A Framework for Three-Dimensional Simulation of Morphogenesis

Trevor Cickovski[1], Chengbang Huang[1], Rajiv Chaturvedi[1], Tilmann Glimm[2], H.G.E. Hentschel[2], Mark Alber[3], James A. Glazier[4], Stuart A. Newman[5], Jesús A. Izaguirre[1]*

[1]Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556, United States

[2]Department of Physics, Emory University, Atlanta, GA 30322, United States

[3]Department of Mathematics, University of Notre Dame, Notre Dame, IN 46556, United States

[4]Biocomplexity Institute and Department of Physics, Indiana University, Bloomington, IN 47405, United States

[5]Department of Cell Biology and Anatomy, New York Medical College, Valhalla, NY 10595, United States

[*] Author to whom correspondence should be addressed. E-mail: `izaguirr@nd.edu`.

NSF grants IBN-0083653, IBN-0313730, and ACI-0135195 partially funded this research. James Glazier acknowledges support from NASA grant NAG3-1619, an IBM Innovation Institute Award and a Pervasive Technology Laboratory Fellowship. Trevor Cickovski was funded by an Arthur J. Schmitt Fellowship. Special thanks to Joseph Coffland and Todd Schneider for their contributions to the design of CompuCell3D.

Abstract

We present COMPUCELL3D, a software framework for three-dimensional simulation of morphogenesis in different organisms. COMPUCELL3D employs biologically relevant models for cell clustering, growth, and interaction with chemical fields. COMPUCELL3D uses design patterns for speed, efficient memory management, extensibility, and flexibility, permitting the simulation of various organisms. We verify COMPUCELL3D by building a model of growth and skeletal pattern formation in the avian (chicken) limb bud. Binaries and source code are available, along with documentation and input files for sample simulations, at `http://www.nd.edu/~lcls/compucell`.

Keywords – Cellular Potts Model (CPM), reaction-diffusion, cellular automata, morphogenesis, Extensible Markup Language (XML)

Index Terms

D.1.5. [Software/Software Engineering]: Programming Techniques – *Object-Oriented Programming*; D.2.13.b [Software/Software Engineering]: Reusable Software – *Reusable Libraries*; E.2.b [Data]: Data Storage Representation – *Contiguous Representations*; G.1.8 [Mathematics of Computing]: Numerical Analysis – *Partial Differential Equations*; J.3.a [Computer Applications]: Life and Medical Science – *Biology and Genetics*

I. INTRODUCTION

Morphogenesis is the structural development of an organism and its organs, involving cell differentiation, growth and migration, bulk changes in tissue shape, and the secretion, resorption and diffusion of extracellular materials (*e.g.*, proteins). Cell interactions via secreted and membrane-bound chemicals generate biologically significant patterning instabilities that we can describe mathematically and implement computationally [1]–[8], allowing us to *model* morphogenesis [9]. The Cellular Potts Model (*CPM*) provides a well-defined framework for simulations of morphogenesis [2]. The CPM is a grid-based stochastic model designed to accurately simulate cell interactions and movement. Some of the many studies using the CPM include Mombach and Glazier's [10] study of chicken limb retinal cells, Marée's [11] study of *Dictyostelium discoideum*, and Jiang's [12] studies of liquid flow during foam drainage and of foam rheology [13].

This paper presents COMPUCELL3D, a three-dimensional (*3D*) multiscale [14] framework for modeling morphogenesis. COMPUCELL3D takes a hybrid approach to modeling morphogenesis

[15], using a combination of discrete cellular automata and continuum methods. We implement the CPM as a cellular automaton [16] governing cell interaction, along with reaction-diffusion (*RD*) equation solvers to establish surrounding chemical gradients. Domain growth is another key factor. COMPUCELL3D includes a three-dimensional density-dependent growth algorithm. Researchers have studied the effect of uniform [17] and spatially nonuniform [18] growth. Dillon and Othmer implemented a domain growth model of early limb development using a continuum approach [19]. Navier-Stokes, reaction-advection-diffusion (*RAD*) equation solvers have reproduced two-dimensional patterns and simulated growth. Fairly coarse RAD models are fast. However, solving the RAD equations in detail is difficult because advection and moving boundaries can cause numerical instability.

COMPUCELL3D can also work in tandem with other existing software frameworks treating subcellular and supercellular phenomena. One of the many related software frameworks is BioSPICE [20], [21], for modeling dynamic cellular network functions. BioSPICE can clarify complex intracellular biochemical networks [22], to simulate cell division, circadian rhythms, bacterial sporulation, and gene transcription [23]. CellO [24] is an object-oriented tool that can model cell characteristics such as the cell cycle, apoptosis, induction, and differentiation. In contrast to the CPM, CellO uses a grid-independent method to model cell motion, using attractive forces to model cell adhesion and repellent forces to model cell elasticity. NEURON [25], [26] provides a simulation environment for neuron modeling, specifically oriented to problems where cell membrane properties are complex. NEURON can treat multiple, large-scale groups of cells and connections. E-Cell [27], [28] is an object-oriented software suite for analysis of large-scale biological interactions, including biological cells, and researchers are currently developing E-Cell 3 as a platform for integration of multiple algorithms such as reaction-diffusion, cellular automata, and Gillespie's algorithm [29]. Finally, Virtual Cell [30], [31] can model cellular physiology by allowing a user to define both biological models such as species, reactions, structures, and cellular geometry, and mathematical models via a general purpose solver for steady and unsteady solutions of algebraic equations, including partial differential equations (*PDE*s) and ordinary differential equations (*ODE*s). Huang *et al*. [32] use agent-based modeling to simulate the behavior of Natural Organic Matter (*NOM*), or compounds resulting from the natural breakdown of animal and plant material. In general, agent-based models are highly flexible but slower than the CPM or equation-based models. For this reason we implement a hybrid PDE-CPM model in COMPUCELL3D.

In order to validate COMPUCELL3D, we built a three-dimensional model of skeletal pattern formation in the experimentally well-studied avian limb. Because cartilage first establishes the skeletal pattern before being replaced by bone, we call this patterning *chondrogenic* (cartilage-forming). During embryonic development, the vertebrate limb progressively generates a sequence of increasing numbers of cartilage elements proximo-distally. That is, the first elements to form are those closest (*proximal*) to the body wall, and the last are those farthest (*distal*) from the body. In a forelimb, this sequence begins with the humerus, followed by the radius and ulna, then the carpals and metacarpals, and finally the digits. Although the bones at any given proximo-distal level are more similar to each other than to those farther up or down the limb, they also differ significantly in the antero-posterior direction, that is, the direction defined by the thumb to the little finger. Figure 1 shows a schematic timeline of patterning in the avian limb bud, viewed with the proximo-distal axis running from left to right, and the antero-posterior axis running from top to bottom. The dorso-ventral axis of the limb, defined by the axis from the back of the hand to the palm, points out of the page in this representation.

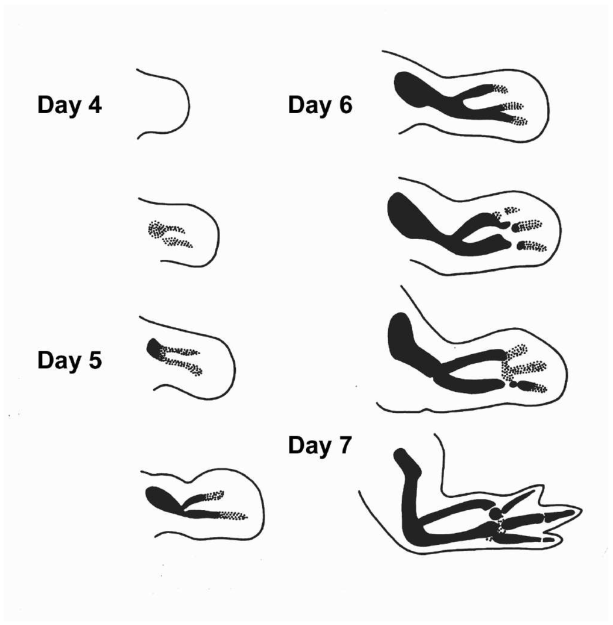


Fig. 1. Timeline of chick limb skeletal pattern formation. Drawings are based on transverse sections of wing buds. For all panels proximal is left, distal right, anterior up and posterior down. From [7], with modifications.

