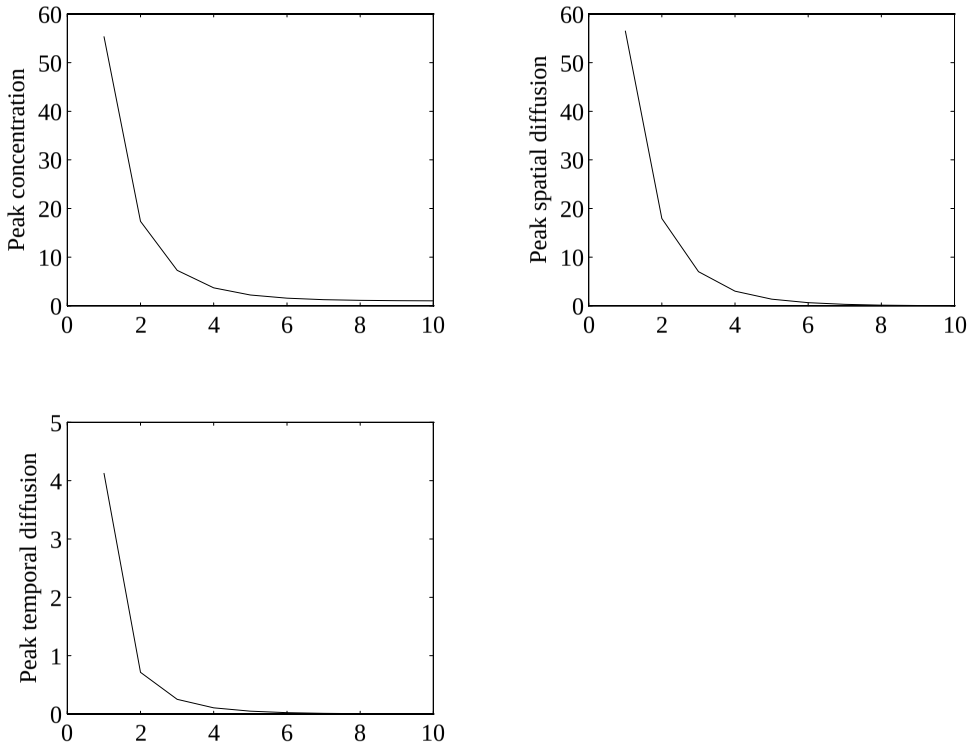


Figure 5.2: cAMP signal peak strengths as a function of distance (in cell diameters) from a single firing cell.

determine that an amoeba should be able to detect a $0.02 \times 10^{-8} M/second$ change in the cAMP level. This is the temporal gradient experienced by an amoeba 6 cell diameters away, and *not* experienced by an amoeba 7 cell diameters away.

Figure 5.3 exhibits the traveling cAMP wave with an amoeba density of $10^5\ cm^{-2}$. Each of the subplots reveals the cAMP concentration at different distances from the initially firing cell.[5] Since the pulse is relayed outward, it takes longer to reach cells further away. The subplots represent the traveling of the cAMP wave in space. The strength of the cAMP wave varies depending upon local *Dictyostelium* density. The waves in our simulations are observed to travel[6] at about 150 $\mu m/min$. This wave speed depends mainly on the signalling delay, and the time taken for cAMP to reach the threshold level at cells $57\mu m$ away from the firing cell by the process of diffusion. The *Dictyostelium* density also has a

[5] The initial firing cell acts as the aggregation center.

[6] This is half of the experimentally measured wave speed 300 $\mu m/min$. It is possible to conduct simulations with the experimentally measured wave speed, but that requires speeding up diffusion, which is only possible by slowing down the simulation considerably.

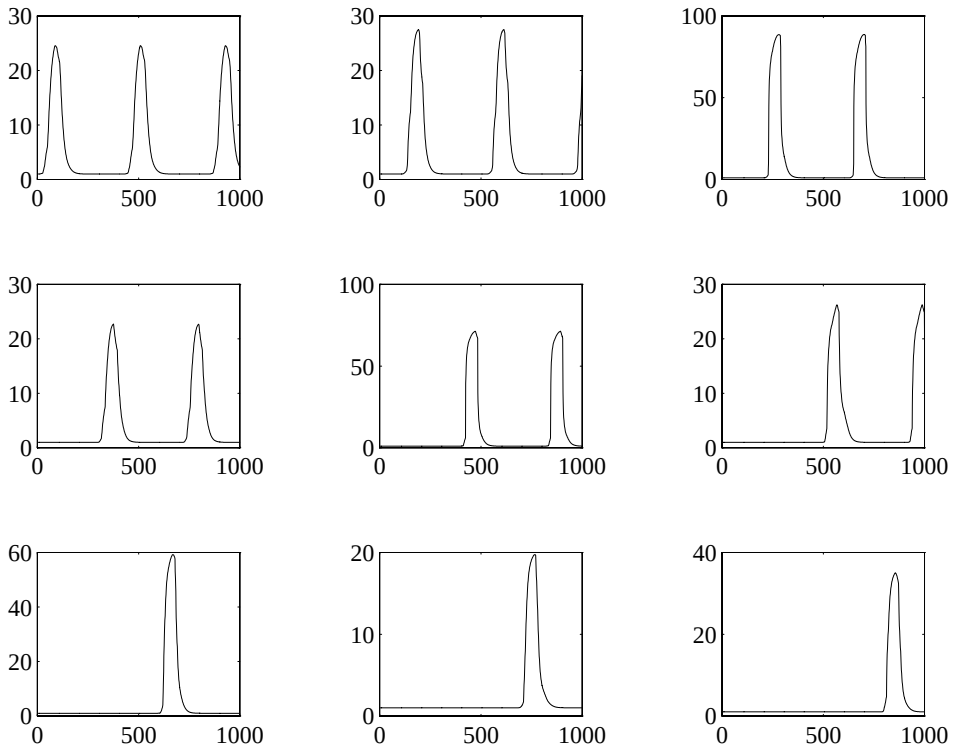


Figure 5.3: cAMP concentration as a function of time (in seconds). Successive subplots display cAMP concentration at points 18 cell diameters further away from the first firing cell.

minor effect on wave speed.

Each time step in our simulation represents 1 second. The signalling delay is 12 seconds; the amoebae start releasing cAMP 12 seconds after they detect the cAMP wave. The cAMP receptors on the cells require time to recover from a cAMP wave before they are able to detect a new wave, which is termed the relay refractory period and ranges from 3 to 7 minutes. In response to the cAMP wave, the amoebae move $20\,\mu m = 2$ cell diameters in about 100 seconds. The amoebae also have a random instantaneous velocity of $5\,\mu m/min$ (observed by sampling the displacement every half second) [WMS92]. This does not cause much overall displacement, but a random motion does help the aggregation process[7], and we model it as 1 cell diameter every 2–4 minutes.

[7]Random motion makes the effective density of amoebae higher, since due to random motion amoebae will stray into the cAMP relaying field, and act as beacons for other amoebae. This enables aggregation over larger territories.

5.4 Program

CPL program 5.1 (pages 59–61) has been used to produce the aggregation images in figure 5.1. It describes a tissue of type generic, which is rectangular in shape (101×101). The outermost cells in this tissue die; thus, their biochemical concentration are frozen. They act as a barrier to diffusion to the environment. A tenth of the inner cells are then chosen at random to act as *Dictyostelium*, while the remaining act as space cells. These space cells are conduits for diffusion, and remove cAMP at each point due to the action of the phosphodiesterase enzyme. These properties of the space cells are also properties of all other cells (*slimeMold* and *PaceMaker*).

A central cell is chosen to act as a pacemaker (an autonomous source of cAMP). Its job is to release cAMP every relay period (*RelayPeriod* = 420 seconds) for the signal duration (*SignalDuration* = 60 seconds).

The slime mold cells cycle between three states. Each amoeba maintains a clock, whose value determines after how much time will the amoeba be responsive to another cAMP signal. This clock is decremented after each time interval. A zero or negative value indicates the amoeba is able to respond to cAMP. A positive clock indicates that the amoeba has received a cAMP signal and is in the process of chemotactic movement and cAMP pulse relay. Upon receiving a cAMP pulse the amoeba's clock is set to the *RelayPeriod*, indicating the time for which it will be unresponsive to further cAMP pulses. Once the clock falls to a value *RelayPeriod* − *SignalDelay* the amoeba starts signalling (i.e. relaying) the cAMP pulse, and the amoeba relays as long as the clock is greater than *RelayPeriod* − *SignalDelay* − *SignalDuration*. The *RelayPeriod* of the amoeba is decremented by 10 seconds after every relay cycle.

All the other states are *like* the *core* state. In the core state, the amoeba computes the change in cAMP due to reaction with phosphodiesterase, and the diffusion. If the amoeba has received a cAMP pulse (clock > 0), then it also moves about two cell diameters every relay period in the direction of the source of the cAMP signal. It also moves one cell diameter every two minutes in a random direction.

The three states that the amoeba cycles between are:

- *waitForSignal*: waiting for a cAMP signal to arrive (detected by the temporal gradient of cAMP being greater than 0.02);
- *readyToSignal:* waiting a short period (*SignalDelay*) until ready to relay the signal;
- *signal*: relaying the signal for *SignalDuration*, and then reverting to the state of waiting for a signal.

Differences in CPL programs for the various simulation images

For simulation image 5.2, a higher cAMP wave speed is obtained by multiplying the time parameters (*SignalDuration*, *SignalDelay* etc.) by a factor of ten. This causes the wave to spread out much quicker, and the signal delay is the major limiting factor.

In simulation images 5.4 and 5.5, there is no amoeba signalling autonomously; instead, one of the amoebae releases a single pulse of cAMP to initiate the aggregation. Only the amoebae in the bottom half of the array start with zero clock (ready to respond to cAMP), the top half do not respond to cAMP signals for until 400 seconds. The initial relay period is also set to 600 seconds, instead of the 420 seconds in the other simulations.

```
#define SignalStrength (33)
#define BaseCAMP (1)
#define SignalDuration 60
#define OutputPeriod (500)
#define SignalDelay 12
#define FACTOR 8.0
#define TAU 12.0

simulation_size (102,102);
biochemical cAMP;
float cAMP,cAMPchange,clock;
vector cAMP_dir,r_dir;
integer RelayPeriod;

tissue generic{
 if (location == point(51,51))
   differentiate_to PaceMaker;
 for_each_neighbor_do
   if (neighbor.tissue_type == environment) die;
 if (random(1,100) <= 10)
     differentiate_to slimeMold;
 else differentiate_to space;
}

tissue space{
  deriv cAMP = -(cAMP-BaseCAMP)/TAU; // cAMP removal due to phosphodiesterase
  for_each_neighbor_do
    deriv cAMP = (neighbor.cAMP - cAMP)/FACTOR; // diffusion
}
```

```
tissue slimeMold{
  state waitForSignal:like core {
    // amoeba wait in this state for cAMP signal to arrive
    if (cAMPchange > 0.02 && clock < 0) {
      // This value reaches cells up to 57um away
      // once a signal arrives they determine the direction of the signal
      cAMP_dir = point(0,0);
      for_each_neighbor_do
        cAMP_dir += neighbor.direction * (neighbor.cAMP-cAMP);
      clock = RelayPeriod;
      goto readyToSignal;
    }
  }
  state readyToSignal:like core {
    if (clock <= RelayPeriod-SignalDelay) goto signal;
  }
  state signal:like core {
    deriv cAMP = SignalStrength;
    if (clock <=  RelayPeriod-SignalDelay-SignalDuration){
      RelayPeriod -= 10;
      if (RelayPeriod < 180) RelayPeriod = 180;
      goto waitForSignal;
    }
  }
  state core{
    clock -= time_interval;
    deriv cAMP = -(cAMP-BaseCAMP)/TAU;
    cAMPchange = 0;
    for_each_neighbor_do
      cAMPchange += (neighbor.cAMP- cAMP)/FACTOR;
    deriv cAMP = cAMPchange;
    if (random(1, RelayPeriod) <= 2 && clock > 0) // move twice every relay period
      with_neighbor_in_direction cAMP_dir
        if (neighbor.tissue_type == space) move cAMP_dir;
        else clock = neighbor.clock;
    if (random(1,240) <= 1) { // one cell diameter every 4 minutes
      r_dir = random_direction;
      with_neighbor_in_direction r_dir
        if (neighbor.tissue_type == space) move r_dir;
    }
  }
}
```

```
tissue PaceMaker{
  clock -= time_interval;
  if (clock < 0)
    deriv cAMP = SignalStrength;
  if (clock + SignalDuration <=0)
    clock = RelayPeriod-SignalDuration-SignalDelay; // quit signalling
  deriv cAMP = -(cAMP-BaseCAMP)/TAU;
  for_each_neighbor_do
    deriv cAMP = (neighbor.cAMP - cAMP)/FACTOR;
}

tissue observer{
  if (int(time) mod OutputPeriod == 1)
    image state;
}

cell {
  unit_area;
  type generic;
  start_up_area rectangle(1,1,101,101);
  cAMP = BaseCAMP;
  clock = 0;
  cAMP_dir = point(0,0);
  RelayPeriod = 420;
}
```

CPL program 5.1: Aggregation in *Dictyostelium*.

5.5 Simulation results

Simulation images 5.1 (page 63) and 5.2 (page 64) show *Dictyostelium* aggregation results. The pacemaker is located in the center for each subframe. Both the simulation images contain about a thousand amoebae in an array of 100×100. The only difference between the two is the wave speed. In the second one, the wave speed is closer to what is experimentally observed. However, the first simulation takes a tenth of the simulation time. The aggregation results are very similar in both cases, and the wave speed seems to have little or no effect on the aggregation in small territories ($1mm \times 1mm$).

The effect of the choice of hexagonal topology is quite evident in these images. The streams have a tendency to form along the six preferred directions. However, the succeeding images illustrate that the streams are not always formed along the six directions dictated by the choice of topology.

