A Framework for Three-Dimensional Simulation of Morphogenesis

Trevor Cickovski¹, Chengbang Huang¹, Rajiv Chaturvedi¹, Tilmann Glimm², H.G.E. Hentschel², Mark Alber³, James A. Glazier⁴, Stuart A. Newman⁵, Jesús A. Izaguirre¹*  

¹Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556, United States  
²Department of Physics, Emory University, Atlanta, GA 30322, United States  
³Department of Mathematics, University of Notre Dame, Notre Dame, IN 46556, United States  
⁴Biocomplexity Institute and Department of Physics, Indiana University, Bloomington, IN 47405, United States  
⁵Department 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.

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, to allow an almost unlimited variety of simulations. We have verified 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 Representa-

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 Kedar Aras, Joseph Coffland, Chris Mueller and Todd Schneider for their contributions to the design of COMPUCELL3D.
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 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]. Experimental studies on cell sorting in vitro [14], [15] and, more recently, in vivo [16]–[18], support an approach to tissue morphogenesis based on consideration of surface forces akin to those that govern the behavior of bubbles and foams. Recent modifications to the CPM further show its biological relevance. In [19], a Hamiltonian is modified to use negative surface energies, constrained surface area and a spin flip energy threshold to improve the correspondence to reality. It is shown that the new model correctly predicts several dynamical behaviors of cells which even the original CPM does not, including the hierarchy of diffusion constants. The influence of surface tension and size on rounding of cell aggregates are studied in [20] using chick embryonic cells and numerical simulations of the CPM. The results show exponential relaxation in both cases as verified in previous studies using 2D Hydra cell aggregates. The relaxation time decreases with higher surface tension as expected from hydrodynamics laws. However, it increases faster than linearly with aggregate size. The results provide an additional support to the validity of the CPM for non-equilibrium situations and indicate that aggregate shape relaxation is not governed by the hydrodynamics of viscous liquids. Merks and Glazier [21] provide a summary of the validity of multi-scale computational models in developmental biology with the cell as the lowest modeling unit. This cell-centered idea that they favor collectively models single cell behaviors such as morphology, response to signals, cell-cell adhesion, the extracellular matrix, and chemotactic and haptotactic responses to gradients,
into a conceptual biological model for tissue and organ level patterns and functions, which is the essence of the CPM. They also present the CPM as their example cell-centered model for cell rearrangement driven by differential adhesion during morphogenesis, and argue its advantages over other cell-centered models in terms of accounting for cell shape, distinguishing between cell adhesion and attraction, and integrating different software frameworks into the model.

This paper presents COMPUCELL3D, a three-dimensional (3D) multiscale [22] framework for modeling morphogenesis. COMPUCELL3D takes a hybrid approach to modeling morphogenesis [23], combining discrete cellular automata and continuum methods. We implement the CPM as a cellular automaton [24] governing cell interactions, along with reaction-diffusion (RD) equation solvers to establish surrounding chemical gradients. Domain growth is another key factor. Researchers have studied the effect of uniform [25] and spatially nonuniform [26] growth on biological development. Dillon and Othmer implemented a domain growth model of the shaping of the developing vertebrate limb using a continuum approach [27]. COMPUCELL3D includes a 3D density-dependent growth algorithm. 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. We avoid solving RAD equations by using a growth algorithm that depends on the cellular density of the Potts lattice, and enforces domain growth when the density exceeds a threshold.

COMPUCELL3D can simulate morphogenesis in both multicellular and unicellular organisms such as Myxobacteria [28]. Myxobacteria simulations require the ability to enforce cell polarity of unicellular organisms by modeling single cells as multiple lattice domains with varying adhesion levels. We extensively studied the role of polarity in cell aggregation [28]–[30] and implemented these approaches into the general CPM approach within COMPUCELL3D Myxobacteria simulations. We have tested these simulations of Myxobacteria by implementing polarity within the single cell, causing migration in a direction dependent upon the cell center of mass and its current orientation, using a linear best fit. Rod shape is preserved by a cylindrical constraint, and early experiments successfully show modeling of directional movement and preservation of shape for rod-like bacteria. The cellular-automaton approach Hösel and Liebscher propose for modeling the nutrient supply of biofilm units [31], and their Gibbsian-type probability for modeling cell interaction using an energy function and realization probability inversely proportional to the system
energy, would also translate easily into the COMPUCELL3D framework.