We first describe the CPM in detail, along with ways in which it can model biological mechanisms such as cell adhesion, cell growth and division, reaction-diffusion, chemotaxis and haptotaxis, and cell type and state. We then describe how COMPUCELL3D addresses the inherent issues present in computational modeling of morphogenesis. Next, we describe various software techniques which we used in the design of COMPUCELL3D. These include polymorphism to make the framework extensible and user-friendly, as well as various computational techniques such as "offset-neighbor evaluation" which uses lazy calculation of neighbor pixels in a grid, and provides a four-fold increase in computational speed and a ten-fold reduction in memory consumption compared to standard alternatives. Finally, we present the three-dimensional avian limb-bud simulation with domain growth that we used to validate COMPUCELL3D. We provide sample input files for this simulation, along with instructions for running COMPUCELL3D, on the COMPUCELL website [33].

II. A MODEL FOR MORPHOGENESIS

The CPM uses a lattice to describe cells, and associates an integer index with each lattice site (*voxel*) to identify the spatial extent and location of each cell at any instant. The index value at a lattice site is σ if the site lies in cell σ. *Domains* in the lattice (the collection of lattice sites with the same index) represent cells. A cell is thus a set of discrete components that can rearrange to produce cell shape changes and motion. The CPM follows the principle of energy minimization, with the configuration of cells gradually rearranging to reduce the generalized pattern energy.

Figure 2 shows three two-dimensional cells and the extracellular matrix (ECM), which requires four distinct indices. It also demonstrates the scheme for determining pixel neighbors and their levels, a key part of the Extended CPM.

A. Principle of Energy Minimization

In the CPM an effective energy, *E*, determines cell interactions, motion under cytoskeletal fluctuations, response to external chemical stimuli, differentiation, and division. The effective energy contains true energies (*e.g.*, cell-cell adhesion) and terms that mimic energies (*e.g.*, the response of a cell to a chemotactic gradient). A pattern evolves under strong damping to reduce its energy. Upadhyaya [34] and Marée [35] have justified the CPM quantitatively, reproducing the behavior of different kinds of cell aggregates. The dynamics favor connected domains of lattice sites with the same index.

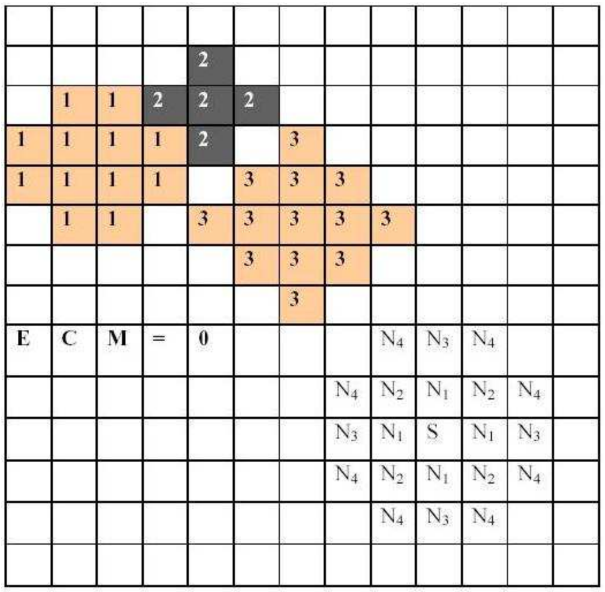


Fig. 2. The Extended CPM grid showing cells and ECM. The shading denotes the cell type. Different cells (for example cells 1 and 3) may have the same type. A site S connects to up to fourth neighbor pixels (N1,. . . ,N4).

In mixtures of liquid droplets, thermal fluctuations of the droplet surfaces cause diffusion, or *Brownian motion*, leading to energy minimization. The simplest phenomenological assumption is that an *effective temperature, T,* drives cell membrane fluctuations. *T* defi nes the size of the typical fluctuation. We implement fluctuations using the Metropolis algorithm for Monte-Carlo Boltzmann dynamics. If a proposed change in lattice confi guration (a change in the indices associated with the voxels of the lattice) produces a change in effective energy ΔE, we use the *acceptance function*:

$$P(\Delta E) = \begin{cases} 1, & \text{if } \Delta E \leq 0, \\ e^{-\Delta E/kT}, & \text{if } \Delta E > 0, \end{cases} \tag{1}$$

where *k* is a constant converting *T* into units of energy.

E includes terms to describe each biological mechanism that we will employ in a model, *e.g.*:

$$E = E_{Contact} + E_{Volume} + E_{Chemical}. \tag{2}$$

We describe each of these terms below.

1) **Cell-Cell Adhesion:** In Eq. (2), $E_{Contact}$ describes the net adhesion/repulsion between two cell membranes. It is the product of the binding energy per unit area, $J_{\tau,\tau'}$, and the total area. $J_{\tau,\tau'}$ depends on the types of the interacting cells, τ and τ'. The equation for $E_{Contact}$ is:

$$E_{Contact} = \sum_{(i,j,k),(i',j',k')} J_{\tau(\sigma),\tau'(\sigma')} \cdot \quad (1 - \delta(\sigma(i,j,k), \sigma'(i',j',k'))), \tag{3}$$

where the *Kronecker delta* $\delta(\sigma,\sigma')$ = 0 if $\sigma \neq \sigma'$ and $\delta(\sigma,\sigma')$ = 1 if $\sigma = \sigma'$, ensuring that only links between surface sites in different cells contribute to the cell-adhesion energy.

2) **Cell Growth, Division, and Death:** A cell of type τ has a prescribed *target volume* $v(\sigma,\tau)$ and *target surface area* $s(\sigma,\tau)$. The actual volume and surface area fluctuate around these target values, *e.g.*, due to changes in osmotic pressure, pseudopodal motion of cells, *etc*. Changes also result from growth and division of cells during morphogenesis. E_{Volume} enforces these targets by exacting an energy penalty for deviations. E_{Volume} depends on four model parameters: *volume elasticity*, λ, *target volume*, $v_{target}(\sigma,\tau)$, *membrane elasticity*, λ', and *target surface area* $s_{target}(\sigma,\tau)$:

$$E_{Volume} = \sum_{cells} \lambda_\sigma (v(\sigma,\tau) - v_{target}(\sigma,\tau))^2 + \sum_{cells} \lambda'_\sigma (s(\sigma,\tau) - s_{target}(\sigma,\tau))^2. \tag{4}$$

We model cell growth by allowing the values of $v_{target}(\sigma,\tau)$ and $s_{target}(\sigma,\tau)$ to increase with time. Cell division occurs when the cell reaches a fi xed, type-dependent volume. We model division by starting with a cell of average size, $v_{target} = v_{targetaverage}$, causing it to grow by gradually increasing v_{target} to $2v_{targetaverage}$ and splitting the dividing cell into two cells, each with a new target volume $v_{target}/2$. One daughter cell assumes a new identity (a unique value of σ). We model cell death simply by setting the cell's target volume and target surface area to zero.

3) **Chemotaxis and Haptotaxis:** Cells can respond to chemical signals by moving along diffusible or substrate-bound concentration gradients of a signal molecule. The fi rst mechanism is chemotaxis, the second is haptotaxis. A chemotaxis model requires a representation of the evolving and spatially varying chemical concentration fi eld, and a model mechanism linking the fi eld to the framework for cell and tissue dynamics. The former depends on the particular morphogen molecule. $C(x,y,z)$ is the local concentration of the morphogen molecules in

extracellular space. An effective chemical potential, $\mu(\sigma)$ models chemotaxis or haptotaxis, to incorporate the effective chemical energy into the CPM energy formalism:

$$E_{Chemical} = \sum_{x,y,z} \mu(\sigma(x,y,z)) \cdot C(x,y,z), \tag{5}$$

for a linear response. Higher-order responses are also possible.
Haptotaxis resembles chemotaxis in Eq. (5) but C(x, y, z) does not diffuse.