Simulation image 5.3 (page 65) exhibits the aggregation with lower amoeba density (only ≈ 300 amoebae in the same sized territory $1mm \times 1mm$). In this simulation image, the streaming is more evident, and the hexagonal topology does not seem to have an effect on the streams. Some streams seem to bend away from the center to join a larger stream. The local distribution of the amoebae determines the location of streams. Due to the limited signalling range at lower densities, all amoebae do not receive the relayed signal from the central pacemaker, and some amoebae on the fringe do not join the aggregation process at this stage. Thus, if the amoebae are present in lower densities then the aggregation territories are smaller,

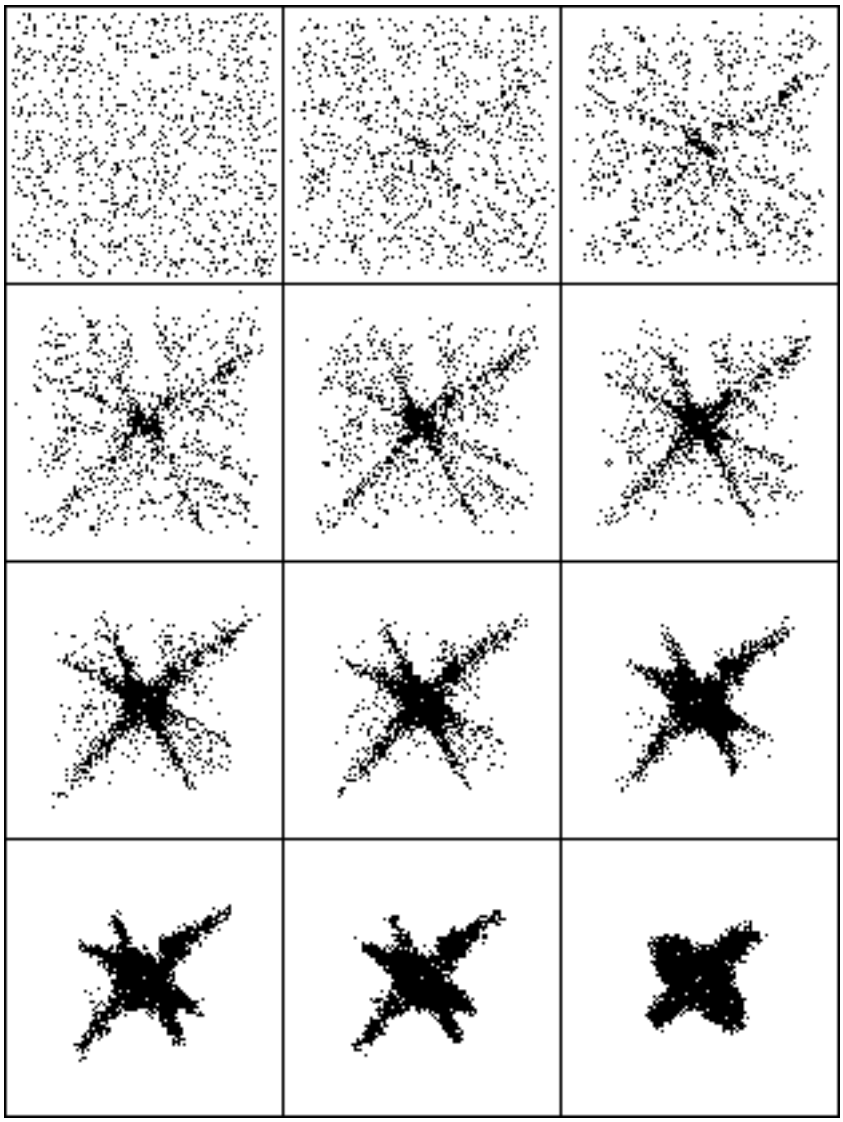
The streaming is a result of the limited signalling range of the amoebae. Decreasing the signalling range increases the amount of streaming seen. However, a smaller signalling range makes signal propagation more difficult, and the signal may not reach the more isolated amoebae. Thus, a higher density is needed for propagation to take place.

Simulation images 5.4 (page 66) and 5.5 (page 67) show *Dictyostelium* aggregation results with no autonomous pacemaker. Both these simulation images contain about a thousand amoebae in an array of about 200×60. The *Dictyostelium* in the top half of the array are not responsive to cAMP until 400 seconds of the simulation[8]. The cAMP wave passes only through the *Dictyostelium* in the lower half of the array. After 400 seconds, the *Dictyostelium* in the top half become responsive, and the cAMP wave circles back. It travels in opposite directions in the top and bottom halves. Thus, a self-sustaining cAMP wave may be set up. This leads to a rotating stream of amoebae, with other streams spiraling into it. Eventually, the rotating stream collapses. The rotating stream requires that there are no *Dictyostelium* within it; however, this need not be artificially created. In fact, such holes arise naturally due to the streaming process and the rotating cAMP wave.

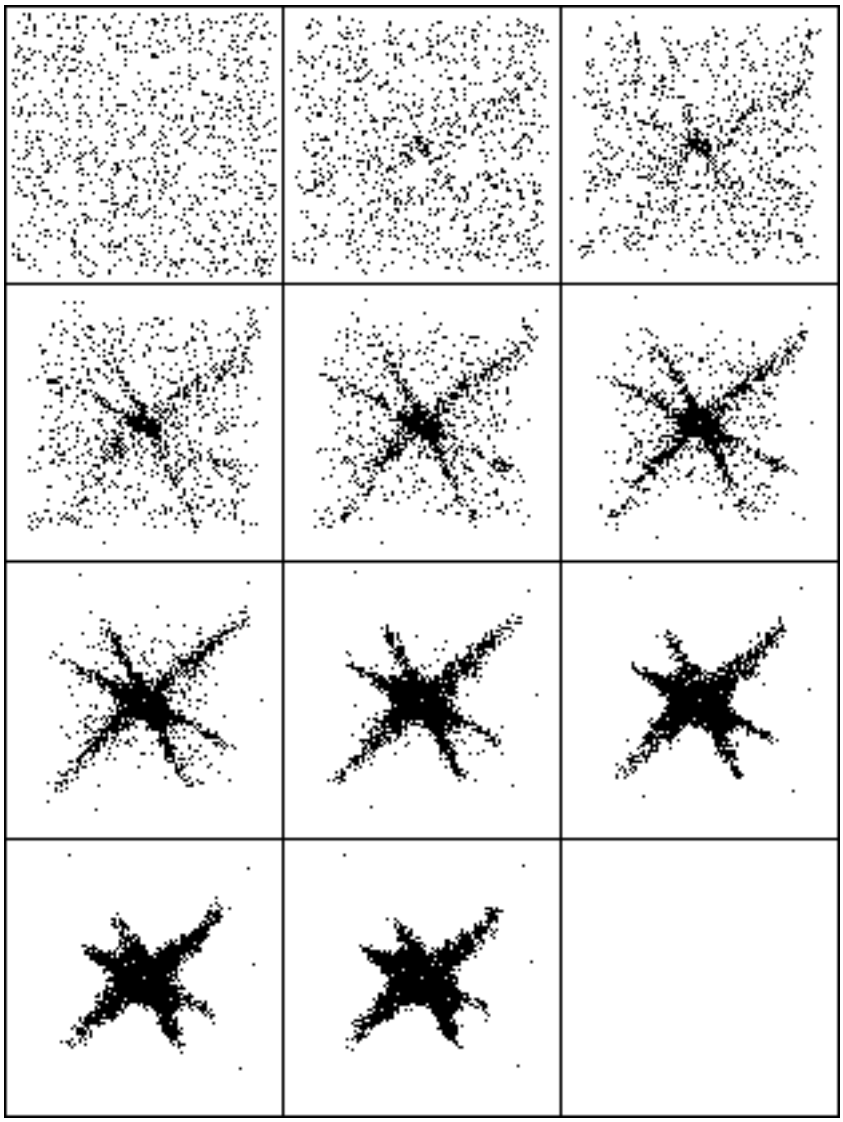
Distributing the amoebae into disjoint halves, such that the amoebae in the top half are not responsive until 400 seconds, is possibly an artificial initial situation. However, similar results are obtained when unresponsive amoebae are randomly interspersed with responsive amoebae. Mackay's simulations yielded very similar results [Mac78]. The spirals are observed in only some of the aggregations, and if the responsive and response-delayed amoebae are interspersed, the spirals are not as prominent and seem to collapse sooner.

In simulation image 5.4 (page 66), the amoeba that initiates the aggregation by releasing a cAMP pulse is located in the center of the array $(100, 30)$. In simulation image 5.5 (page 66), the amoeba that initiates the aggregation by releasing a cAMP pulse is located in the left center of the array $(30, 30)$. In simulation image 5.5, the aggregation territory breaks up into two halves with two rotating cAMP waves.

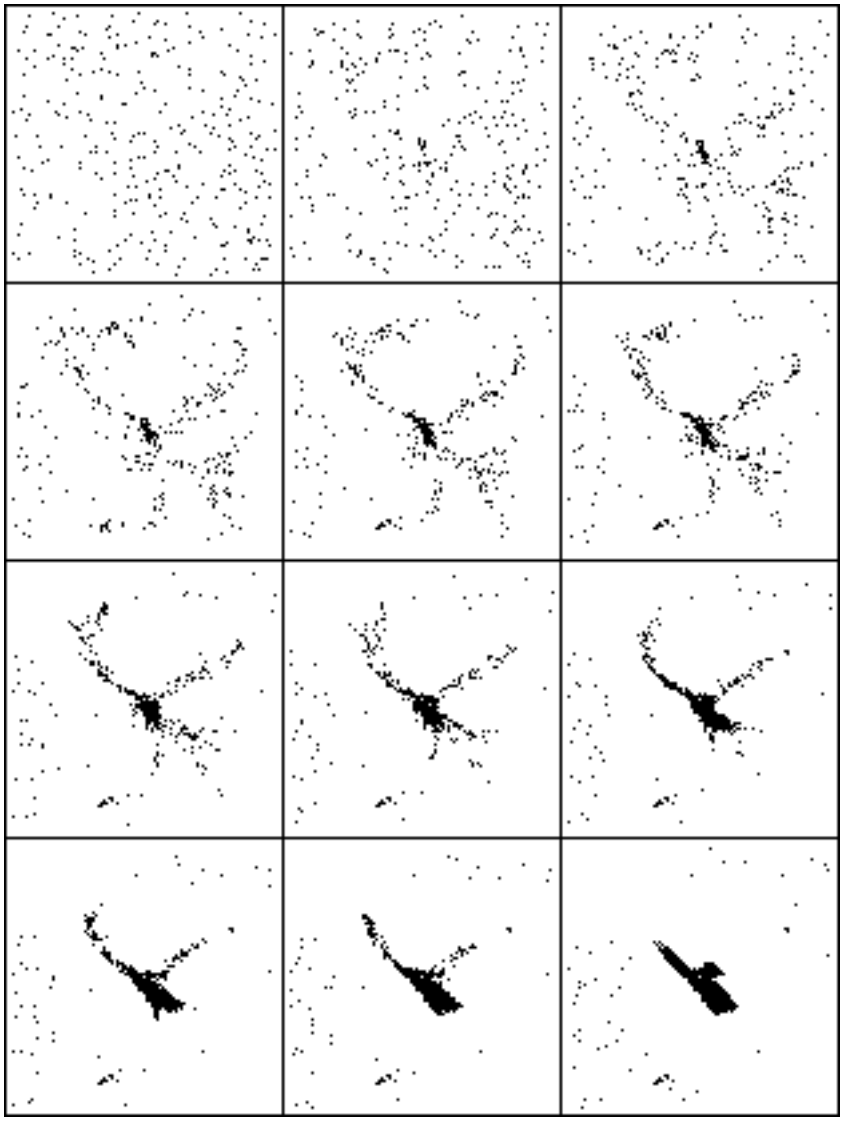
[8] The relay period is 600 seconds in these simulations.



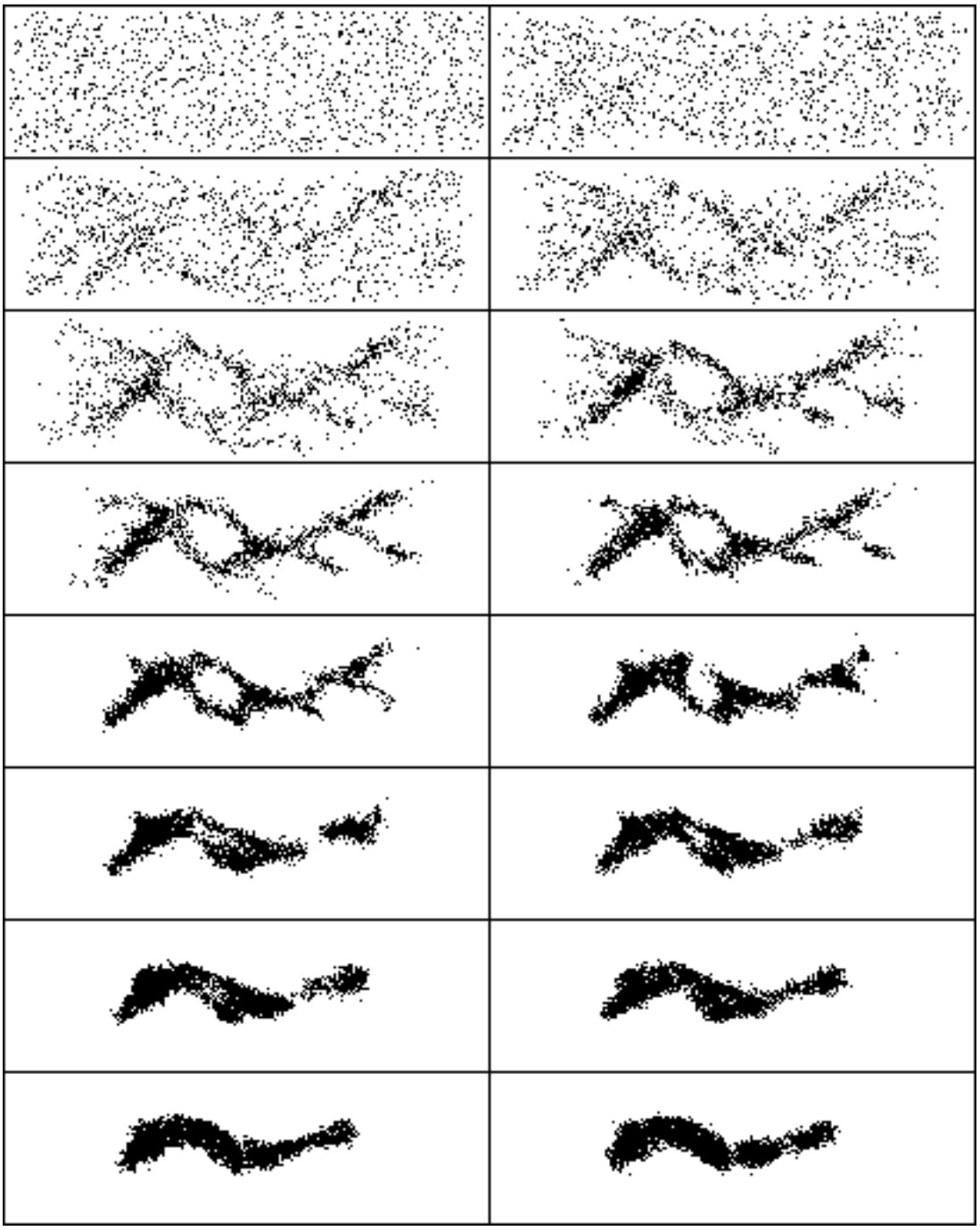
Simulation image 5.1: Aggregation in *Dictyostelium*. Each point represents a *Dictyostelium* amoeba. The subframes are images taken after every 1000 seconds of the simulation run. The last subframe shows the final aggregate and is after 20,000 seconds of the simulation.