COMPUCELL3D can also work in tandem with other existing software frameworks treating subcellular and supercellular phenomena. For example, BioSPICE [32], [33] models dynamic cellular network functions. BioSPICE can clarify complex intracellular biochemical networks [34] to simulate cell division, circadian rhythms, bacterial sporulation, and gene transcription [35]. BioSPICE can provide initial models for COMPUCELL3D. CellO [36] is an object-oriented tool that can model apoptosis, induction, differentiation, and the cell cycle. 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. We can cross validate the results the results of COMPUCELL3D with CellO’s grid-independent method. NEURON [37], [38] provides a simulation environment for neuron modeling, specifically supporting complex cell membrane properties. NEURON can treat large groups of cells and connections and possesses a GUI with a CellBuilder that can create new models or modify existing models, a potentially useful tool to interface with COMPUCELL3D. E-Cell [39], [40] is an object-oriented software suite for analysis of large-scale biological interactions, including biological cells. 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 [41] for stochastic simulation of coupled chemical reactions. E-Cell 3 will be another useful tool for cross validation. Finally, Virtual Cell [42], [43] can model cellular physiology by allowing a user to define both biological models such as species, reactions, structures, and cell geometry, and mathematical models via a general purpose solver for steady and unsteady solutions of algebraic equations, including partial differential equations (PDEs) and ordinary differential equations (ODEs). Virtual Cell models can set up the initial state for a COMPUCELL3D simulation including cell positions and structure, and we can use Virtual Cell equation solvers to generate external chemical gradients.

In order to validate COMPUCELL3D, we have built a 3D model of skeletal pattern formation in the experimentally well-studied avian limb. Because the skeletal pattern first establishes as cartilage 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 developmental 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.

![Developmental timeline of chick limb skeletal pattern formation. Drawings show transverse sections of wing buds. For all panels proximal is left, distal right, anterior up and posterior down. From [7], with modifications.](image)

Hentschel et al.'s [3] recent model provides an experimentally-motivated “bare-bones” mechanism for the major features of limb skeletal pattern formation. In particular, the model proposes a core of cell interactions and a biochemical network based on interactions known to produce precartilage “condensations” (i.e., tight cell aggregates) in limb-cell cultures and in developing limbs. The core mechanism has RD form and governs the spatiotemporal evolution of the condensation pattern [44].

The dynamics of limb growth define in a natural fashion three experimentally-determined zones in the developing limb: an apical zone in which a reserve of cells remain unpatterned, an active zone in which cells rearrange and condense, and a frozen zone in which condensing cells differentiate into cartilage and cease rearranging (see Section V and [3]). Our model explicitly encodes neither the zonal organization of the developing limb nor the spatiotemporal development of the skeleton. They are, instead, emergent
“biological” properties (given the symmetry assumptions of [3]) of both the 2D continuum bare-bones mechanism and its more realistic embodiment in a COMPUCELL3D simulation (see Section V).

We establish an exterior chemical gradient by solving 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 suppresses the production, or downstream effects, of the activator [6], [46], [47]. 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:

\[
\frac{\partial c_a}{\partial t} = \gamma[(J_0 + J_a(c_a)\beta(c_a))R_0 - k_ac_ac_i] + b_a(c_a - c_as)^3 + (d_{ax} \frac{\partial^2 c_a}{\partial x^2} + d_{ay} \frac{\partial^2 c_a}{\partial y^2} + d_{az} \frac{\partial^2 c_a}{\partial z^2}),
\]

\[
\frac{\partial c_i}{\partial t} = \gamma[J_i(c_a)\beta(c_a)R_0 - k_ic_ac_i] + b_i(c_i - c_is)^3 + D(d_{ix} \frac{\partial^2 c_i}{\partial x^2} + d_{iy} \frac{\partial^2 c_i}{\partial y^2} + d_{iz} \frac{\partial^2 c_i}{\partial z^2}),
\]

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 implemented in COMPUCELL3D 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 [48], [49]. 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.

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, for a four-fold increase in computational speed and a ten-fold reduction in memory consumption compared to standard alternatives. Finally, we present our 3D avian limb-bud simulation with domain growth. We provide sample input files for this simulation, along with instructions
for running COMPUCELL3D, on the COMPUCELL website [50].

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 $\sigma$ if the site lies in cell $\sigma$. 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 2D cells and their 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.

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 up to fourth-neighbor pixels ($N_1, \ldots, N_4$).

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 [51] and Marée [52] 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.