B. Reaction-Diffusion

Turing [36] introduced the idea that interactions of reacting and diffusing chemicals (usually of two species denoted u_1 and u_2) could form self-organizing instabilities that provide the basis for biological patterning. We use his continuum, PDE *reaction-diffusion* approach. For simplicity we assume isotropic diffusion (*i.e.*, d^i_j does not depend on *j*), so:

$$\frac{\partial \vec{u}}{\partial t} = D\nabla^2\vec{u} + F(\vec{u}), \tag{6}$$

where $\vec{u} = (u_1, u_2)^T$ and $D = diag(d_1, d_2)$. Without loss of generality, we can assume that $d_1 = 1$, and $d_2 = d$. The term $F(\vec{u})$ describes the reaction kinetics.

C. Cell Type and State

During morphogenesis, cells *differentiate* from initial multipotent stem cells into the specialized types of the developed organism. Though every cell is different, identifying cells with broadly similar behaviors and grouping them into *differentiation types* is standard practice in biology. Cell differentiation from one cell type to another is a comprehensive, qualitative change in cell behavior, generally abrupt and irreversible (*e.g.*, responding to new sets of signals, turning on or off whole genetic pathways). All cells of a particular differentiation type share a set of parameters describing their state, while two different cell types (*e.g.*, muscle and bone) have different parameter sets. Cells of the same type can also exist in different *states*, corresponding to a specifi c set of values for the parameter set of the cell type. A cell's behavior depends on its state; if all parameters associated with their cell type were exactly the same, two model cells would behave identically in the same external environment, while cells of the same type with different parameter values would behave differently.

We model differentiation using a *type-change* map, representing a state automaton. Each type in this map corresponds to a cell type (with a defined parameter set) that exists during a particular morphogenetic process (see section IV). Change of a cell from one type to another corresponds to cell differentiation. The type-change map models regulatory networks by defining the rules governing type change, which take into account the intracellular and intercellular effects of chemical fields.

III. Motivation Behind CompuCell3D

We originally developed a two-dimensional engine for morphogenesis called CompuCell [37]. This work extends CompuCell to 3D. We have improved the efficiency of the engine through better data structures and algorithms, and extensibility through a more thoroughly object-oriented design, which uses scientific design patterns [38], [39].

Specifically, CompuCell consumed too much memory when running 3D simulations. For example, we could not extend the technique of representing grid space as a 2D array to 3D because of the quantity of memory such an array consumed. Consider a relatively small 200^3 grid, with each pixel consuming a very conservative 32 bits. In three dimensions, this grid requires approximately 30 MB of memory, compared to only 156 KB for a 200^2 grid. To reduce memory usage, CompuCell3D implements conservative grid allocation, which only allocates space to a grid pixel if the pixel belongs to a cell and otherwise points to a singleton representing the surrounding medium.

Paging causes a second memory issue. The Metropolis algorithm attempts pixel index flips hundreds of thousands to billions of times per simulation step, requiring new pixel information that many times per step. If the information (for example, attributes) associated with each pixel is heavily scattered in virtual memory, the page fault rate could skyrocket with multiple sets of pixels and attributes consistently swapping in and out, greatly degrading performance due to thrashing. We addressed this problem via two techniques: offset neighbor evaluation and contiguous attribute allocation. The former specifically improves the performance of energy Hamiltonians that need to calculate pixel neighbors (for example, $E_{Contact}$ in the CPM) and neighbor selection. Offset evaluation finds neighbors for a pixel only when necessary, and caches neighbor pixels in an array for later use by the same or different pixels in the grid. Contiguous attribute allocation takes a grid point, and if it forms part of a cell, stores a pointer to a location in memory which contiguously stores the parameter set representing the state of that cell. This technique reduces the page fault rate by storing related information within a single page, to reducing the number of pages swapped-in when we reference a cell in the grid.

Certain programming languages (*e.g.* Fortran [40]) have built-in features for contiguous allocation that could benefi t COMPUCELL3D. However, Fortran lacks flexibility. An object-oriented language provides a solid basis for a flexible framework through *polymorphism* (allowing objects that share common logic to inherit methods and data members from a predefi ned interface), so we implemented the back end of COMPUCELL3D in C++ with careful memory management and certain other techniques to improve flexibility.

We can add new functionality to COMPUCELL3D using six different simulation objects, each with its own predefi ned interface:

1) **Energy function**: Computes energies used by the CPM. An example is $E_{Contact}$ (Eq. (3)), implemented as a `ContactEnergy` energy function in the COMPUCELL3D source.
2) **Acceptance function**: Computes the probability of accepting a CPM pixel flip. An example is the Metropolis acceptance function of Eq. (1).
3) **Steppable**: Provides functionality to execute a routine after every *n* simulation *Monte Carlo steps (MCS)* (a Monte Carlo step takes N pixel flips, where N is the number of lattice points). An example would be a dumper that outputs grid data for visualization, perhaps representing chemical concentrations or current cell types.
4) **Cell-Change Watcher**: Provides functionality to execute a routine after every successful pixel flip. A volume calculator is an example, since a pixel flip changes the volume of two cells.
5) **Stepper**: Provides functionality to execute a routine after every pixel flip attempt. An example application is a deallocator of memory for dead cells (those with zero volume). We perform the zero-volume check in a cell-change watcher and set a flag. We then check the flag in a stepper and deallocate if necessary. This sequence avoids null references if the flip has additional watchers to execute.
6) **Plugin**: Encapsulates the functionality of a combination of the previous objects. An example of this situation is the implementation of cell mitosis. A cell mitoses if its volume exceeds the global doubling volume. Therefore, a successful pixel flip (cell-change watcher) requires

checking the volume of the cell with the added pixel and setting a boolean flag if the cell's volume is higher than the doubling volume. The stepper checks the flag before the next pixel flip attempt and invokes with the cell mitosing if the flag is set. This sample plugin would inherit from two interfaces: the cell-change watcher and the stepper, as a result possessing both their abilities.

These simulation objects work in conjunction with a COMPUCELL3D XML input configuration file, which provides input elements: normally by the addition of a few lines of code. Plugins and steppables implement functions to read from the XML configuration file and so can accept input variables and values from it. This benefit of a configuration file extensible by a single method comes with the inherent cross-platform compatibility of XML, permitting the use of COMPUCELL3D on machines running different operating systems.

IV. IMPLEMENTATION OF TECHNIQUES AND BIOLOGICAL CONCEPTS IN COMPUCELL3D

This section explains how we translate the various biological phenomena that compose our simulations into a COMPUCELL3D software implementation. We also include a more detailed description of the software techniques that we outlined in the previous section.

A. *Biological Cell*

A cell *factory* creates COMPUCELL3D biological cells. Factories are useful techniques in highly polymorphic object-oriented design due to their runtime decisions on derived-class object creation and deletion [41]. We use the BasicUtils library, which contains an implementation of a `BaseDynamicClassFactory` with `virtual` functions for allocation and deallocation analogous to `new` and `delete` functions in C++ [42]. This organization improves software flexibility, since we can now allocate and deallocate an object of a derived class without specifying the actual derived class to which it belongs.

A pointer to a set of contiguous memory locations that encompasses all elements of a cell's parameter set represents the basic `Cell` unit. Parameters include center-of-mass coordinates, volume, surface area, *etc.* A *dynamic class node* (*DCN*) represents each parameter. A DCN improves flexibility by allowing users to define customized `init()` methods that execute parameter value initialization for each individual DCN, and aids memory management by allowing access to a given DCN via a specifi c pointer and offset.

The user can add parameters to each cell by *registering* new DCNs. A registered DCN stores its size (in bytes) and offset from each `Cell` pointer. We defi ne the offset as the size (in bytes) of the total number of DCNs registered thus far. Figure 3 shows a schematic of the contiguous memory allocation for a cell, assuming one word blocks.