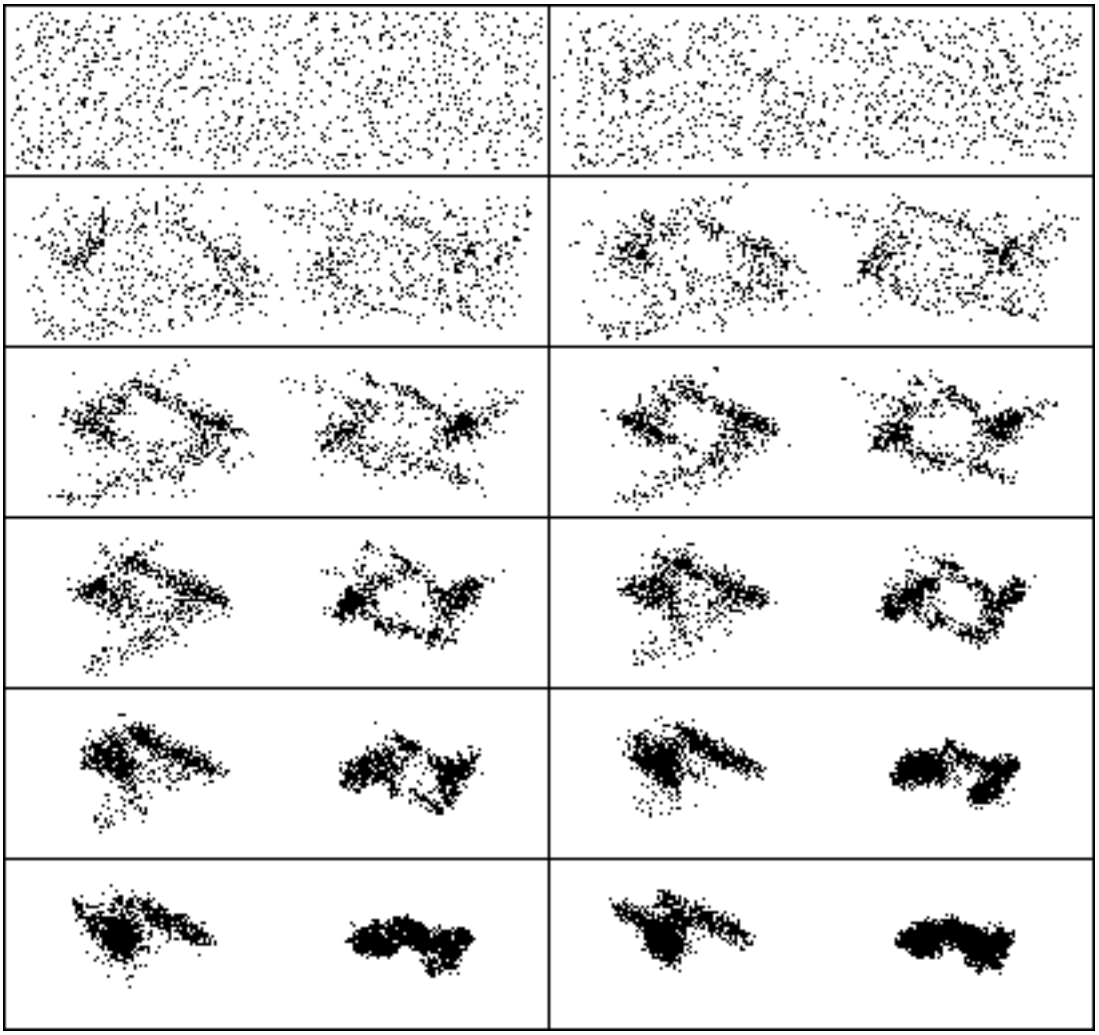
Simulation image 5.2: Aggregation in *Dictyostelium*. Almost identical aggregation is observed by doubling the wave speed to 300 μm. All other parameters are the same as in the previous simulation image.



Simulation image 5.3: Aggregation in *Dictyostelium* present at low density. Only 3% of the points are occupied by amoebae, thus their density is $0.3 \times 10^5\ cm^{-2}$. The subframes are images taken after every 1000 seconds of the simulation run. The last subframe shows the final aggregate and is after 20,000 seconds of the simulation.



Simulation image 5.4: The amoebae in the top half of the first subimage do not react to the cAMP until a later time. This sets up a rotating cAMP wave throughout the aggregate, causing the amoebae to spiral (clockwise) around an spontaneously created empty-center. The subframes are images taken after every 2000 seconds of the simulation run.



Simulation image 5.5: Two spirals are created by choosing a different aggregation initiating amoeba. The left spiral rotates anticlockwise and the right one clockwise. The subframes are images taken after every 2000 seconds of the simulation run.

5.6 Simulation speed

A simulation for a thousand *Dictyostelium* amoebae on a lattice of 100×100 points for 100 time steps on SPARCstation-10 takes about 30 seconds. Thus, typical aggregation runs of 20,000 seconds (or time steps) can be completed in about 100 minutes of CPU time[9]. The running time for the *Dictyostelium* simulation for a lattice size of 200×200 for 100 time steps is about 120 seconds. Profiles of the program indicate that most of the compute time is spent calculating the biochemical values at each lattice point. We estimate the complexity of the simulation to be approximately $O(lw)$, where l,w are the length, width of the simulation lattice. There is also a slight dependence on *Dictyostelium* density since each amoeba requires added computation as compared to a amoeba-free lattice point.

5.7 Future refinements

There are some approximations we have made, regarding the cAMP reaction-diffusion mechanics and the granularity of the movement and time, which can be improved upon. Simulations on larger territories should also produce interesting results.

- We have not modeled the detailed reaction mechanics as proposed by Martiel and Goldbeter [MG87]. Instead, we have assumed the mechanics of the cAMP wave, and modeled the amoeba signalling so as to produce a cAMP wave in conformity with that experimentally observed by Tomchik and Devreotes [TD81].

- Modeling each amoeba by a single point provides a simple simulation, but the drawback is that the motion of each amoeba is extremely discrete. The smallest step it can move is its own diameter. Reducing the smallest step size may reduce the effect of the choice of topology. In fact, Mackay in his simulations found pronounced vertical and horizontal streaming if the movement step size was large [Mac78]. The hexagonal topology negates some of that effect.

- We are unable to conduct simulations on aggregation in territories larger by an order of magnitude then our current $1mm \times 1mm$ due to limitations of computer memory and time. Simulations on territories of size at least $10mm \times 10mm$ are needed to observe the spiraling cAMP wave patterns.

[9]There is approximately an order of magnitude improvement over interpreted CPL programs. In the current implementation, CPL programs are translated into C++ and then compiled into object code.

Chapter 6

Limb skeleton formation

> "The morphogenesis of the vertebrate limb, and its skeleton in particular, has long fascinated embryologists as a conspicuous and experimentally accessible example of the developmental process. The wide spectrum of adult limb forms found in different vertebrates, which nonetheless represent variations on a common structural theme, has stimulated a search for embryological mechanisms consistent with such degrees of freedom and constraints." [New88]

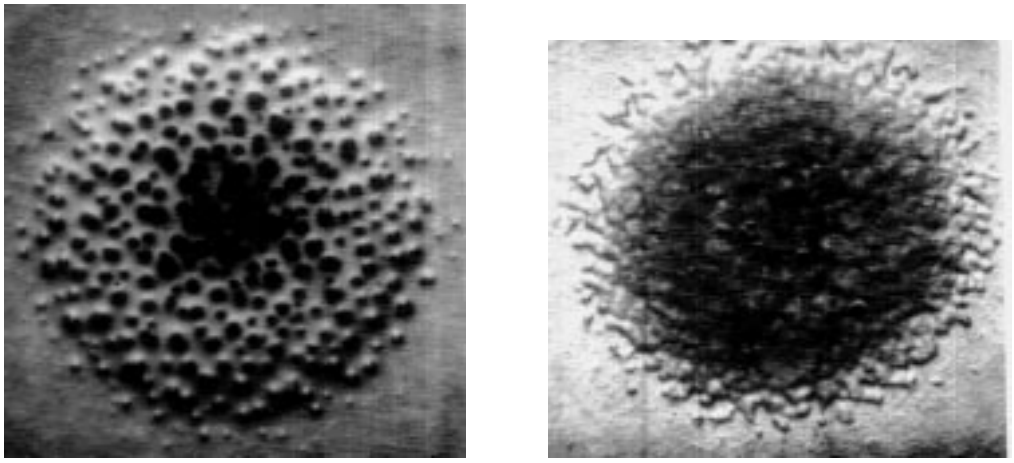
6.1 Theory

The limb bud is composed of mesenchymal cells encased by epithelial cells. Some of these mesenchymal cells form condensations (or clusters), and they differentiate to form chondrocytes. Chondrocytes are the precursors of cartilage. Bone eventually replaces cartilage by the process of ossification.

The formation of condensations is mediated by the protein fibronectin, which is present at the site of these condensations. Interference with fibronectin activity disrupts the condensations. The transforming growth factor-β (TGF-β) stimulates the production of both fibronectin and itself, and is non-uniformly distributed in precartilage tissue. There is evidence that TGF-β could be the primary morphogen that forms the standing waves for the reaction-diffusion model (Turing model, discussed in section 3.1.1) proposed by Newman et al. to account for the condensations [NF79,New88,LFF+91].

The formation of the vertebrate limb involves the formation of an increasing number of skeletal elements (cartilage/bone) from the base of the limb bud towards its tip[1] (i.e. along the proximo-distal axis). Wilby and Ede (1975) proposed a model which, given the proximo-distal pattern, simulates the anterior-posterior positioning of skeletal elements [WE75]. They were able to produce patterning resembling the layout of the bones in

[1] This is also referred to as the *in vivo* condensation pattern.



(a) Leg (b) Wing

Figure 6.1: The precartilage condensation patterns in leg and wing mesenchyme. Alcin blue-stained six day cultures are shown. Both tissues were isolated from stage 24 (4 1/2 day) chicken embryos. Culture diameter $\approx$ 5mm. Pictures courtesy of Downie and Newman [DN].

the limb (humerus; radius and ulna; metacarpals; digits). These patterns were produced by computer simulations using only local cell interactions, with one computer cell being equivalent to a hundred real cells. A single morphogen system was used, and a wave of this morphogen was set up in the system using simple rules, including a boundary layer of cells that actively destroys this morphogen.

Newman and Frisch mathematically analyzed a single morphogen model for the appearance of the precartilage elements [NF79]. They determined a system that would permit standing waves of a morphogen. The number of such waves along the anterior-posterior axis depends on the proximo-distal length of the system. A shorter proximo-distal axis would yield a larger number of standing waves. The proximo-distal length decreases from the humerous to the radius-ulna to the digits, and correspondingly the number of skeletal elements increases.

Newman et al. have studied the formation of precartilage (chondrogenic) condensations in chick limb bud mesenchyme *in vitro* [FJN89,LFF$^+$91]. The condensation patterns in the wing and leg mesenchyme are different [DN]. As can be observed from figure 6.1, the condensations in the leg are focussed, while the wing condensations are unable to stay focussed, and merge to form larger but weaker condensations.

6.2 *In vitro* model

In conjunction with Stuart Newman and J.K. Percus, we explored possible models that would explain the differences in the precartilage condensations. A simple activator-inhibitor model is proposed which accounts for these condensations.[2] The model was tested using CPL. The condensations are formed in regions of high fibronectin activity. Our model assumes the fibronectin concentration is linked to an activating biochemical (A) that diffuses (possibly TGF-β). In addition, a fast diffusing inhibitor (I) is present. The two interact according to the following equations:

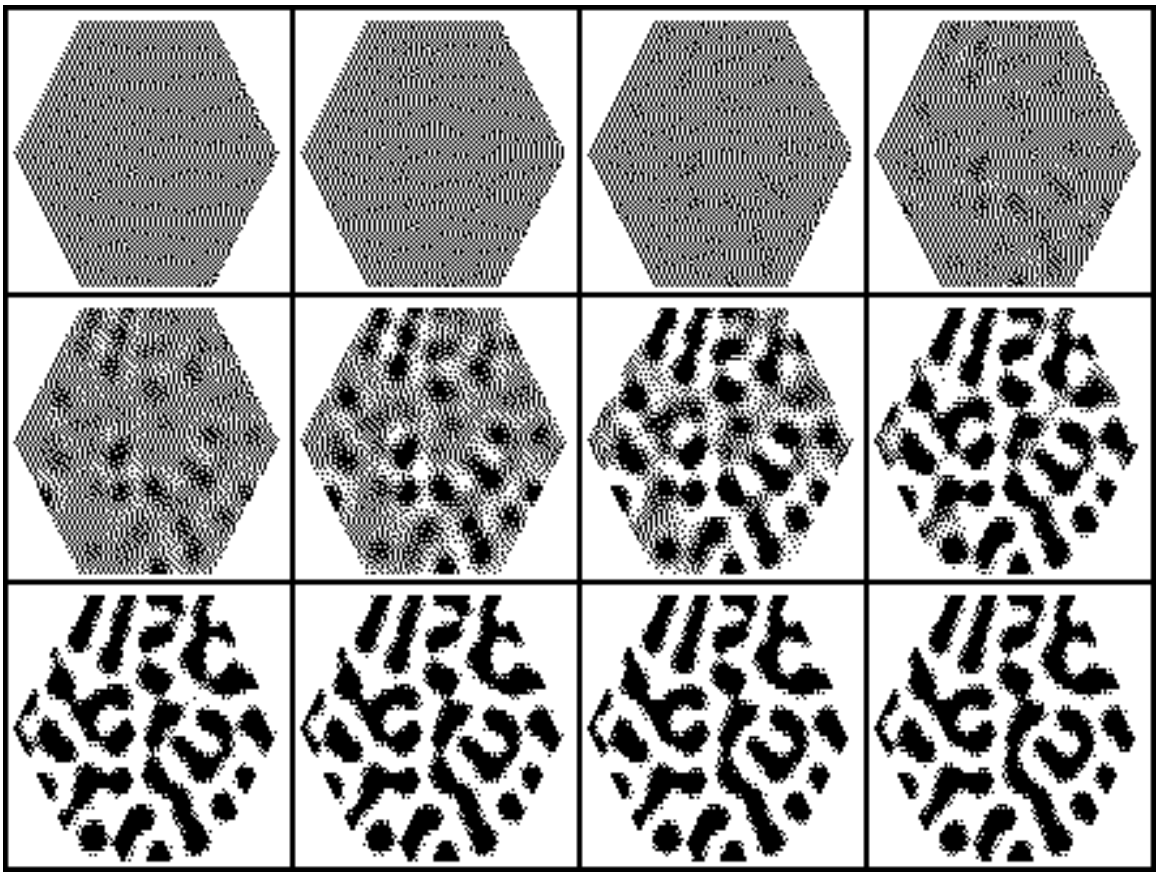
$$\frac{\delta A}{\delta t} = (0.1 + \frac{A}{9} - \frac{I}{5}) + \frac{1}{50}\sum_{n=1}^{6}(A_n - A)$$

$$\frac{\delta I}{\delta t} = Iresponse * (0.1 + \frac{A}{9} - \frac{I}{5}) + \frac{1}{7}\sum_{n=1}^{6}(I_n - I)$$

$\frac{1}{50}\sum_{n=1}^{6}(A_n - A)$ and $\frac{1}{7}\sum_{n=1}^{6}(I_n - I)$ are the activator and inhibitor diffusion terms from the six neighboring cells (we assume a hexagonal topology). It is a linear system with characteristics observed in the Turing system of section 3.1.1. Observe that I diffuses faster than A. The interaction of A and I provides zones of influence of A and I. The constant *Iresponse* determines how rapidly the inhibitor I responds to changes in the concentration of both A and I. If *Iresponse* is very low, then an increase in the activator level A spreads rapidly (the inhibitor being too slow to respond), and most of the region is influenced by A. However, if *Iresponse* is high, then an increase in the activator is quickly controlled (in area), since I is produced rapidly, and its quicker diffusion contains the region of influence of A. Simulations with different value of this parameter *Iresponse* yield patterns similar to those of the condensations in leg and wing mesenchyme. The variations in *Iresponse* are biologically plausible; the presence or absence of certain catalysts could presumably alter the speed of the reaction. Alternatively, *Iresponse* may be considered as a concentration of a distinct biochemical substance, and this concentration varies between the wing and the leg.