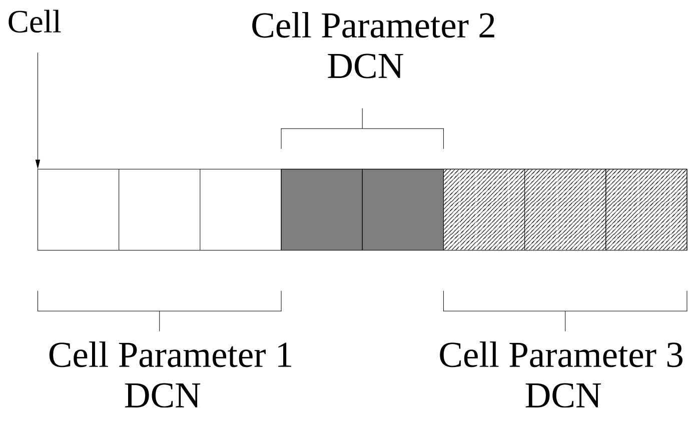


Fig. 3. Representation in memory of a `Cell` object with three attributes represented as dynamic class nodes of size 12, 8 and 12 bytes. Each cell takes 32 bytes of contiguously allocated memory. This fi gure assumes one word blocks.

COMPUCELL3D performs this allocation for every single `Cell` in the simulation. We can access `Cell` parameters represented as DCNs by supplying the `Cell` pointer and the name of the DCN, since the DCN stores the offset from the `Cell` pointer.

In this scheme, if the amount of memory a `Cell`'s attributes consume is small enough to fi t into one virtual page, the fragmentation within each `Cell` object is zero. The method also reduces global external fragmentation. Consider a simulation with a large number of `Cell`s, each with large parameter sets. If the memory allocation for each individual parameter set for each `Cell` is not contiguous, thousands of parameters of varying sizes will be scattered in memory, creating many small holes and potentially requiring frequent compaction. On the other hand, using DCNs for parameters enforces contiguous parameter set allocation for each `Cell`, as long as all attributes can fi t into one page. Thus data sets for `Cell`s lie scattered in memory, rather than individual parameters for each `Cell`, a much better granularity. Contiguous allocation also allows us to take advantage of spatial locality which can drastically reduce page faults.

B. Cell Grid and Watchers

We implement the 3D cell grid as a resizable array of `Cell` object pointers called a *field*, defined by the class `Field3D`. We used conservative grid allocation as mentioned, and after CPM pixel flips, pointers to the medium singleton may change to point to `Cell` objects, and vice-versa.

Because some operations must execute after every change in the `Cell` field, we must keep track of, or *watch*, the `Cell` field. Therefore we declare the `Cell` field not just as a `Field3D` object, but as a `WatchableField3D` object - the only difference being that a `WatchableField3D` keeps an array listing its "watchers." We represent watchers as `CellChangeWatcher` objects, and each defines a method `field3DChange()` which defines the appropriate actions to take upon a change in the `Cell` field. After a `Cell` field update, we pass through the array of watchers and invoke their `field3DChange()` methods. Watcher examples include cell-volume, surface-area or center-of-mass calculators.

A simulation can contain many `Cell` objects. Consider the impact of conservative grid allocation on the previous example with 32 byte `Cell`s. In this case, allocating memory only as needed in the `Cell` grid and having each medium grid point reference a singleton, saves 536,722 `Cell` object allocations. The total memory all `Cell`s consume in the simulation is roughly 81 KB versus up to 16 MB for naive allocation, a savings of about 95%. The savings becomes even larger for larger grids.

C. Energy Computation and Arbitrary Neighbors

The CPM implements a physical description of cells based on the principle of energy minimization. A CPM simulation must account for many different energies (*e.g.*, contact, volume/surface, chemical). Energy functions within COMPUCELL3D inherit from an abstract class `EnergyFunction`, which contains a `virtual` function `changeEnergy()`, which the user defines for each energy function. A single call to a method `registerEnergyFunction()` *registers* an `EnergyFunction`, passing the `EnergyFunction` object as a parameter. Invocation of this method tells COMPUCELL3D to include this energy calculation when deciding whether or not to flip the index of a selected pixel.

Some energy functions (*i.e.*, contact energy) require a grid pixel to compute its interaction with neighboring pixels. To find pixels neighboring a grid point, we implement a `NeighborFinder` singleton which uses offset evaluation. Algorithm 1 gives pseudocode for the offset neighbor

Algorithm 1 Pseudo-code for offset neighbor evaluation.

```
Neighbor Finder:
  1) Pre-processing: Initialize $x$ := 0 and $neighbor\_array$ to be empty.
     $neighbor\_array$ is an array of pairs of points and integer distances, and $n$ := 0;
  2) getNeighbor(int $n$, double &$D$)
     a) while length of $neighbor\_array < n$
        i) $x$ := $x$ + 1;
        ii) for each $(X, Y, Z)$ such that $x = X^2 + Y^2 + Z^2$
            A) for each unique point $Q$ that is a rotation of $(X, Y, Z)$ around the axes
               Add ($Q$, $\sqrt{x}$) to $neighbor\_array$;
     b) $D$ := $neighbor\_array[n]$.distance;
     c) return $neighbor\_array[n]$.point;
  3) To look at level 1 neighbors, distance 1 from a point $P$:
     a) do
        i) $neighbor$ = getNeighbor($n$, $D$) + $P$;
        ii) ... Do something with neighbor ...
        iii) $n$ := $n$ + 1;
        while $D$ <= 1
```

evaluation. The algorithm for offset evaluation assumes that a pixel's neighbors lie within some small constant distance D. Knowing this distance, we first find neighbors *to the origin*, by finding X, Y, Z such that $\sqrt{X^2 + Y^2 + Z^2} \leq D$. We find other neighbors at this distance by rotating point (X, Y, Z) about the origin. Then, for a specific point (x, y, z) we translate these neighbors by adding (x, y, z) to their coordinates, giving us the neighbors of (x, y, z).

Finding the first n neighbors of a given point (x, y, z) requires D^2 integer iterations, where D is the distance within which all n neighbors lie. For each i, we test all possible integer values of X, Y, Z between 0 and $\sqrt{i}$. If $X^2 + Y^2 + Z^2 = i$, we add the neighbors at distance $\sqrt{i}$ to a neighbor array, until we reach n neighbors. In this way, we insert the neighbors into the neighbor array in order of distance. Because we calculate neighbors with respect to the origin, we can reuse them to find the neighbors of multiple grid pixels. Since executing this entire algorithm to calculate each neighbor is prohibitively slow, we cache the neighbors into an array for later use. With offset evaluation and dynamic array growth, we calculate neighbors with value n only as needed.

This lazy evaluation technique for calculating arbitrary neighbors of a pixel greatly improves simulation speed. To illustrate this improvement, compare a 3D CPM algorithm with offset evaluation (program B) to a different version (program A) that forces each grid pixel to maintain pointers to all first, second, third and fourth neighbors. We ran the two versions on a PC with an AMD Athlon XP 1800+ at 1.6 GHZ, and 512 MB of memory running RedHat Linux 9.0, kernel 2.4.22. The field dimensions were 71x36x211 pixels with a cell size of 2x2x2, an initially uniform cell distribution, temperature of 1.0, data output every 10 steps, and 539,316 flip attempts per simulation step. We used contact, volume and chemical energies for energy computation in the CPM algorithm, and turned off visualization to restrict performance measurement to computation. We measured times using ‘real’ or wall clock times using the Linux `time` command.

Version A has a much longer startup time than program B. Subtracting this initial startup time (time for the first timestep), the timing and memory usage are:

	A	B	Ratio A/B
Execution Time For 100 CPM Flips	1959 s	501 s	3.91
Memory Usage	70656 KB	6564 KB	10.76

Therefore even not accounting for the expensive initialization costs in version A, the offset neighbor evaluator yields a four-fold speedup in computation. The offset evaluator for neighbors also yields a ten-fold memory savings, since a pixel in memory does not need pointers to all neighbors.