The simulation result for the activator and fibronectin concentrations leg tissue are shown in simulation images 6.1 and 6.3 ($Iresponse = 1.0$) and for the wing in simulation image 6.2 and 6.4 ($Iresponse = 0.01$) respectively. The fibronectin concentration profile mirrors the activator profile except for a smoother gradient. Cells that have an activator level higher than the equilibrium level manufacture fibronectin, and its production is assumed to deplete a resource. The simulation are on aggregates of about 50×50 cells. The activator and inhibitor concentrations in the cells are at equilibrium initially. There is no

[2] This model is only one *possible* explanation. However, the results of this model are encouraging in that the same model yields both the *in vivo* and *in vitroo* precartilage patterns in vertebrate limbs.

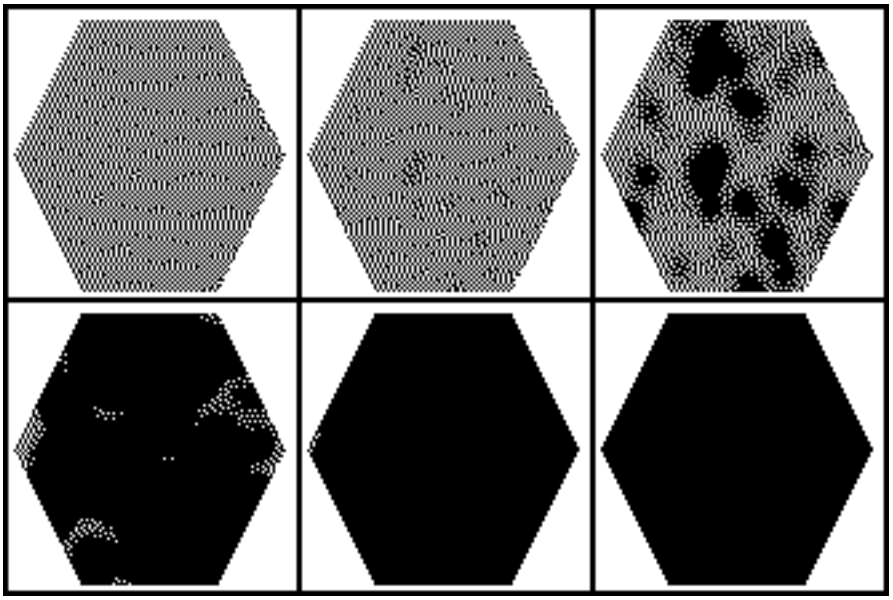


Simulation image 6.1: The *activator* concentrations for the *in vitro* condensation pattern in the chick *leg*. The subframes are at every 30 time steps starting from 0.

diffusion across the aggregate boundary; i.e. it is a closed system. Minor random positive perturbations in the activator concentrations of randomly chosen cells over the course of the simulation time result in variations in the activator-inhibitor values in the cells. Each perturbation results in unbounded maxima and minima. However, the simulation results improve if the concentrations are bounded; this is also more realistic, since concentrations cannot have infinite range, especially in biological systems.

Downie and Newman have also observed the effect of addition of TGF-β on the condensation patterns (see figure 6.2 on page 75). The effect of this additional TGF-β is to make the condensations almost uniform in both the leg and wing. The same effect is observed in the CPL simulations due to the addition of extra TGF-β (see simulation image 6.5).

The CPL program 6.1 provides details of the simulation parameters. The simulation running speed is comparable to that of the *Dictyostelium* simulations; however, the simulation has to run for only about 300 time steps as opposed to the 20,000 time steps for *Dictyostelium*.



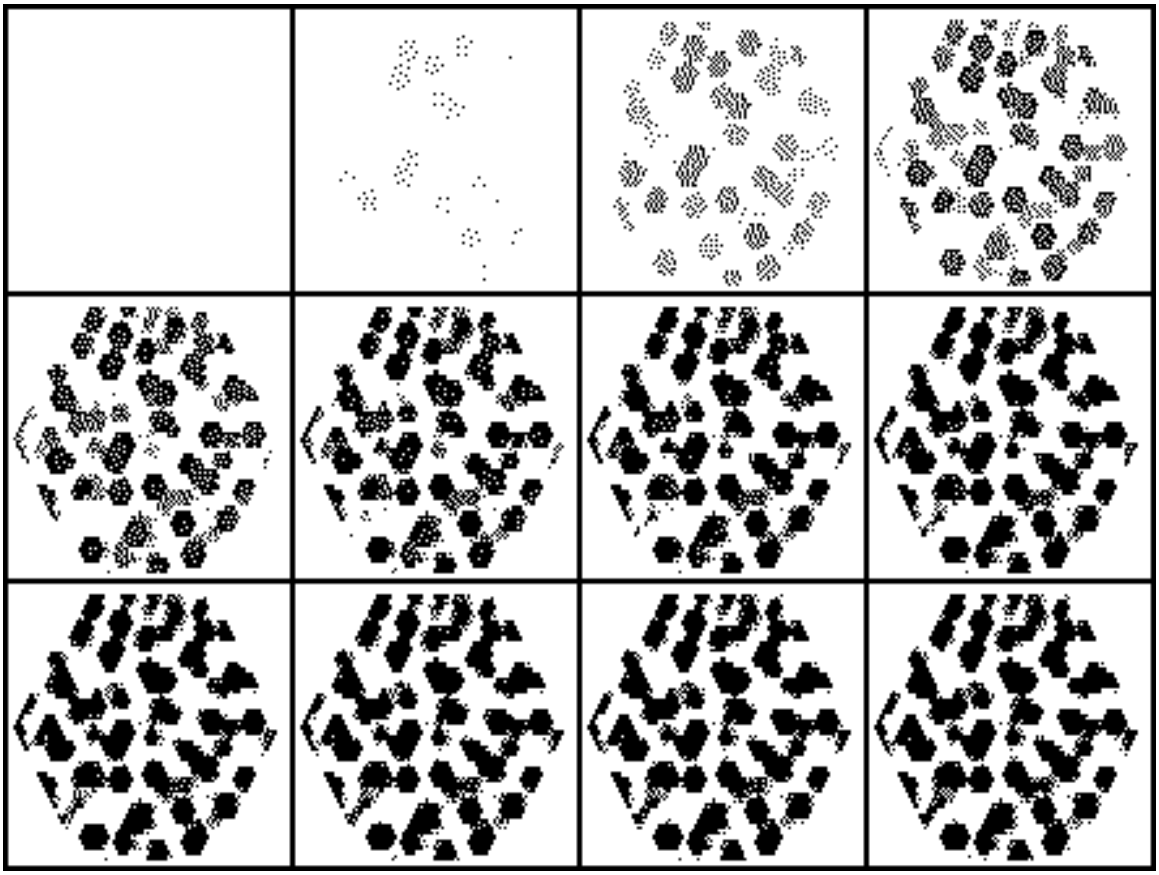
Simulation image 6.2: The *activator* concentrations for the *in vitro* condensation pattern in the chick *wing*. The subframes are at every 30 time steps starting from 0. The inhibitor has little influence, resulting in the activator being present throughout in high concentration.

6.3 *In vivo* model

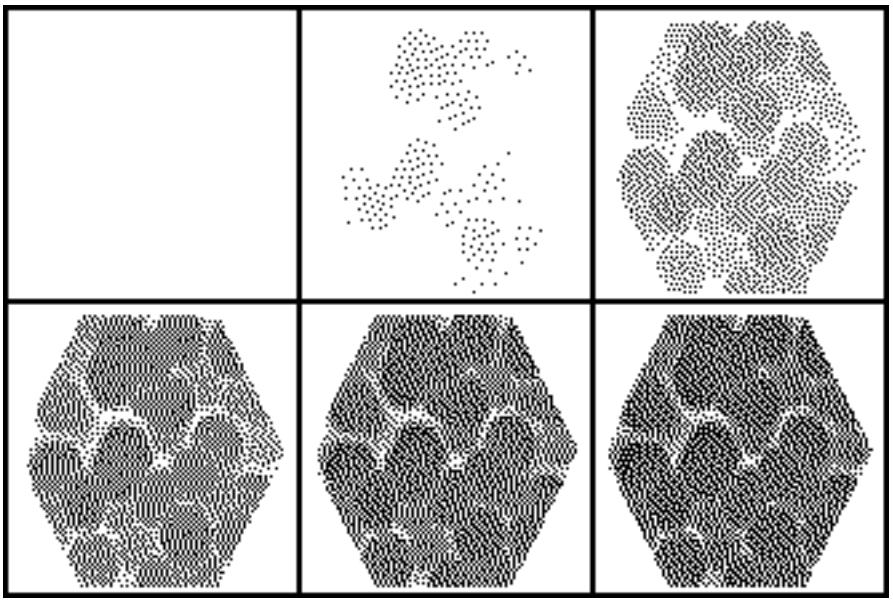
The choice of random cells with perturbations in activator values produces *in vitro* condensation patterns as previously shown. Remarkably, modifying the boundary to permit a slight leakage of the activator yields the *in vivo* condensation patterns, which is the pattern of the skeletal elements in the limb. This leakage could be explained by the ectoderm (epitheleal) cells surrounding the tissue as having slightly lower activator levels.

This leakage gives rise to banded patterns of activator concentration. The number of bands depends upon the width (anterior-posterior dimension) of the cellular aggregate, and the strength of response of the inhibitor ($Iresponse$). For a 50 cell wide aggregate $Iresponse = 0.01, 0.08, 0.5$ yields 1, 2, and 5 bands of precartilage respectively, which corresponds to the humerus, radius-ulna, and metacarpals, simulation image 6.6. Thus, if the width of the cellular aggregate is held constant, then increasing $Iresponse$ increases the number of waves, and thus increases the number of precartilage elements[3]. For constant $Iresponse$, increasing the width of the cellular aggregate increases the number of waves. Thus, the cellular aggregate widths and $Iresponse$ can be selected to produce the desired number of precartilage bands. The *talpid* mutant of the chick has a wider limb bud and not surprisingly a larger number of skeletal elements [NF79,WE75], which is predicted by

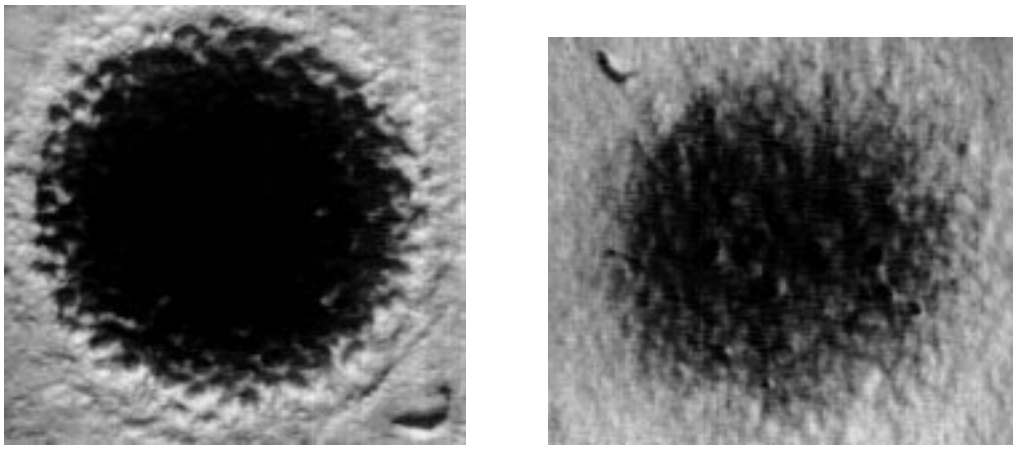
[3]If $Iresponse$ is modeling a biochemical substance, then the appearance of each successive precartilage element along the proximo-distal axis is marked by an increase in the concentration of $Iresponse$.



Simulation image 6.3: The *fibronectin* concentrations for the *in vitro* condensation pattern in the chick *leg*. The subframes are at every 30 time steps starting from 0. In contrast to the activator profile in simulation image 6.1, this has more clump-like formations.

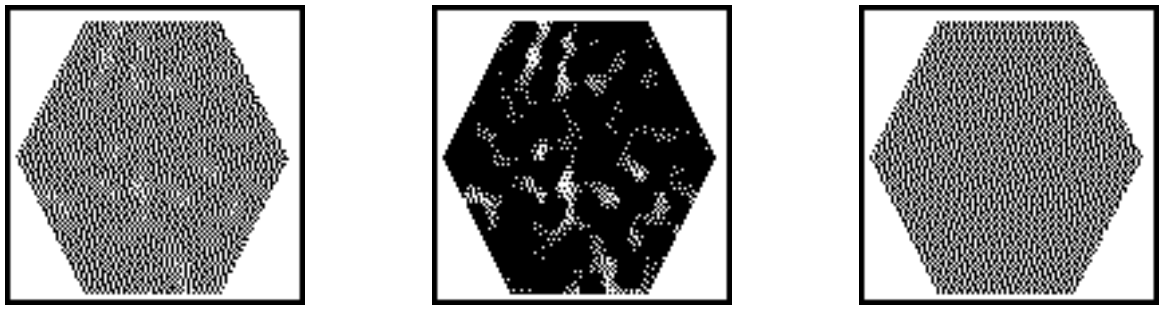


Simulation image 6.4: The *fibronectin* concentrations for the *in vitro* condensation pattern in the chick *wing*. The subframes are at every 30 time steps starting from 0. In contrast to the activator profile in simulation image 6.2, the overlapping clumps can be observed. This clumps in this simulation are also in contrast with the more focussed clumps in the leg case (simulation image 6.3).



(a) Leg (b) Wing

Figure 6.2: The precartilage condensation patterns in leg and wing mesenchyme on treatment with TGF-β (1 ng/ml TGF-β1 for 5 hours on day 1 after plating). Alcin blue-stained six day cultures are shown. Both tissues were isolated from stage 24 (4 1/2 day) chicken embryos. Culture diameter $\approx$ 5mm. Pictures courtesy of Downie and Newman [DN].



(a) (b) (c)

Simulation image 6.5: The effect of addition of TGF-β to the *in vitro* condensations in the chick leg and wing. In the leg mesenchyme, (a) almost uniform fibronectin concentration is observed at low resolution; (b) at high resolution some condensations can be distinguished. In the wing mesenchyme, (c) uniform fibronectin concentration is observed at all resolutions.

our model. The $Iresponse$ may also be adjusted to compensate for different widths of limb buds so as to produce the same number of skeletal elements. Assuming that $Iresponse$ is the concentration of a catalyst, and in a wider limb bud the catalyst is present in lower concentrations, then this could generate wider (but the same number of) skeletal bands.

In this model, the proximo-distal dimension has *no* effect on the number of skeletal elements formed.