The current implementation of CompuCell3D includes just one acceptance function, which implements the Metropolis algorithm with a Boltzmann acceptance function, but we can create custom acceptance functions, for example to implement Kawasaki or Glauber dynamics [43].

D. Cellular Automaton

CompuCell3D includes a cellular automaton to describe each cell’s transitions between *types*, and to hold a set of variables and corresponding values, which make up the cell’s *state*. Each individual type of cell has three methods defined:

1) A method for initializing state variables.
2) A method for updating states (changing state variable values overtime).

3) A method for changing type.

Cells of different types may react differently to external and internal conditions. Therefore the definitions of each of these methods will vary with cell type. COMPUCELL3D changes cell types after each CPM step, so we implement the automaton as a `TypePlugin`. When the CPM selects a pixel candidate, it finds the corresponding cell and invokes the appropriate methods for updating its state and type. If the cell's type changes, we reinitialize the state variables appropriately for the new type.

E. External Chemical Concentration

COMPUCELL3D implements both resizable field structures and platform-independent file reading of external chemical concentrations. Various simulation objects such as plugins, steppables, or steppers can update these fields.

F. Flexibility

We can extend COMPUCELL3D by encapsulating new functionality (*e.g.*, new energy functions, new cell-change watchers, or new rules for cell differentation and state) into one of the six simulation objects: plugins, steppables, steppers, acceptance functions, energy functions and cell-change watchers. COMPUCELL3D accepts a configuration file input. This configuration file can add or remove plugins and steppables, as well as 3D graphical field renderer objects, from simulations. The COMPUCELL webpage [33] provides complete COMPUCELL3D configuration file syntax along with configuration files for this paper's validation simulation.

V. VALIDATION SIMULATION

Chaturvedi *et al*. [44] and Izaguirre *et al*. [37] used the systems-biology approach of integrating discrete and continuous models for biological mechanisms to describe a reduced, 2D model of vertebrate limb development. Figure 4 schematically represents the major axes and the progress of chondrogenic patterning in a developing vertebrate forelimb at a stage part-way through development. The humerus (in dark gray) has already differentiated into cartilage, the radius and ulna (light gray) are beginning to form, and the wrist bones and digits are still to form.

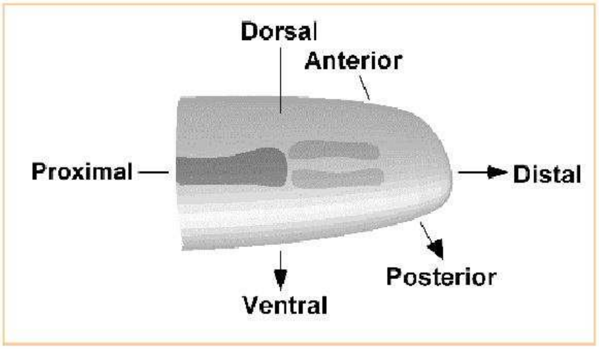


Fig. 4. Schematic diagram of chick limb organogenesis.

Here, we use discrete models to describe cell movement, division, and interactions, and differentiation from multipotent cells into specific cell types. We solve RD equations to obtain the concentration field of a diffusible *activator* molecule, which we identify with the positive autoregulatory growth factor TGF-β [45]. An *inhibitor* molecule that suppresses the production, or downstream effects, of the activator [46] is a necessary ([6], [36]) component of the pattern-forming RD equations. We assume the cells respond to the activator by producing a secreted molecule, fibronectin, to which they adhere (see [3] and [5] for additional details).

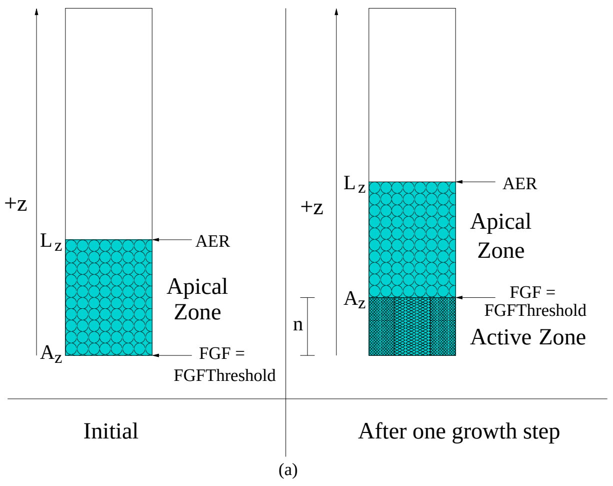
The RD equations are:

$$\begin{aligned}\frac{\partial c_a}{\partial t} &= \gamma[(J_0 + J_a(c_a)\beta(c_a))R_0 - k_a c_a c_i] + b_a(c_a - c_{as})^3 + (d_{ax}\frac{\partial^2 c_a}{\partial x^2} + d_{ay}\frac{\partial^2 c_a}{\partial y^2} + d_{az}\frac{\partial^2 c_a}{\partial z^2}),\\ \frac{\partial c_i}{\partial t} &= \gamma[J_i(c_a)\beta(c_a)R_0 - k_i c_a c_i] + b_i(c_i - c_{is})^3 + D(d_{ix}\frac{\partial^2 c_i}{\partial x^2} + d_{iy}\frac{\partial^2 c_i}{\partial y^2} + d_{iz}\frac{\partial^2 c_i}{\partial z^2}),\end{aligned}$$

where c_a and c_i represent the respective concentrations of activator and inhibitor, c_{as} and c_{is} are the spatially homogeneous steady states for the activator and inhibitor concentrations, and R_0 denotes the average cell density.

We employed a modified form of the equations of Hentschel *et al*. [3]. We kept only the two dominant equations and added two terms $b_a(c_a - c_{as})^3$ and $b_i(c_i - c_{is})^3$ to enforce stability [47], [48]. These equations represent known biological interactions in the chick limb. We use them to generate the chemical field in our validation simulation. The emergence of the sequence of bone structures results from changes in the domain geometry as well as in the reaction kinetics.

COMPUCELL3D extends the originally 2D RD model to 3D. The concentration of the activator



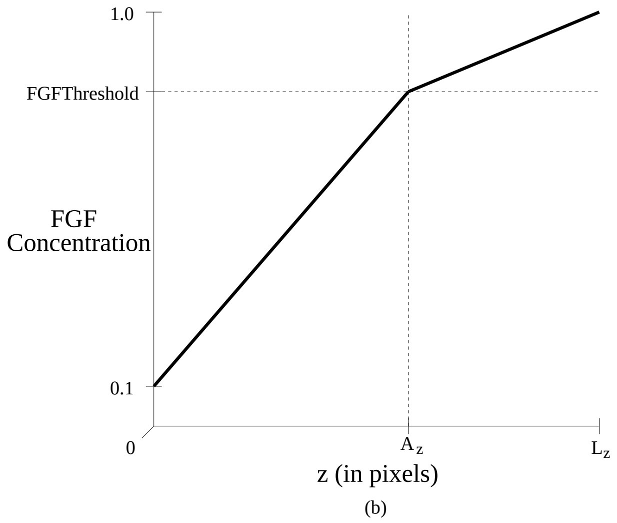


Fig. 5. (a) One step of the growth algorithm applied to the chick limb simulation. Cells are initially uniformly distributed and occupy a third of the grid, as does the Apical Zone. After one growth step the Apical Zone shifts up by *n* rows in the *z* direction allowing cells in the newly formed Active Zone to condense into patterns. (b) The distribution of FGF with respect to the *z*-coordinate of the grid point.

chemical, to which cells chemotax and respond by increasing their adhesivity levels, occupies a second matching grid. We measure activator concentration for a cell grid pixel using its corresponding location in the concentration grid. High activator concentration induces production of a second secreted molecule, which we identify with fi bronectin. As soon as the TGF-β concentration at a grid pixel exceeds a threshold, the corresponding cell secretes fi bronectin at that pixel location, at a user-specifi ed rate. Fibronectin concentration, in turn, supplies the chemical energy for haptotaxis in Eq. (5).

We superimpose a third chemical, FGF, on the grid to control growth and reaction-diffusion. FGF concentration monotonically increases with *z* across the entire grid, with normalized values between 0.1 and 1. A domain known as the *Apical Zone*, within which no reaction-diffusion can occur, moves proximo-distally within the grid. Figure 5(a) shows pictorally a 2D example of one step of the growth algorithm starting from a uniform cell distribution.

Initially, cells are uniformly distributed in a user-specifi ed fraction of the overall grid (with respect to *z*) and the Apical Zone occupies this grid fraction. When the grid grows by *n* rows in the positive *z* direction, the Apical Zone shifts upward in the positive *z* direction by *n* rows, allowing cells in the *Active Zone* below it to react and form patterns. We specify a variable `FGFThreshold` which will set the lower boundary of the Apical Zone. The *z* value in the grid where the FGF concentration is equal to this threshold defi nes the lower boundary of the Apical Zone, and the *z* value at which the FGF concentration is at its maximum defi nes the *Apical Ectodermal Ridge* (*AER*), the upper boundary of the Apical Zone. When a cell attempts to react to the surrounding activator we check the FGF concentration at its location in the grid. The cell cannot react if the concentration is greater or equal to `FGFThreshold` (implying that it is in the Apical Zone). The grid thus divides into two "zones," the Apical Zone and the Active Zone. The FGF concentration changes linearly *within each zone* (see Figure 5(b)):

$$FGF(z) = \begin{cases} \frac{z-A_z}{L_z-A_z}(1-F_t)+F_t, & z \geq A_z, \\ \\ \frac{z}{A_z}(F_t-0.1)+0.1, & z < A_z. \end{cases} \tag{7}$$

A_z is the *z*-coordinate of the lower boundary of the Apical Zone, L_z the *z*-dimension of the grid itself, F_t the `FGFThreshold`, and *z* is the proximodistal position.

TGF-β also plays a role in our customized cellular automaton. Cells can be of type `Non-Condensing` or `Condensing`, with `Condensing` cells being more adhesive. When a cell outside the Apical Zone occupies a point in the grid where the TGF-β concentration exceeds a threshold, the cell becomes `Condensing`, otherwise the cell is `NonCondensing`. Cells are initially `NonCondensing` and cells cannot condense within the Apical Zone. Figure 6 shows the state diagram for these cell types. Only `Condensing` cells have an energy term for haptotaxis to fi bronectin (Eq. (5)).

We implemented the algorithm for domain growth in COMPUCELL3D as a `Steppable` object. We defi ne density as follows:

$$Density = \frac{C}{T} \times 100, \tag{8}$$

where *C* is the total number of pixels which contain cells, *T* is the total number of pixels, and we specify a *range box*, the domain over which to calculate the density. We also specify the input variables `delay` (in timesteps), `threshold` (percentage), and `n` (number of rows to grow at a time) for simulations with growing domains. If the density within the range box exceeds the `threshold`, the mathematical grid grows by `n` rows in the *z* direction. A growth step turns off

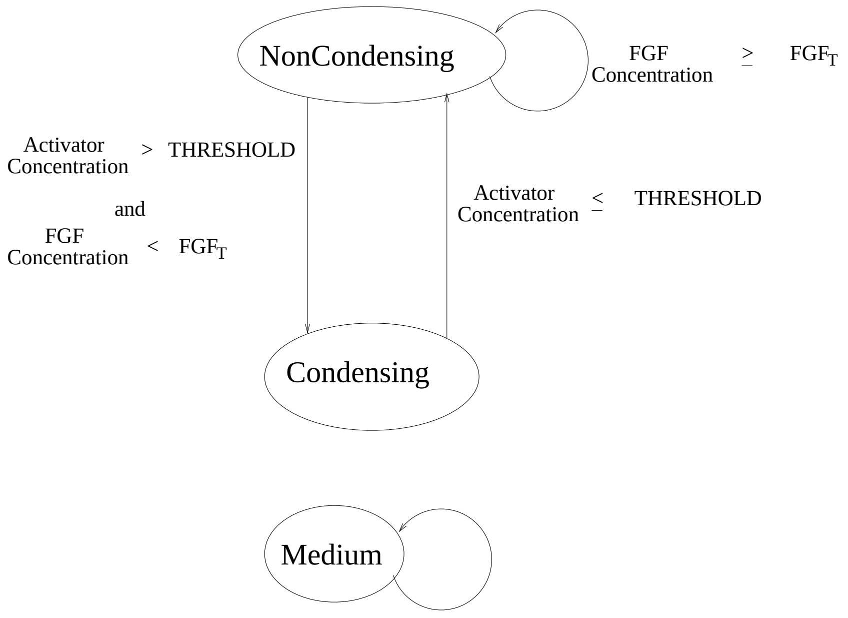

NonCondensing
FGF Concentration ≥ FGF$_T$
Activator Concentration > THRESHOLD
and
FGF Concentration < FGF$_T$
Activator Concentration ≤ THRESHOLD
Condensing
Medium

Fig. 6. Cellular-automaton state map implemented using COMPUCELL3D for the sample simulation.

cell mitosis for `delay` steps, and the algorithm repeats. This `delay` allows the cells time to cluster and fill in the `n` initially empty added rows. Figure 7 shows the domain growth algorithm in COMPUCELL3D.

Figure 8 shows our simulation of 3D limb growth and pattern formation at 2250 MCS, 3250 MCS and 4250 MCS, visualized using Ogle [49]. We ran the simulation for 4250 MCS and we show two sets of three screenshots, the first showing all cells and the second showing only `Condensing` cells (green). We have superimposed the `Condensing` cells on the grid containing all the cells for clarity. `NonCondensing` cells in the Apical Zone, where no condensation can occur, are red, and `NonCondensing` cells outside the Apical Zone are blue.

VI. CONCLUSIONS

We have presented COMPUCELL3D and a validation simulation of skeletal pattern formation in the avian limb bud. The various software techniques we applied yielded substantially reduced computation time and memory consumption, which are especially important when dealing with large three-dimensional grids. We discussed techniques for creating a flexible and extensible software

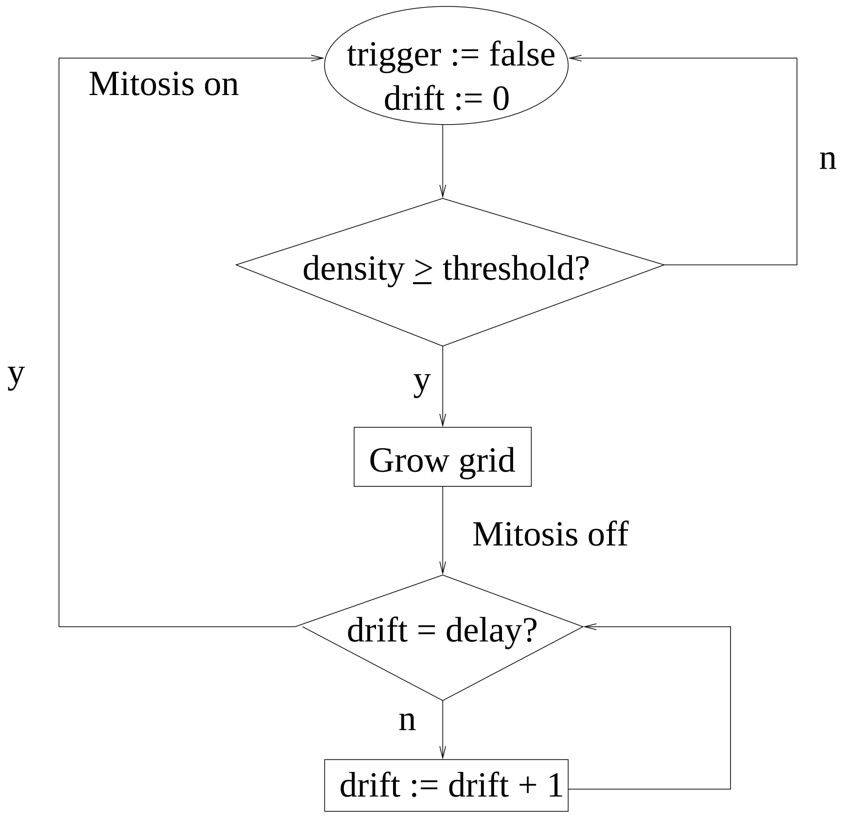


Fig. 7. Growth algorithm.

package, and the object-oriented implementation of various biologically signifi cant components of morphogenesis modeling such as the CPM, chemical gradients, cell growth and cellular automata.