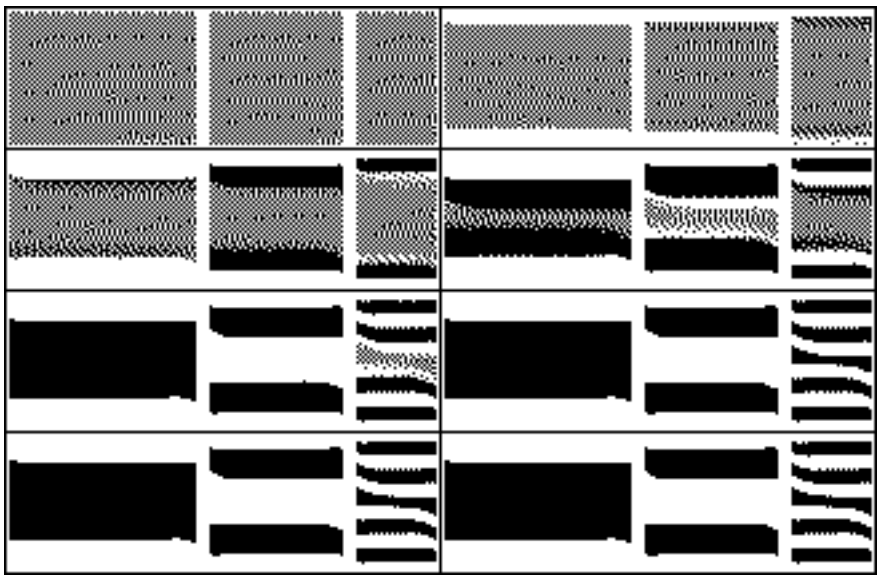
In the simulation image 6.6, the proximo-distal dimensions (length of the cellular aggregate) were arbitrarily selected to be 70, 50, and 30 cells (corresponding to the humerus, radius-ulna, and metacarpals). These 3 aggregates are isolated from each other (no diffusion across boundaries). This is biologically justifiable, since the precartilage forms along the proximo-distal axis at different times, and thus when the skeletal elements for the radius-ulna are being formed those for the humerus are already developed, while the metacarpals etc. have not yet been initiated.

Simulation image 6.7 presents the effects of varying the anterior-posterior dimension and $Iresponse$. Either thicker or more bands of precartilage may be obtained by varying the $Iresponse$ and increasing the anterior-posterior dimension.

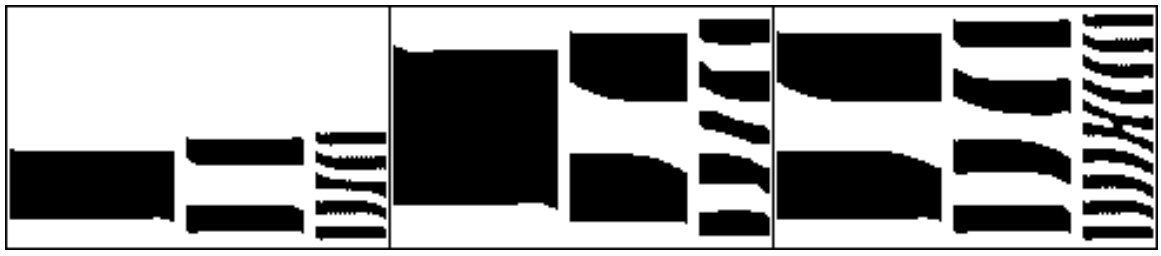
It is evident that introducing some fluctuation in the activator/inhibitor level is necessary to change the uniform equilibrium condition. We have examined the effects of random fluctuations (producing the *in vitro* pattern), and external leakage (producing the *in vivo* pattern). For completeness, the simulation image obtained by introducing both these phenomena together is exhibited. Simulation image 6.8 exhibits both external leakage of activator, and random fluctuations in activator concentration at all cells. As can be seen, the leakage dominates the exterior pattern (producing stripes), and the random fluctuations dominate the interior pattern (producing foci). Moreover, the size of the stripes and the

```
#define Iresponse 1.0
// 0.01 for wing, 1.00 for leg
#define TS 30
#define ConsumptionRate 40.0
#define BaseAct 0.9
#define BaseInh 1.0
#define Amplitude 0.8
float Act, Inh, resource, fibronectin;
biochemical Act, Inh, resource, fibronectin;

tissue mesenchyme{
  if (Act < BaseAct-Amplitude) Act = BaseAct-Amplitude;
  else if (Act > BaseAct+Amplitude) Act = BaseAct+Amplitude;
  deriv Act = (0.1+ Act/9.0-Inh/5.0)
              + (random(1,1000) == 1)*0.001*random(0,100);
              // produces the random positive perturbations
  deriv Inh = Iresponse*(0.1+ Act/9.0-Inh/5.0);
  for_each_neighbor_do
    // closed system, diffusion only to mesenchyme
    if (neighbor.tissue_type == mesenchyme){
      deriv Act = (neighbor.Act-Act)/50.0;//Inh diffuse faster than Act
      deriv Inh = (neighbor.Inh-Inh)/7.0;
      deriv resource = (neighbor.resource-resource)/40.0;
      // consumable resource required to make fibronectin
    }
  if (Act > BaseAct) {
    // resource is consumed to produce fibronectin
    deriv fibronectin = resource/ConsumptionRate;
    deriv resource = - resource/ConsumptionRate;
  }
}

tissue observer{
  if (int(time) mod TS == 0)
    image fibronectin;
}

cell{
  unit_area;
  type generic;
  start_up_area hexagon(26,26,25);
  Act = BaseAct;
  Inh = BaseInh;
  fibronectin=0.0;
  ConsumptionRate = 1.0;
}
```

CPL program 6.1: CPL program for the *in vitro* condensation pattern in the chick limb.

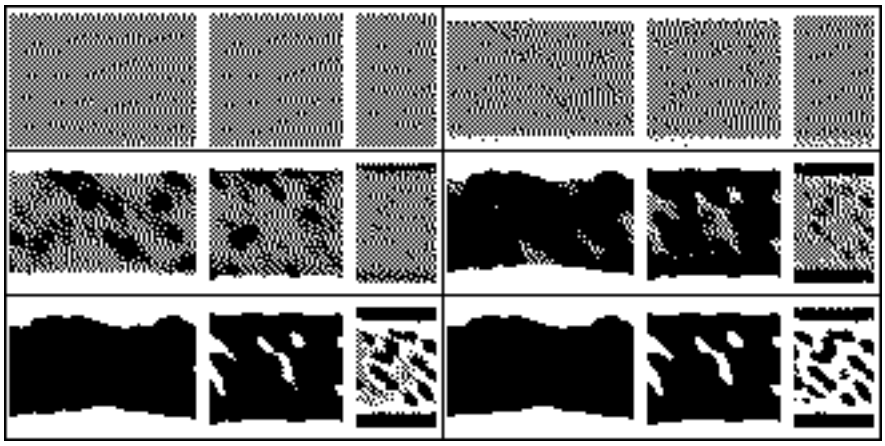


Simulation image 6.6: The activator concentrations for the *in vivo* precartilage pattern in vertebrate limb. The subframes are at every 50 time steps starting from 0. The proximo-distal axis is from left to right in each subimage. The anterior-posterior axis is 50 cells high in each subimage. The humerus, radius-ulna, and the metacarpals/digits appear as bands of 1,2, and 5 precartilage elements in aggregates with length 70, 50, and 30 cells along the proximo-distal axis respectively.



(a) (b) (c)

Simulation image 6.7: The activator concentrations for the *in vivo* precartilage pattern in a wider vertebrate limb. In (a), we have a 50 cell high image with a 1–2–5 skeletal pattern. In (b) and (c), the images have 100 cells along the height (anterior-posterior). However, in (b) it still produces the 1–2–5 pattern with thicker bands (smaller $Iresponse$), but in (c) it produces a 2–4–10 pattern with thinner bands (greater $Iresponse$). Other parameters are identical in the three subimages.



Simulation image 6.8: The activator concentrations in the case of both external leakage and random centers. The subframes are at every 30 time steps starting from 0. The proximo-distal axis is from left to right in each subimage. The anterior-posterior axis is 50 cells high in each subimage.

foci depend on the $Iresponse$. The $Iresponse$ values and other parameters are the same as in simulation image 6.6 (except for the addition of random activator fluctuations).

6.4 Commentary

What are the characteristics of this model that provide this range of developmental behavior? This is but one possible model. This model requires activation and inhibition resulting in the formation of stationary waves. For the *in vitro* case, the stationary wave centers are randomly distributed resulting in simple clumps; in the *in vivo* case, the centers align themselves, and the stationary wave fronts travel in unison resulting in a striped pattern. In addition, the model employs the fact that the inhibitor may react at different speeds to changes in its environment, which changes the wavelength of the stationary waves. In the leg mesenchyme, the inhibitor reacts quickly (smaller wavelength), and sit reacts slowly in the wing mesenchyme (larger wavelength). As long as the reaction-diffusion model displays these characteristics, it is possible to obtain the aforementioned developmental behavior.

One unsatisfactory aspect of this model is that the *in vitro* condensations require random *positive* perturbations in the activator level[4]. It may be difficult to organize a system such that only positive fluctuations in a biochemical concentration are permitted. However, a system that permits both positive and negative random perturbations in a biochemical concentration is easily achieved. The negative perturbations have the effect of creating disks of high inhibitor concentrations in which no clumps would form. This is not observed

[4]The same effect is obtained by permitting *negative* perturbations in the *inhibitor*.

in experiments; thus, permitting random negative perturbations in the activator provides unsatisfactory results.

Chapter 7

Segregation of tissues

7.1 Theory and simulations

Dissociated embryonic tissue reaggregates to resemble the original structure. Wilson in 1907 discovered that a dissociated sponge cells would reaggregate to form a functional sponge structure. Steinberg has extensively investigated the phenomena of reaggregation and sorting out of cells in embryonic tissue [Ste70] [Gil91, page 532]. The theory has been extensively discussed in sections 3.2, 3.2.1 and 3.2.3. To summarize, Steinberg observed that:

- A fragment of tissue rounds up into almost spherical shape.
- If two different cell types are mixed with each other, they segregate, with one of the cell types aggregating centrally and the other cell type surrounding it. For example, heart cells migrate towards the center when mixed with pigmented retinal cells, and pigmented retinal cells migrate towards the center when mixed with neural retinal cells.
- The central migration is transitive. If A sorts internally to B and B sorts internally to C, then A sorts internally to C. In fact, there is a complete hierarchy among cell types of such central migration.

Steinberg concluded that cells adhere to one another with varying strengths, and they try to maximize adhesive energy (or minimize free energy). This thermodynamic model was termed the "Differential Adhesion" model. In section 3.2.3, we have already discussed some of the simulations that have employed this model.[1]

[1] The volume by Mostow is a comprehensive collection of papers on cellular sorting and engulfment [Mos75].

We tested the differential adhesion hypothesis using CPL. The results we obtained are in accordance with the hypothesis. In fact, they are significantly better than the results obtained by Goel et al. [GR78]. The simulations by Goel et al. produced only local clumping. Other researchers have had great difficulty in producing anything more than local clumping. Their clumping results resemble those in simulation image 7.1. The improved clumping we observed was due to the addition of random motion (simulation images 7.2 and 7.3). Our experiments show that clumps of any size can be obtained provided random motion persists. Thus, clump size can be directly correlated to the duration of cell motility.

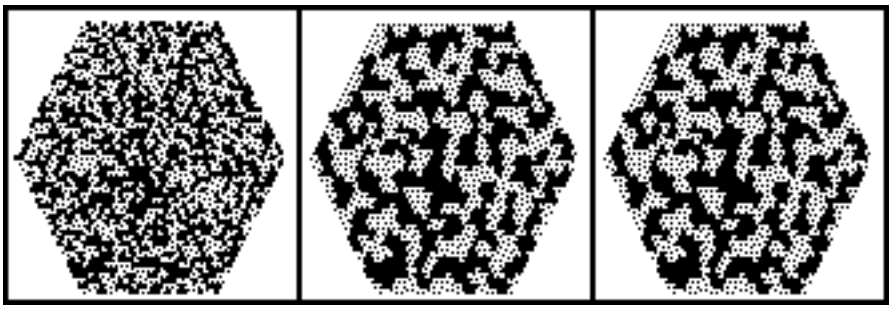
Engulfment is observed if two tissues of different adhesive strengths are placed besides each other. One of the tissues envelops the other. This is observed both in experiments and in the simulations we conducted (simulation images 7.4 and 7.5). Goel et al. were able to simulate engulfment but only by using a distant exchange mechanism. It is significant that the same effect can be observed by utilizing purely local mechanisms. No engulfment would be observed if there was no random motion, since the adjacent tissue are in a state of local adhesive energy maxima. The random motion eventually takes them to an even higher adhesive energy (through a path of lower adhesive energies).

The difficulty of getting the smaller clumps to combine and to form larger clumps increases exponentially with clump size. Experimental data on clump formation in laboratory experiments has been difficult to obtain. The size of the clumps obtained in the lab is limited, but the relationship of clump size with respect to time is not known to us. Possibly the slow motion of the cell clumps plays a role in their merging.

Simulation image 7.1 contains the tissue segregation obtained in the absence of any random motion. No cell motion is observed after 13 time steps, since each cell is at a local adhesive energy maxima. This is in contrast to the next few simulation images (7.2–7.5) in which the cells had random motion.

Simulation image 7.2 (page 84) represents segregation of two different tissue types. The more adhesive tissue migrates centrally, and the periphery is occupied by cells belonging to the less adhesive tissue. Simulation image 7.3 (page 85) represents the central movement that occurs when three different tissues are mixed together. The most adhesive tissue rounds up centrally; the cells of the tissue with the medium adhesivity form a shell around the most adhesive tissue; the periphery is occupied by cells belonging to the least adhesive tissue. Thus, an onion layered pattern is seen.

Simulation images 7.4 and 7.5 exhibit the engulfment of a tissue by a tissue of lower adhesive strength, when the two tissues are placed in contact. The shapes of the tissues are different in the two figures. However, engulfment occurs for both rectangular and hexagonal tissues in contact, illustrating that it does not depend upon initial shape.



Simulation image 7.1: Segregation of two tissues using adhesivity differences *without* any random motion. Each point represents a cell, and the two different tissue types are represented by light and dark points. The subframes are after simulation time steps 1, 10, and 20.