Extending COMPUCELL3D currently involves adding new simulation objects coded in C++ and interfacing them to the framework through the confi guration fi le. While this structure lays a solid groundwork for simple extensibility, we realize that many biocomplexity researchers and potential users of COMPUCELL3D may not be experienced C++ programmers. For this reason, we are currently interfacing COMPUCELL3D with BIOLOGO [50], an XML-based domain-specifi c language that we have designed for morphogenesis simulation engines. We designed the syntax of this language to be easy to understand by researchers studying morphogenesis or the CPM. Together BIOLOGO and COMPUCELL3D allow for more flexible models with different energy computations and cellular automata, and allow users to extend COMPUCELL3D the framework by writing programs in a comprehensible language that automatically generates C++ code. We are also integrating more realistic geometry and better growth models that include moving boundaries.

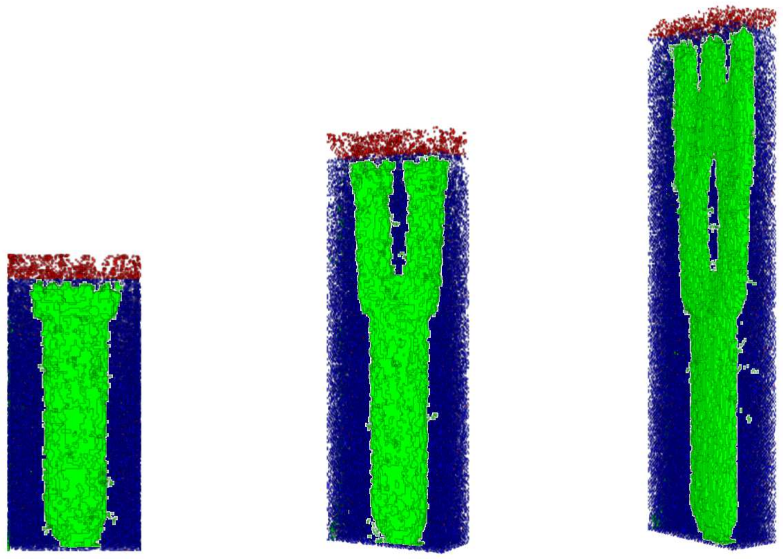


Fig. 8. 3D chicken limb growth and patterning visualized with Ogle. We successively rotate the grid to demonstrate the three-dimensional structure. On a visualization of all cells, we superimpose one showing only `Condensing` cells (in green).

References

[1] J. A. Glazier and F. Graner. A simulation of the differential adhesion driven rearrangement of biological cells. *Phys. Rev. E.*, 47:2128–2154, 1993.

[2] F. Graner and J. A. Glazier. Simulation of biological cell sorting using a two-dimensional extended potts model. *Phys. Rev. Lett.*, 69:2013–2016, 1992.

[3] H. G. E. Hentschel, T. Glimm, J. A. Glazier and S. A. Newman. Dynamical mechanisms for skeletal pattern formation in the vertebrate limb. Proc R Soc Lond B Biol Sci, 271:1713–1722, 2004.

[4] Y. Jiang, H. Levine, and J. Glazier. Possible cooperation of differential adhesion and chemotaxis in mound formation of *Dictyostelium*. *Biophys. J.*, 75:2615–2625, 1998.

[5] M. A. Kiskowski, M. S. Alber, G. L. Thomas, J. A. Glazier, N. B. Bronstein, J. Pu and S. A. Newman. Interplay between activator-inhibitor coupling and cell-matrix adhesion in a cellular automaton model for chondrogenic patterning. *Dev. Biol.* 271:372–387, 2004.

[6] H. Meinhardt. *Models of Biological Pattern Formation*. London: Academic Press, 1982.

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

[8] W. Zeng, G. L. Thomas, S. A. Newman and J. A. Glazier. A novel mechanism for mesenchymal condensation during limb chondrogenesis in vitro. *Mathematical Modeling and Computing in Biology and Medicine: 5th Conference of the European Society of Mathematical and Theoretical Biology*, Milan, V. Capasso and M. Ortisi, 2002.

[9] L. I. Held Jr. *Imaginal Discs: The Genetic and Cellular Logic of Pattern Formation*. New York: Cambridge University Press, 2002.

[10] J. Mombach and J. Glazier. Single cell motion in aggregates of embryonic cells. *Phys. Rev. Lett.*, 76:3032–3035, 1996.

[11] S. Mar´ee. *From Pattern Formation to Morphogenesis*. PhD thesis, Utrecht University, Netherlands, Oct. 2000.

[12] Y. Jiang and J.A. Glazier. Foam Drainage: Extended Large-Q Potts Model Simulation. *Phil. Mag. Lett.*, 74:119–128, 1996.

[13] Y. Jiang, P. Swart, A. Saxena, M. Asipauskas, and J. A. Glazier. Hysteresis and Avalanches in Two Dimensional Foam Rheology Simulations. *Phys. Rev. E.*, 59:5819, 1999.

[14] R. Chaturvedi, C. Huang, J. A. Izaguirre, S. A. Newman, J. A. Glazier, M. Alber. On multiscale approaches to 3-dimensional modeling of morphogenesis. Submitted.

[15] R. Chaturvedi, C. Huang, J. A. Izaguirre, S. A. Newman, J. A. Glazier, M. Alber. A hybrid discrete-continuum model for 3D skeletogenesis of vertebrate limb. *Lecture Notes in Computer Science*, Springer-Verlag, New York (to appear).

[16] M. S. Alber, M. A. Kiskowski, J. A. Glazier, and Y. Jiang. On cellular automaton approaches to modeling biological cells, in J. Rosenthal and D.S. Gilliam (Eds.), *Mathematical Systems Theory in Biology, Communication, and Finance*, IMA Volume 134, Springer-Verlag, New York, 2003, p. 1.

[17] K.A. Landman, G.J. Pettet and D.F. Newgreen. Mathematical models of cell colonization of uniformly growing domains. *Bull. Math. Biol.*, 65(2):235–262, 2003.

[18] E.J. Crampin, W.W. Hackborn and P.K. Maini. Pattern formation in reaction-diffusion models with nonuniform domain growth. *Bull. Math. Biol.*, 64(4):747–769, 2002.

[19] R. Dillon and H. G. Othmer. A mathematical model for outgrowth and spatial patterning of the vertebrate limb bud. *J. Theor. Biol.*, 197:297–330, 1999.

[20] BioSPICE Community Web Site: Biology in silicio URL: https://community.biospice.org.

[21] A. Arkin, J. Ross and H. H. McAdams. Stochastic Kinetic Analysis of Developmental Pathway Bifurcation in Phage Lambda-Infected *Escherichia coli* Cells. *Genetics*, 149:1633–1648, 1998.

[22] S. P. Kumar and J. C. Feidler. BioSPICE, 2 *Omics: A Journal of Integrative Biology*, 7(4):335, 2003.

[23] DARPA News Release: *DARPA Releases BioSPICE Software*. URL: https://community.biospice.org/public/bio_release.pdf. 2002.

[24] Cell-O-Sim. URL: http://mbi.dkfz-heidelberg.de/projects/cellsim/cellosim.

[25] NEURON. URL: http://www.neuron.yale.edu/neuron.