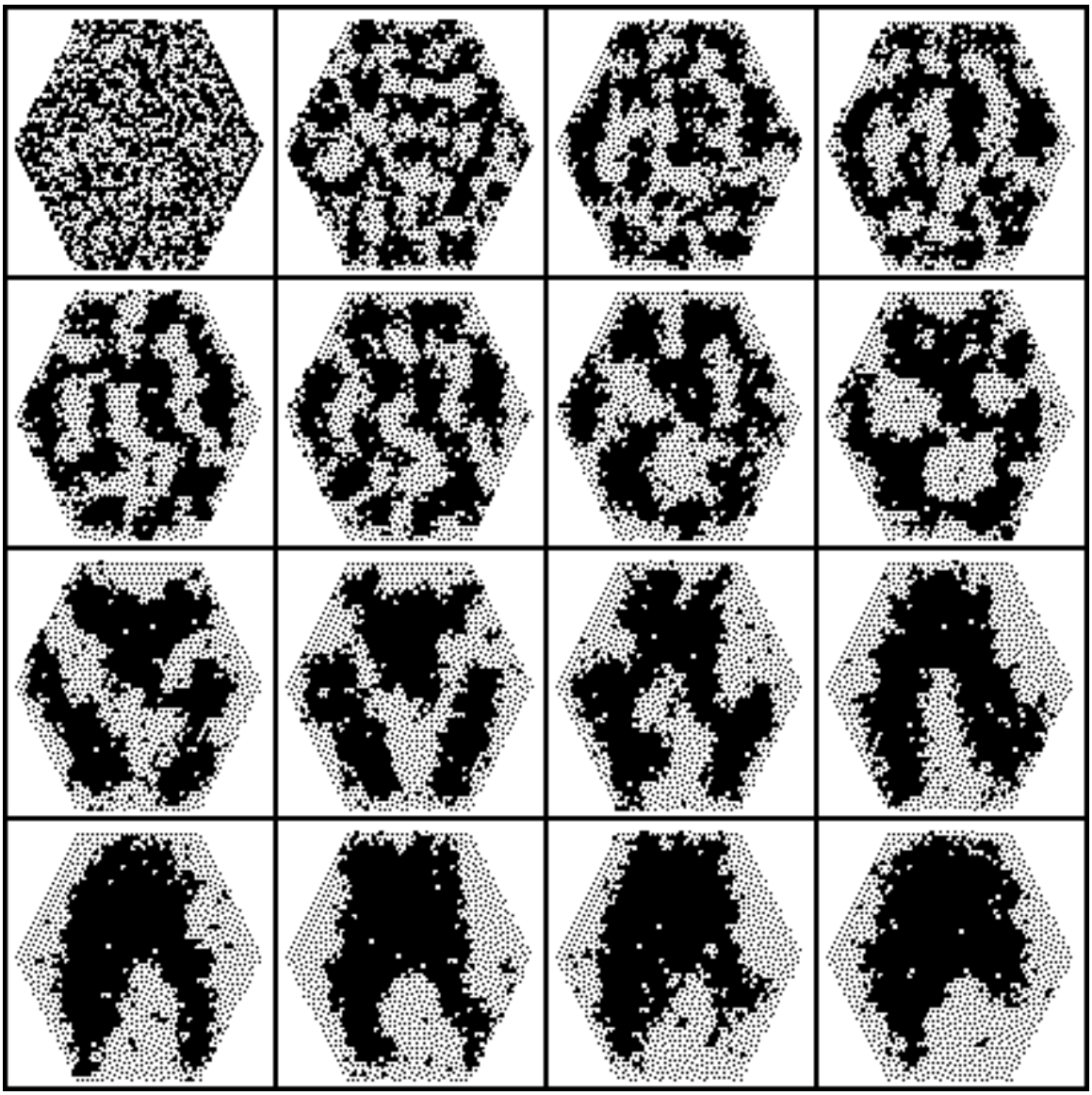
7.2 Program

CPL program 7.1 (page 88) was used to test the differential adhesion hypothesis. The program specifies a simulation starting with a hexagonal arrangement of cells in a hexagon of radius (side) 25, each cell of unit area. These cells, numbering about 2000, are of a *generic* tissue type, and randomly choose the number of contacts on their surface to be either 1 or 4. The adhesion between cells is proportional to the product of the contacts on their surface. Thus, cells with different contacts represent different tissue types.

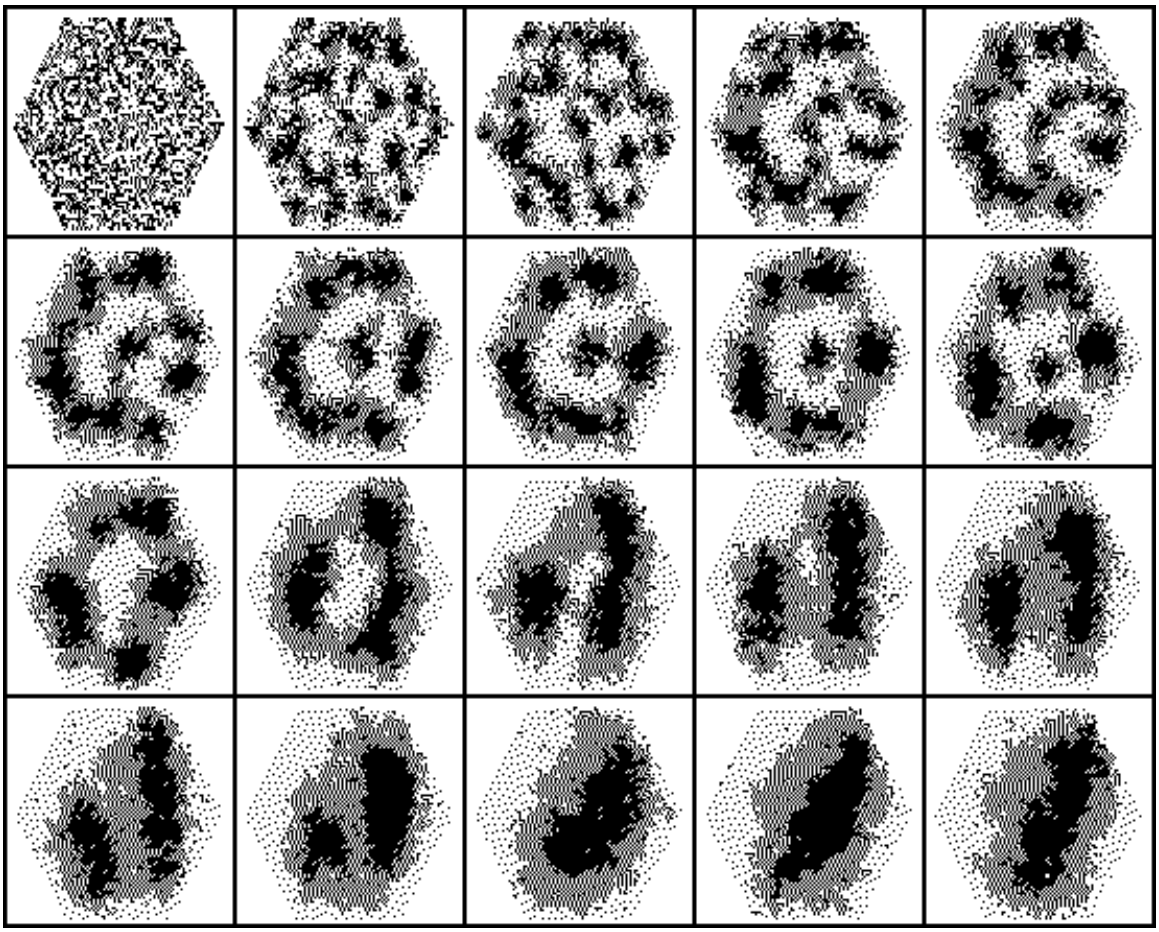
The first definition is that of a generic tissue, which uses a random number generator to split the tissue into cells with differing contact strengths. Each cell at every time step computes its own energy by summing up the number of contact sites on the cells surrounding it. The adhesive energy is given by the product of these contact sites and the number of the cell's own contact sites. The cell computes the move to a neighboring point that would result in the maximum increase in the adhesive energy, and with probability half makes that move. Cells also move in a random direction once every 4 time steps.

7.3 Simulation speed

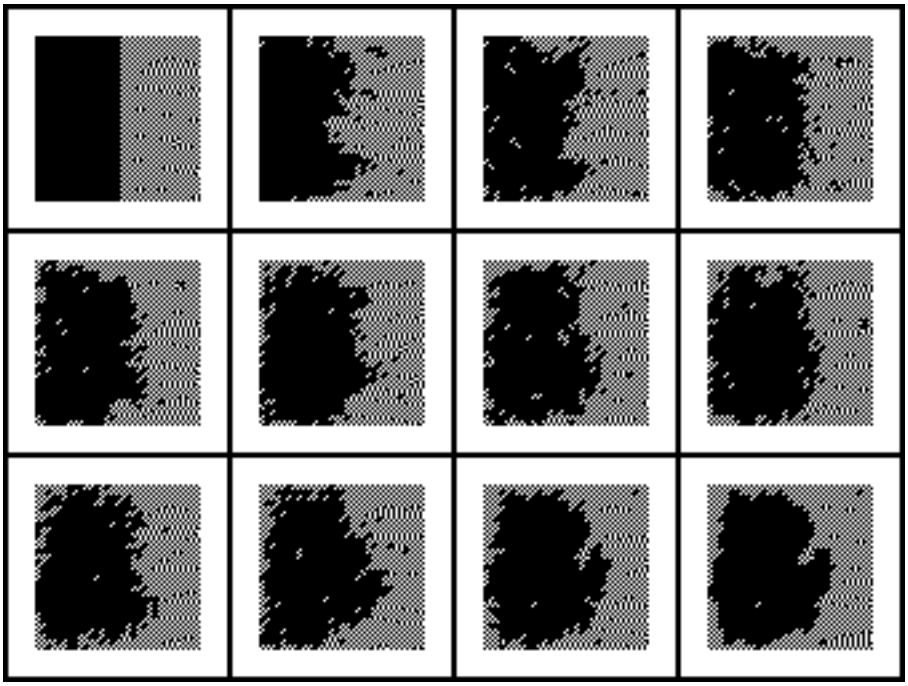
The cell segregation simulations have algorithmic complexity of approximately $O(lw)$ (identical to the *Dictyostelium* simulations). The running time for the cell segregation simulation for a lattice size of 100×100 for 100 time steps is about 240 seconds. The running time is greater compared to the *Dictyostelium* simulations due to increased cell motion. Cell motion requires recomputation of the neighborhood, which is expensive.



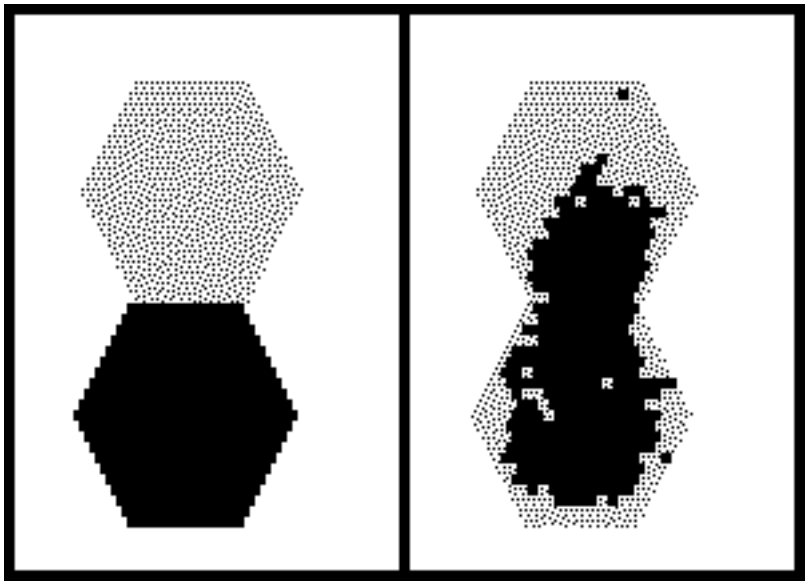
Simulation image 7.2: Segregation of two tissues using adhesivity differences. Each point represents a cell, and the two different tissue types are represented by light and dark points. The subframes are after 1, 20, 60, 120; 200, 300, 500, 800; 1300, 2000, 3000, 5000; 10,000, 15,000, 20,000 and 25,000 simulation time steps.



Simulation image 7.3: Segregation of tissues using adhesivity differences. The darkest points belong to the tissue with the highest adhesivity (they aggregate centrally), and the lightest ones have the weakest adhesivity (they aggregates externally). The subframes are after 1, 20, 60, 120, 200; 300, 500, 800, 1300, 2000; 3000, 6000, 9000, 12,000, 15,000; 18,000, 21,000, 24,000, 27,000 and 30,000 simulation time steps.



Simulation image 7.4: Engulfment of tissues placed in contact using adhesivity differences. The dark squares represent cells (496 in number) belonging to the tissue with the higher adhesivity, and the lighter ones (465 in number) have the weaker adhesivity. The subframes are after every 200 simulation time steps starting from time 0. In the last subframe (at 2200 time steps) the tissue is rounded up, since random motion is discontinued after time step 2000.



Simulation image 7.5: Engulfment of tissues placed in contact using adhesivity differences. The dark squares represent cells (331 in total) belonging to the tissue with the higher adhesivity, and the lighter ones (also 331) have the weaker adhesivity. The tissue with the weaker adhesivity envelops the more adhesive tissue.

```
#define ContactA 2
#define ContactB 1
simulation_size (53,53);
integer contactSum, contact, bestEnergy, exchangeEnergy;
direction dir;

tissue generic{
  state splitUp{
     if (random(0,1) == 0) contact = ContactB;
     else contact = ContactA;
     goto motion;
  }
  state motion{
    contactSum = 0;
    for_each_neighbor_do
      contactSum += neighbor.contact;
    bestEnergy =0;
    for_each_neighbor_do
      if (neighbor.contact != contact) {  //different tissue type
        exchangeEnergy =
            contactSum*neighbor.contact+neighbor.contactSum*contact -
            (contactSum*contact+neighbor.contactSum*neighbor.contact) -
            (contact - neighbor.contact)*(contact - neighbor.contact);
        if (exchangeEnergy > bestEnergy) {
          bestEnergy = exchangeEnergy;
          dir = neighbor.direction;
        }
      } // end for_each_neighbor_do
    if (bestEnergy > 0 and random(0,1) == 0) move dir;
       // half the time the best move is made
    if (random(0,3) == 0) move random_direction;
       // quarter of the time a random move is made
  }
}

tissue observer{
  image contact;
}

cell{
  unit_area;
  type generic;
  start_up_area hexagon(26,26,25);
}
```

CPL program 7.1: Segregation of tissues using adhesivity differences.

Chapter 8

Assorted applications

In the previous three chapters, we have discussed detailed models of different aspects of developmental behavior. In this chapter, we examine through simple models various simple developmental phenomena that may possibly be used as building blocks for complex behavior.

8.1 Unit area cells

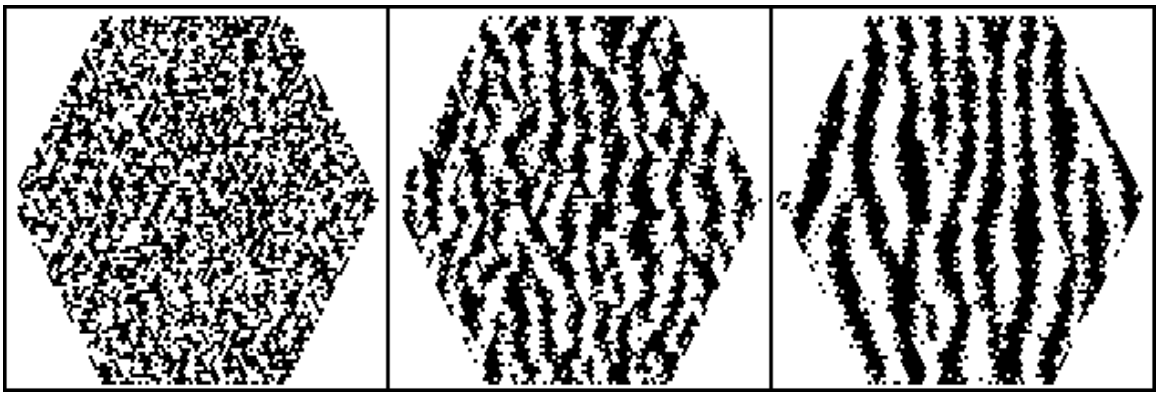
Each subsection displays a simulation with the patterns being formed mainly by the controlled motion of cells, each cell having unit area.

8.1.1 Stripe formation

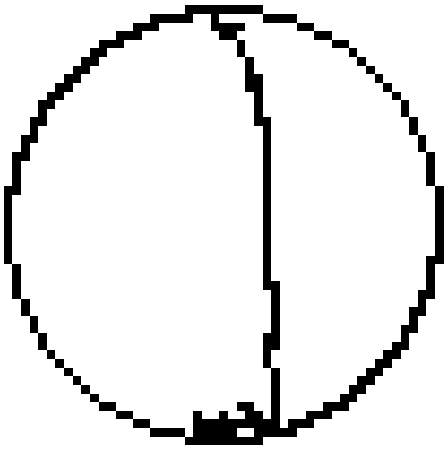
Utilizing a sorting mechanism, similar to that employed in chapter 7 but restricting cell motion to be parallel to one axis, one can obtain striped patterns (simulation image 8.1). Stripes are frequently seen developmental patterns, and this example illustrates one technique for obtaining them. Alternatively, they may be obtained by the interaction of diffusing biochemicals (Turing's stationary wave patterns).

8.1.2 Cell migration

The ability of cells to follow biochemical gradients is useful in pattern formation. The *Dictyostelium* cells exhibit an extreme sensitivity to cAMP gradients, and are able to move in direction of higher cAMP (discussed in chapter 5). In a smaller example, in simulation image 8.2, we exhibit the trail left by a single cell following a biochemical gradient due to its diffusion. There is a single source of the biochemical at the top of circular cell aggregate. Leakage of the biochemical by the peripheral cells acts as the sink. This establishes a biochemical gradient. The motile cell is initially present at the bottom of the circle, and



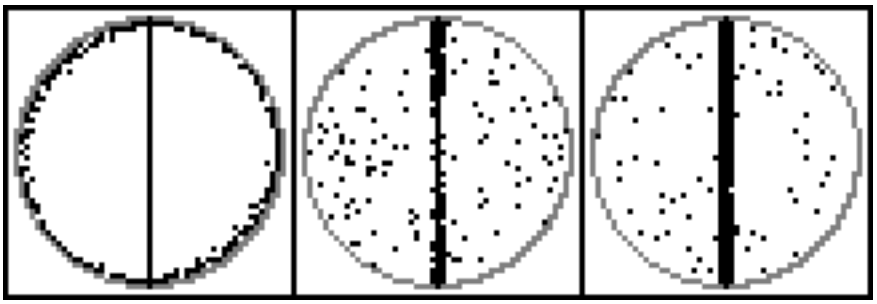
Simulation image 8.1: Striped pattern formation using lateral sorting.



Simulation image 8.2: Cell migration along a chemical gradient in a circular aggregate of cells.

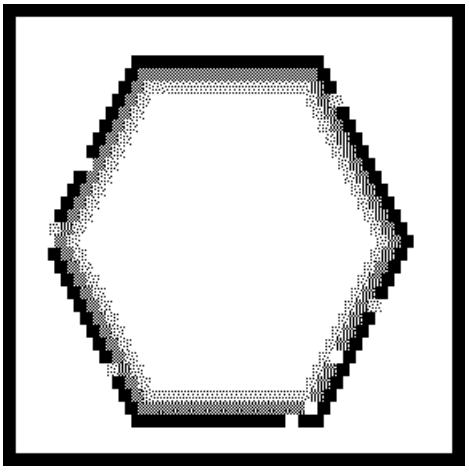
wanders randomly. On detecting the biochemical gradient it heads, more or less, in a straight line for the biochemical source.

There are numerous examples in developmental biology of cells moving around, apparently at random, until they happen to arrive at their destination, whence they loose their motility (for example, the migration of mesenchyme cells in the blastula). Simulation image 8.3 exhibits this phenomena with a central spine acting as the destination. One could envision the central spine to be an underlying tissue to which the wandering motile cells stick. In part (a) of the simulation image, the motile cells have not yet gained motility, and are stuck to the periphery. On gaining motility, by random wandering alone they are able to find the spine and stick to it.



(a) (b) (c)

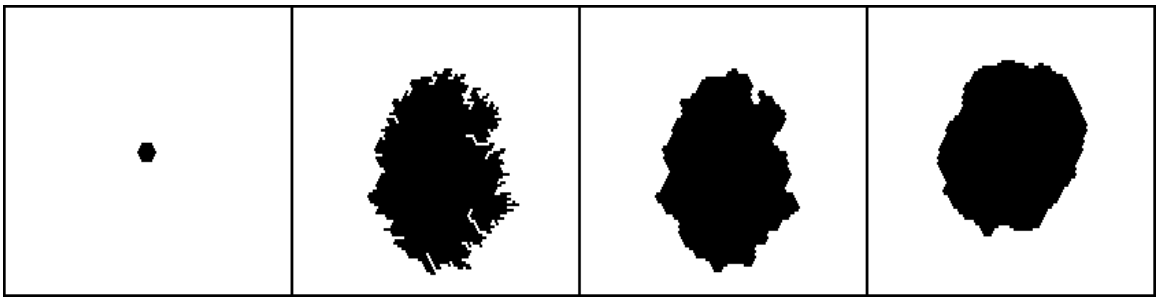
Simulation image 8.3: This assumes an underlying adhesive tissue along the central spine. The outer cells move randomly and stick on contact to the spine. (a) before the cells become mobile (b) some of the mobile cells have already found the central spine (c) the spine is covered by the mobile cells.



Simulation image 8.4: Three layers of tissues being formed using timing hypothesis of segregation.

8.1.3 Segregation using the timing hypothesis

This example demonstrates the formation of bands of tissues. Simulation image 8.4 exhibits an image of three layers of tissue. Initially, cells of all three kinds are randomly intermixed in the aggregate. We assume that the cells have a tendency to stick to the outer layer, and that different tissue types regain their adhesivities at different times. The tissue type that regained adhesivity first would form the outermost layer, and the tissue regaining adhesivity last would form the innermost layer. A similar explanation called *timing hypothesis of segregation* has also been used to explain cellular sorting. In chapter 7, we examined a more convincing explanation of segregation.



(a) (b) (c) (d)

Simulation image 8.5: Cell growth from 37 points to 2537 points by adding points in random directions. (a) Initial 37 point cell (b) After random growth by 2500 points (c) After rounding up the boundary from previous simulation image (d) After rounding up the boundary at each stage of the growth process (50 stages).

8.2 Multiple area cells

It is not a coincidence that the previous three chapters examined models that all involved cells of the same size and shape. In this section, we will grow and divide cells.

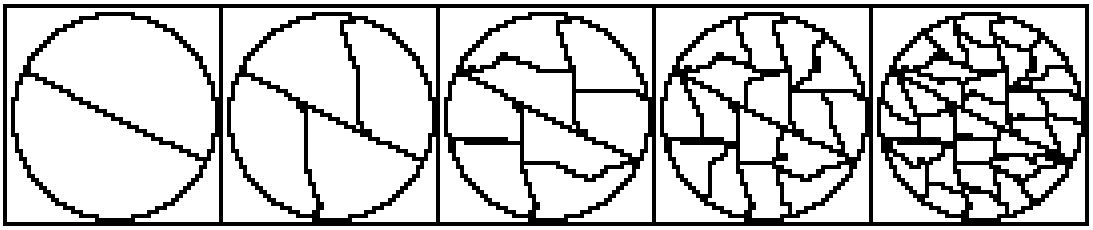
8.2.1 Cell growth

Simulation image 8.5 contains images of cell growth. Originally the cell is small (area = 37). The cell grows by stealing points from its neighbors, which in this case is the environment. The second image reveals a rather jagged cell. Since, in nature, cell growth does not lead to such angular features, it is evident that a rounding up of the cell is needed to ensure cohesiveness. Simulation image 8.5(c) exhibits the effect of rounding up this jagged cell. Rounding up is effective in removing the jagged edges, but the cell still has a jagged skeleton. As simulation image 8.5(d) shows, the best results are obtained by rounding up the cell at each stage of the growth process. This cell was grown by adding 50 points at each time step in randomly chosen directions.

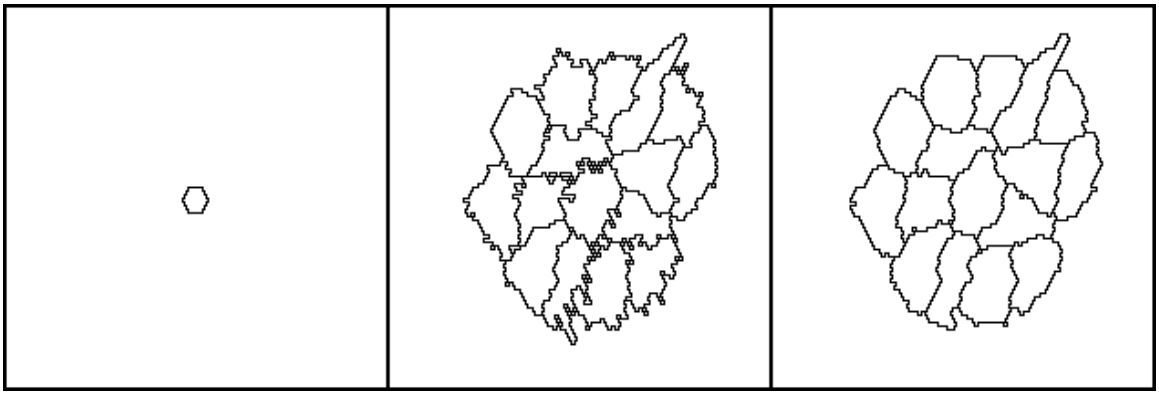
8.2.2 Cell division

Simulation image 8.6 demonstrates the effect of repeated cell division on a reasonably large cell. Successive divisions are made at right angles to the previous one. Cell division lines have to be chosen with care, so as to divide the cell into daughter cells of equal area, while maintaining the connectivity of cells. The effect of the choice of hexagonal topology and the division algorithm is evident in the division lines[1].

[1] Refer to the implementation note on the division algorithm in section 4.8.5.



Simulation image 8.6: Repeated cell division.



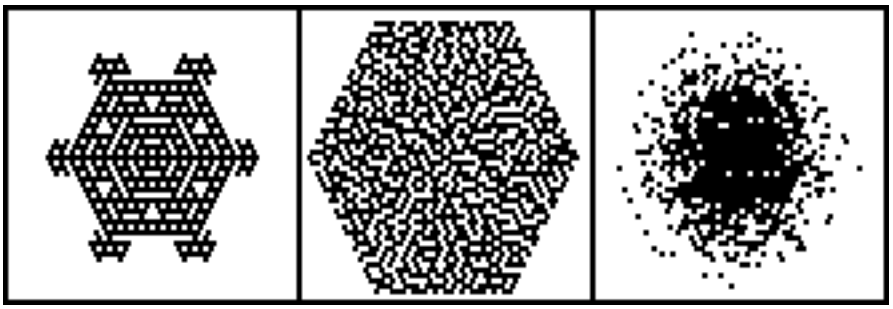
(a) (b) (c)

Simulation image 8.7: Tissue grows by about 3500 points. The cells also divide upon reaching a size of 400 points. (a) Initial 37 point cell (b) After growth and division (c) After a few stages of rounding up the cells.

8.2.3 Cell growth and division example

Simulation image 8.7 combines the cell growth and division of the previous two examples. The simulation commences with a cell with an area of 37 points (as in the growth example). The cell increases in area by 50 points at every time step, and rounds up after every growing step. The cells divide if their area surpasses 400 points. Successive divisions are again perpendicular to the previous one. The rounding up is seen to have a positive effect on cell shape in simulation image 8.7 (c).

These growth and division examples reveal interesting patterns but lack serious biological significance. Growth and division are intrinsically three dimensional activities. We would like to explore examples of development where two dimensional growth and/or division play significant roles in pattern formation.



(a) (b) (c)

Simulation image 8.8: Cellular automaton based (a) fractal (b) dendrite formation by controlled growth (c) diffusing gas

8.3 Cellular automaton

Simulation image 8.8 includes traditional cellular automaton simulations. Similar ones may be found in Toffoli's book on Cellular Automata [Tof87]. Simulation image 8.8(a) exhibits an image with a fractal flavor. (b) exhibits dendrite formation by allowing only cells that have exactly one neighboring cell of dendritic form, to differentiate to form dendrites. (c) exhibits a diffusing gas. Each cell is a gas molecule, and random motion causes it to diffuse. Thus, although CPL has a distinct developmental biology flavor, it can be employed for other cellular automaton applications. However, CPL programs, being versatile, compromise on speed compared to other cellular automaton programs: *cellular* [Eck91] and *SLANG* [SC91].

Chapter 9

Conclusions

The aim of this thesis was to develop a simple language (a means of description) for a variety of developmental behavior. This language took the shape of a programming language (CPL). The various models we developed emphasized the scope of CPL. The modeling of *Dictyostelium* aggregation involved chemotactic movement due to biochemical diffusion. The formation of skeletal elements in limb was explained by a Turing reaction-diffusion model. The segregation and engulfment of tissues involved minimization of free energy and dealt with mechanical forces in the form of contact, adhesion, and motion.

The examination of the instruction sequences for the applications we have developed has also proved useful.

- The CPL programs serve as succinct descriptions of the cell's activities.
- Comparison between the instruction sequences of distinct cells has highlighted both the similarity and the differences between them.

Running simulations on aggregates of these cells has yielded developmental behavior. We have been able to encode various hypothesis regarding developmental behavior in terms of these instruction sequences, and the simulation results have at times reinforced or weakened the hypothesis. Some simulations have suggested possible experiments to confirm hypotheses. For example, an experiment on the time required to double clump size due to cell segregation is suggested.

CPL is similar in some respects to Fleischer and Barr's work on designing a system to study multicellular pattern formation [FB93]. Both CPL and Fleischer's work are concerned with observing pattern formation in multicellular simulations. However, the goals are different: Fleischer and Barr want to create artificial genomes to simulate evolution, while CPL's intention is to shed light on natural processes by making it easy to model different possible explanations. Fleischer and Barr employ a continuous model with differential equations, whereas CPL utilizes a discrete model with difference equations. CPL can

model a larger number of cells, but the difference equations involved make our calculations more approximate than the continuous model of Fleischer and Barr.

Computational power

CPL takes an approach akin to the cellular automata approach; however, the cells are infused with more power. In cellular automata, cells compute simple next state functions based on their own and their neighbors current states. Cellular automata arrays can represent logical circuits, and thus they are Turing computable (since circuits are Turing computable). In CPL, each cell is Turing computable, and we believe this adds to its descriptive power. Most cellular automatons use a cryptic language with a compact and efficient syntax. Run-time computational efficiency is their primary concern. CPL is designed with a specific objective — modeling in developmental biology — and can afford to be more descriptive. In CPL, we write programs for cells and not equations for grid points. CPL is a high-level language as opposed to the assembly language appearance of most cellular automaton models.

Open problems

Major problems in the realm of growth, division, and motion of arbitrary sized cells are still unresolved. The solutions CPL adopts for these problems are unsatisfactory. CPL has drawn inspiration from biology for the design of its instruction set, but simple biological models of growth, division, and motion are not available. Motion can be managed if all the cells are similar in shape and size, but in other cases our solution is somewhat ad hoc.