[26] M. L. Hines and N. T. Carnevale. The NEURON simulation environment. *Neural Computation*, 9:1179–1209, 1997.

[27] Institute for Advanced Biosciences: E-Cell Project. URL: http://www.e-cell.org.

[28] K. Takahashi, N. Ishikawa, Y. Sadamoto, *et al*. E-CELL2: Multi-platform E-CELL Simulation System. *Bioinformatics*, 19(13):1727-1729, 2003.

[29] D. T. Gillespie. A General Method for Numerically Simulating the Stochastic Time Evolution of Coupled Chemical Reactions. *J. Comput. Phys.*, 22:403–434, 1976.

[30] National Resource for Cell Analysis and Modeling. URL: http://www.nrcam.uchc.edu.

[31] J. Schaff, C. C. Fink, B. Slepchenko, J. H. Carson, and L. M. Loew. A general computational framework for modeling cellular structure and function. *Biophys. J.*, 73:1135–1145, 1997.

[32] Y. Huang, X. Xiang, G. Madey and S. Cabaniss. Agent-based Scientifi c Simulation Using Java/Swarm, J2EE, RDBMS and Automatic Management Techniques. Preprint.

[33] COMPUCELL- LCLS. URL: http://www.nd.edu/˜lcls/compucell

[34] A. Upadhyaya. *Thermodynamic and fluid properties of cells, tissues and membranes*. Ph.D. thesis, University of Notre Dame, USA, 2000.

[35] F. M. Mar´ee and P. Hogeweg. How amoeboids self-organize into a fruiting body: multicellular coordination in dictyostelium discoideum. *Proc. Natl. Acad. Sci. USA*, 98:3879–3883, 2001.

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

[37] J. A. Izaguirre, R. Chaturvedi, C. Huang, T. Cickovski, J. Coffland, G. Thomas, G. Forgacs, M. Alber, H. G.E. Hentschel, S. A. Newman and J. A. Glazier. COMPUCELL: A Multi-Model Framework for Simulations of Morphogenesis. *Bioinformatics*, 20:1129–137, 2004.

[38] A. Shalloway and J. R. Trott. Design Patterns Explained: A New Perspective on Object-Oriented Design. Addison-Wesley, 2002.

[39] C. Blilie. Patterns in Scientifi c Software: An Introduction. *Computing in Science & Engineering*, 4(3):48–53, 2002.

[40] User Notes on Fortran Programming (UNFP): An Open Cooperative Practical Guide. URL: http://www.ibiblio.org/pub/languages/fortran/unfp.html.

[41] A. Alexandrescu. *Modern C++ design: generic programming and design patterns applied*. Addison-Wesley, 2001.

[42] J. Coffland. *BasicUtils Library*, Downloadable from SourceForge. URL: http://compucell.sourceforge.net/phpwiki/index.php/BasicUtils

[43] S. Artz and S. Trimper. Competing Glauber and Kawasaki Dynamics. *Int. J. of Modern Phys. B* 12(23):2385-2392, 1998.

[44] R. Chaturvedi, J. A. Izaguirre, C. Huang, T. Cickovski, P. Virtue, G. Thomas, G. Forgacs, M. Alber, H.G.E. Hentschel, S. Newman, and J. A. Glazier. Multi-model simulations of chicken limb morphogenesis. Springer Verlag LNCS 2659, *Proceedings of the International Conference on Computational Science (ICCS)*, Melbourne, Australia and St. Petersburg, Russia, Part III, 39–49, 2003.

[45] T. Miura and K. Shiota. TGF-beta 2 Acts As an "Activator" Molecule in Reaction-Diffusion Model and Is Involved in Cell Sorting Phenomenon in Mouse Limb Micromass Culture. *Developmental Dynamics*, 217:241–249, 2000.

[46] M. Z. Moftah, S. A. Downie, N. B. Bronstein, N. Mezentseva, J. Pu, P. A. Maher, S. A. Newman. Ectodermal FGFs Induce Perinodular Inhibition of Limb Chondrogenesis In Vitro and In Vivo Via FGF Receptor 2. *Dev. Biol.*, 249:270–282, 2002.

[47] B. Ermentrout. Stripes or Spots? Nonlinear Effects in Bifurcation of Reaction-Diffusion Equations on the Square. *Proc. R. Soc. Lond. A.*, 434:413-417, 1991.

[48] M. Alber, T. Glimm, H. G.E. Hentschel, B. Kazmierczak, S. A. Newman. Stability of n-Dimensional Patterns in a Generalized Turing System: Implications for Biological Pattern Formation. *Nonlinearity*, (to appear).

[49] Ogle Large-Scale Scientifi c Data Visualizer. URL: http://www.cora.nwra.com/Ogle

[50] T. Cickovski and J. Izaguirre. BIOLOGO: A Domain-Specifi c Language for Morphogenesis Simulation Engines. Preprint available. URL: http://www.nd.edu/˜tcickovs/acmtr2e.pdf

Trevor Cickovski is in his third year of graduate study in the department of computer science and engineering at the University of Notre Dame, Notre Dame, Indiana, directed by Dr. Izaguirre. His current research interests include domain-specifi c language development, stochastic simulations of biocomplexity and software engineering. Cickovski has a B.S. in computer science from the University of Notre Dame.

Chengbang Huang is a Ph.D. student in the department of computer science and engineering at the University of Notre Dame, Notre Dame, Indiana, directed by Dr. Izaguirre. His current research interest is to use a multi-model framework to simulate avian limb growth. Huang has an M.S. in computer science from the University of Notre Dame.

Rajiv Chaturvedi is a postdoctoral research associate at the University of Notre Dame, Notre Dame, Indiana. His current research interests include computational biology and software engineering. He has been working on models of biological phenomena occuring at multiple scales, and their integration. Chaturvedi has a Ph.D. in Computational Fluid Dynamics from the Indian Institute of Technology, Bombay.

Tilmann Glimm is a postdoctoral research associate in the Emory University physics department, Atlanta, Georgia, where he is working on the mathematical modeling of limb development and mesenchymal cell condensation. In general, his research interest is the analysis of nonlinear partial differential equations. He has studied at TU Berlin and Emory University and has a Ph.D. in mathematics from Emory.

George Hentschel is a professor of physics at Emory University, Atlanta, Georgia. His current research interests are in the areas of nonlinear and biological physics. Hentschel has a Ph.D. in theoretical chemistry from the University of Cambridge.

Mark Alber is a professor of mathematics, concurrent professor of physics, and director of the Interdisciplinary Center for the Study of Biocomplexity (ICSB) at the University of Notre Dame, Notre Dame, Indiana. His current research interests include methods of nonlinear dynamical systems and statistical mechanics with applications in biology. Alber has a Ph.D. in mathematics from the University of Pennsylvania.

James A. Glazier is a professor of physics, adjunct professor of Informatics, and director of the Biocomplexity Institute at Indiana University, Bloomington, Indiana. His current research interests include biophysics, development, DNA sequence analysis, neuroscience, and the mechanics of liquid foams. Glazier received a B.A. in physics and mathematics from Harvard College, and a Ph.D. in soft condensed matter physics from the University of Chicago.

Stuart A. Newman is a professor of cell biology and anatomy at New York Medical College, Valhalla, New York. He has contributed to several scientific fields including biophysical chemistry, developmental biology, and evolutionary theory. His current research interests include the mechanisms of vertebrate limb development, the dynamics of collagen assembly, and the evolution of morphogenesis. He received an A.B. from Columbia University and a Ph.D. in chemical physics from the University of Chicago.

Jesus A. Izaguirre is an assistant professor of computer science and engineering at the University of Notre Dame, Notre Dame, Indiana. His current research is on efficient methods in chemistry and biology, particularly molecular dynamics, Monte Carlo methods, cellular automata, and analysis of biological networks. He is also interested in the portable implementation of high-performance software for scientific computing. He received a Ph.D. in computer science from the University of Illinois at Urbana-Champaign in 2000. Dr. Izaguirre received a CAREER Award of the National Science Foundation in 2001.