Most development is essentially three-dimensional. It is evident that the two-dimensional models are in most cases approximations to reality. In the *Dictyostelium*, the conical mound that forms is three-dimensional and removes cells from the two-dimensional aggregation plane. This provides extra space for the *Dictyostelium* streams to converge. Cellular segregation is a three-dimensional process, and critics have even questioned the thermodynamic validity of the two-dimensional model. The extension of CPL to three-dimensions is straightforward (essentially the neighborhood function has to be modified). However, the extra computational memory and time such simulations require raise questions about the choice of discrete representation of cells. Spheres and /or ellipsoids of varying radii, or polyhedrons with a limited number of surfaces, are possibilities for cell representations in three dimensions.

The current implementation of CPL is a sequential one, but cells inherently work in parallel. The behavior of cells would be further illuminated if we adopted a parallel implementation that mimics their natural behavior. Cells only have local information, and they utilize it to decide their next state. If they move or grow, this effect is felt by their

immediate neighbors, and through them by the rest of the aggregate. By utilizing such local schemes, the entire cell aggregate is able to maintain energetically favorable shapes. We still need better understandings of exactly how local behavior determines global shape and size. That knowledge would enable us to assign processes to each cell which work semi-synchronously, and would make a good parallel implementation.

In addition to the problems mentioned above, each of the models we have discussed raises some questions and has possibilities for refinements. These have been discussed in the appropriate chapters.

Appendix

BNF Grammar for CPL

The syntax is described in extended Backus-Naur Form (BNF) [Mac87, page 159–169]. Only terminals are in boldface. The following symbols are meta-symbols belonging to the BNF formalism, and are not symbols of the language.

::= | [ ] * +

program	::=	[hash_defines]* simSize [declarations]* [tissue]+ [cell]+
hash_defines	::=	**#define** identifier value
simSize	::=	**simulation_size (** integer, integer **);**
		[**time_interval =** real;]
declarations	::=	[**biochemical** | **integer** | **float** | **direction** |
		global | **static** identifier [**,** identifier]+ **;**]+
tissue	::=	**tissue** tissue_name { state+ } | bracedCommands
state	::=	**state** state_name [**:like** statename] bracedCommands
commands	::=	bracedCommands | command **;**
bracedCommands	::=	{ [command **;**]+ }
command	::=	assignment | if_then | go_to_state | differentiate_to |
		ForNeighbor | WithNeighbor | **exit** | **die** |
		grow | divide | move | **roundup**|
		image | write | echo
assignment	::=	variable = expression |
		biochem_name = expression |
		deriv biochem_name = expression |
		variable += expression |
		variable −= expression

if_then	::=	**if** expression **then** commands |
		if expression **then** commands **else** commands
go_to_state	::=	**goto** state_name
differentiate_to	::=	**differentiate_to** tissue_name
ForNeighbor	::=	**for_each_neighbor_do** commands
WithNeighbor	::=	**with_neighbor_in_direction** direction commands
grow	::=	**grow** size direction
divide	::=	**divide** divide_option
divide_option	::=	**horizontal** | **vertical** | **perpendicular** | **any**
move	::=	**move** direction
image	::=	**image** [**tissue** | **state** | **cell_number** |
		biochem_name | variable]
write	::=	**write** expression
echo	::=	**echo** "string"
tissue_name	::=	identifier | **environment** | **observer**
state_name	::=	identifier
variable	::=	identifier
biochem_name	::=	identifier
size	::=	expression
distance	::=	expression
direction	::=	identifier | **random_direction**
expression	::=	(expression) |
		+ expression |
		− expression |
		expression + expression |
		expression − expression |
		expression ^ expression |
		expression * expression |
		expression / expression |
		expression == expression |
		expression != expression |
		expression <= expression |
		expression >= expression |
		expression < expression |
		expression > expression |
		expression **and** expression |
		expression **or** expression |

not expression |
sqrt(expression **)** |
point(integer, integer**)** |
time |
steps |
time_interval |
area |
perimeter |
cell_number |
location |
random(integer, integer**)** |
neighbor.contact_length |
neighbor.area |
neighbor.perimeter |
neighbor.direction |
neighbor.variable |
neighbor.biochem_name |
neighbor.tissue_type == tissue_name |
neighbor.tissue_type != tissue_name |
neighbor.state == state_name |
neighbor.state != state_name |
integer |
real |
variable

cell ::= **cell** { initialization }

initialization ::= [**unit_area;**]
type tissue_name **;**
start_up_area cell_location **;** variable_init

cell_location ::= object | cell_location **union** object

object ::= **rectangle(** integer, integer, integer, integer**)** |
circle(integer, integer, integer**)** |
hexagon(integer, integer, integer**)** |
triangle(integer, integer, integer, integer,
integer, integer**)**

variable_init ::= variable_init variable **=** expression **;** | [ ]

Bibliography

[Aga93] Pankaj Agarwal. The Cell Programming Language. Technical Report 630, Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, 1993.

[AM74] F. Alacantra and M. Monk. Signal propagation during aggregation in the slime mold *Dictyostelium Discoideum*. *Journal of General Microbiology*, 85:321–334, 1974.

[ARW75] P. Antonelli, T.D. Rogers, and M. Willard. Cell aggregation kinetics. In G.D. Mostow, editor, *Mathematical Models for Cell Rearrangement*, chapter 10, pages 176–195. Yale University Press, New Haven, 1975.

[Bon67] J.T. Bonner. *The Cellular Slime Molds*. Princeton University Press, Princeton, NJ, 1967.

[CM88] Richard Cowan and Valerie B. Morris. Division rules for polygonal cells. *Journal Of Theoretical Biology*, 131:33–42, 1988.

[CP78] Stephen Childress and Jerome K. Percus. *Mathematical Models in Developmental Biology*. Courant Institute of Mathematical Sciences Lecture Notes. New York Univeristy, 1978.

[CR71] M.H. Cohen and A. Robertson. Wave propagation in the early stages of aggregation in cellular slime molds. *Journal Of Theoretical Biology*, 31:101–118, 1971.

[DN] Sherry A. Downie and Stuart A. Newman. Morphogenetic differences between fore and hind limb precartilage Mesenchyme: Relation to mechanisms of skeletal pattern formation. Submitted for Publication: June 1993.

[Eck91] J Dana Eckart. A *cellular* automata simulation system. *SIGPLAN Notices*, 26:8:80–85, 1991.

[FB93] Kurt Fleischer and Alan H. Barr. A simulation testbed for the study of multicellular development: The multiple mechanisms of morphogenesis. In *Artificial Life III*. Addison-Wesley, 1993.

[FJN89] Dorothy A. Frenz, Navdeep S. Jaikaria, and Stuart A. Newman. The mechanism of precartilage Mesenchymal condensation: A major role for interaction of the cell surface with the Amino-terminal Heparin-binding domain of Fibronectin. *Developmental Biology*, 136:97–103, 1989.

[GCG+75] Narendra Goel, Richard D. Campbell, Richard Gordon, Robert Rosen, Hugo Martinez, and Martynas Yčas. Self-sorting of isotropic cells. In G.D. Mostow, editor, *Mathematical Models for Cell Rearrangement*, chapter 7, pages 100–144. Yale University Press, 1975.

[Gil91] Scott F. Gilbert. *Developmental Biology*. Sinauer Associates, Sunderland, MA, 1991.

[GL75] Narendra S. Goel and Ardean G. Leith. Self-sorting of anisotropic cells. In G.D. Mostow, editor, *Mathematical Models for Cell Rearrangement*, chapter 8, pages 145–158. Yale University Press, 1975.

[Gor66] Richard Gordon. On stochastic growth and form. *Proceedings of the National Academy of Sciences*, 56:1497–1504, 1966.

[GR78] Narendra S. Goel and Gary Rogers. Computer simulation of engulfment and other movements of embryonic tissues. *Journal Of Theoretical Biology*, 71:103–140, 1978.

[Har76] Albert K. Harris. Is cell sorting caused by differences in the work of intercellular adhesion? A critique of the Steinberg hypothesis. *Journal Of Theoretical Biology*, 61:267–285, 1976.

[Hon78] Hisao Honda. Description of cellular pattern by Dirichlet domains: The two-dimensional case. *Journal Of Theoretical Biology*, 72:523–543, 1978.

[LFF+91] Claire M. Leonard, Howard M. Fuld, Dorothy A. Frenz, Sherry A. Downie, Joan Massagué, and Stuart A. Newman. Role of Transforming Growth Factor-β in chondrogenic pattern formation in the embryonic limb: Stimulation of Mesenchymal condensation and Fibronectin gene expression by exogenenous tgf-β and evidence for endogenous tgf-β-like activity. *Developmental Biology*, 145:99–109, 1991.

[LG71] Ardean G. Leith and Narendra S. Goel. Simulation of movement of cells during self-sorting. *Journal Of Theoretical Biology*, 33:171–188, 1971.

[LG75] Ardean G. Leith and Narendra S. Goel. Simulation of movement of cells during self-sorting. In *Mathematical Models for Cell Rearrangement*, chapter 9, pages 159–175. Yale University Press, 1975.

[Lin68] Aristid Lindenmayer. Mathematical models for cellular interaction in development, I & II. *Journal Of Theoretical Biology*, 18:280–315, 1968.

[Mac78] S.A. MacKay. Computer simulation of aggregation in *Dictyostelium Discoideum*. *Journal Of Cell Science*, 33:1–16, 1978.

[Mac87] Bruce J. Maclennan. *Principles of Programming Languages : Design, Evaluation, and Implementation*. Holt, Rhinehart and Winston, 1987.

[Mei82] Hans Meinhardt. *Models of Biological Pattern Formation*. Academic Press, New York, 1982.

[MF80] Raymond J. Matela and Robert J. Fletterick. Computer simulation of cellular self-sorting: A topological exchange model. *Journal Of Theoretical Biology*, 84:673–690, 1980.

[MG87] J.-L. Martiel and A. Goldbeter. A model based on receptor densensitization for cAMP signalling in *Dictyostelium* cell. *Biophysical journal*, 52:807–828, 1987.

[Mos75] G.D. Mostow. *Mathematical Models for Cell Rearrangement*. Yale University Press, 1975.

[MR85] Raymond J. Matela and Robert Ransom. A topological model of cell division: Structure of the computer program. *BioSystems*, 18:65–78, 1985.

[MSR91] Eric Mjolsness, David H. Sharp, and John Reinitz. A connectionist model of development. *Journal Of Theoretical Biology*, 152:429–453, 1991.

[New77] P.C. Newell. Aggregation and cell surface receptors in cellular slime molds. In J.L. Reissig, editor, *Receptors and Recognition: Series B Volume 3: Microbial Interactions*, chapter 1, pages 1–58. John Wiley and Sons, New York, 1977.

[New88] Stuart A. Newman. Lineage and pattern in the developing vertebrate limb. *Trends in Genetics*, 4:329–332, 1988.

[NF79] Stewart A. Newman and H. L. Frisch. Dynamics of skeletal pattern formation in developing chick limb. *Science*, 205:662–668, 1979.

[NS76] B. Novak and F.F. Seelig. Phase-shift model for the aggregation of amoeba: A computer study. *Journal Of Theoretical Biology*, 56:301–327, 1976.

[OOAB81] G. Odell, G. Oster, P. Alberch, and B. Burnside. The mechanical basis of morphogenesis. I. Epithelial folding and invagination. *Developmental Biology*, 85:446–462, 1981.

[OS91] Q. Ouyang and Harry L. Swinney. Transition from a uniform state to hexagonal and striped Turing patterns. *Nature*, 352:610–612, 1991.

[PL90] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer Verlag, 1990.

[PS77] H. Parnas and L.A. Segel. Computer evidence concerning the chemotactic signal in *Dictyostelium Discoideum*. *Journal Of Cell Science*, 25:191–204, 1977.

[PS78] H. Parnas and L.A. Segel. A computer simulation of pulsatile aggregation in *Dictyostelium Discoideum*. *Journal Of Theoretical Biology*, 71:185–207, 1978.

[Ran81] Robert Ransom. *Computers and Embryos: Models in Developmental Biology*. John Wiley, New York, 1981.

[Rap84] Kenneth B. Raper. *The Dictyostelids*. Princeton University Press, Princeton, NJ, 1984.

[RG78] Gary Rogers and Narendra S. Goel. Computer simulation of cellular movements: Cell-sorting, cellular migration through a mass of cells and contact inhibition. *Journal Of Theoretical Biology*, 71:141–166, 1978.

[RM84] R. Ransom and R.J. Matela. Computer modeling of cell division during development using a topological approach. *Journal of Embryology and Experimental Morphology: Supplement*, 83:233–259, 1984.

[RMS90] John Reinitz, Eric Mjolsness, and David H. Sharp. A connectionist model of *Drosophilia* blastoderm. In *Principles of Organization in Organisms*, pages 109–118. Addison-Wesley, 1990.

[SC91] Hans B. Sieburg and Oliver K. Clay. The cellular device machine development system for modeling biology on the computer. *Complex Systems*, 5:575–601, 1991.

[Sch88] Jacob T. Schwartz. Some speculations concerning biotechnology. Unpublished Manuscript, 1988.

[Ste70] Malcolm S. Steinberg. Does differential adhesion govern self-assembly processes in histogenesis? Equilibrium configurations and the emergence of a hierarchy among populations of embryonic cells. *Journal of Experimental Zoology*, 173:395–434, 1970.

[Ste75] Malcolm S. Steinberg. Reconstruction of tissues by dissociated cells. In G.D. Mostow, editor, *Mathematical Models for Cell Rearrangement*, chapter 6, pages 82–99. Yale University Press, 1975.

[Ste78] Malcolm S. Steinberg. Cell-cell recognition in multicellular assembly: levels of specificity. In *Cell-cell recognition*, pages 25–49. Cambridge University Press, Cambridge, 1978.

[Str91] Bjarne Stroustrup. *The* C++ Programming Language. Addison-Wesley, second edition, 1991.

[Sul82] Deborah Sulsky. *Models of Cell and Tissue Movement.* PhD thesis, Courant Institute of Mathematical Sciences, June 1982.

[TD81] K.J. Tomchik and P.N. Devreotes. Adenosine 3', 5'- Monophosphate waves in *Dictyostelium Discoideum*: A demonstration by isotope dilution-fluorography. *Science*, 212:443–446, 1981.

[Tho17] D'Arcy Thompson. *Of Growth and Form.* Cambridge University Press, Cambridge, 1917.

[TM89] John J. Tyson and J.D. Murray. Cyclic AMP waves during aggregation of *Dictyostelium* amoeba. *Development*, 106:421–426, 1989.

[Tof87] Tommaso Toffoli. *Cellular Automata Machines : A New Environment for Modeling.* MIT Press, 1987.

[Tur52] Alan Turing. The chemical basis of morphogenesis. *Phil. Trans. Roy. Soc. B*, 237:37–72, 1952.

[Wad66] C.H. Waddington. *Principles of Development and Differentiation.* Macmillan Company, New York, 1966.

[WE75] O.K. Wilby and D.A. Ede. A model generating the pattern of cartilage skeletal elements in the embryonic chick limb. *Journal Of Theoretical Biology*, 52:199–217, 1975.

[Wil07] H.V. Wilson. On some phenomena of coalescence and regeneration in sponges. *Journal of Experimental Zoology*, 5:245–258, 1907.

[WMS92] Deborah Wessels, John Mourray, and David R. Soll. Behavior of *Dictyostelium* amoebae is regulated primarily by the temporal dynamic of the natural cAMP wave. *Cell Motility and the Cytoskeleton*, 23:145–156, 1992.

Index