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. $T$ defines the size of the typical fluctuation. We implement fluctuations using the Metropolis algorithm for Monte-Carlo Boltzmann dynamics. If a proposed change in lattice configuration (a change in the indices associated with the voxels of the lattice) produces a change in effective energy $\Delta E$, we use the acceptance function:

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

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.:

$$
E = E_{Contact} + E_{Volume} + E_{Chemical}.
$$

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, $\tau$ and $\tau'$. The equation for $E_{Contact}$ is:

$$
E_{Contact} = \sum_{(i,j,k),(i',j',k')_{\text{neighbors}}} J_{\tau(\sigma),\tau'(\sigma')} \cdot (1 - \delta(\sigma(i,j,k),\sigma'(i',j',k')))\),
$$

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 $\tau$ has a prescribed target volume $v(\sigma, \tau)$ and target surface area $s(\sigma, \tau)$. Translating actual volumes to CPM target volumes involves fixing the ratio between the CPM lattice size in pixels and the actual domain length. 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. $E_{\text{Volume}}$ enforces these targets by exacting an energy penalty for deviations. $E_{\text{Volume}}$ depends on four model parameters: volume elasticity, $\lambda$, target volume, $v_{\text{target}}(\sigma, \tau)$, membrane elasticity, $\lambda'$, and target surface area $s_{\text{target}}(\sigma, \tau)$:

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

We model cell growth by allowing the values of $v_{\text{target}}(\sigma, \tau)$ and $s_{\text{target}}(\sigma, \tau)$ to increase with time. Cell division occurs when the cell reaches a fixed, type-dependent volume. We model division by starting with a cell of average size, $v_{\text{target}} = v_{\text{target,average}}$, causing it to grow by gradually increasing $v_{\text{target}}$ to $2v_{\text{target,average}}$ and splitting the dividing cell into two cells, each with a new target volume $v_{\text{target}}/2$. One daughter cell assumes a new identity (a unique value of $\sigma$). 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 first mechanism is chemotaxis, the second is haptotaxis. A chemotaxis model requires a representation of the evolving and spatially varying chemical concentration field, and a model mechanism linking the field 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 molecule in extracellular space. An effective chemical potential, $\mu(\sigma)$ models chemotaxis or haptotaxis, to incorporate the effective chemical energy into the CPM energy formalism:

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

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 [47] 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 RD approach. For simplicity we assume isotropic diffusion (i.e., $d_j^i$ does not depend on $j$), so:

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

where $\vec{u} = (u_1, u_2)^T$ and $D = \text{diag}(d_1, d_2)$. 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 or 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 specific 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.

In developmental processes like the one under consideration, networks of autoregulatory transcription factors common to all cells at a particular stage form multistable, multistate dynamical systems based on their ability to switch arrays of cell-type-specific genes on and off [54].

Here, 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 V). 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 (See [55], [56] for formal issues involved in creating cell-type transition maps from continuum and discrete
approximations of complex gene regulatory networks). Other approaches to modeling gene regulatory networks are possible [57].

### III. Motivation Behind CompuCell3D

We originally developed a 2D engine for morphogenesis called CompuCell [58]. This work extends CompuCell to 3D, increasing the range and biological realism of CompuCell3D simulations. 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 [59], [60].

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. This technique eliminates memory allocation for potentially hundreds of thousands of pixels and is one of multiple techniques we use for careful memory management.

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 of 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, reducing the number of pages swapped-in when we reference a cell in the grid.

Certain programming languages (e.g., Fortran [61]) have built-in features for contiguous allocation that could benefit 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 predefined interface), so we implemented the back end of COMPUCELL3D in C++ with careful memory management and certain other techniques to improve flexibility.

Building on polymorphism to achieve flexibility, we can add new functionality to COMPUCELL3D using six different simulation objects, each with its own predefined interface:

1) **Energy function**: Computes energies used by the CPM. An example is $E_{\text{Contact}}$ (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 attempts $N$ 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. A sample 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 divides 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 the cell mitosis code 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.

Figure 3 describes the execution order of these simulation objects, which work in conjunction with a COMPUCELL3D XML input configuration file that 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. The benefit of a configuration file extensible by a single method comes with the inherent cross-platform compatibility of XML, increasing flexibility by permitting the use of COMPUCELL3D on machines running different operating systems.

Fig. 3. Pseudocode for the execution of simulation objects. Diagram by Chris Mueller

While this structure for adding new simulation objects coded in C++ and interfacing them to the framework through the configuration file 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 have interfaced COMPUCELL3D with BIOLOGO [62], an XML-based domain-specific language for morphogenesis. We designed the syntax of this language to be easy
to understand by researchers studying morphogenesis or the CPM. BIOLOGO allows users to extend COMPUCELL3D with customizable energy functions, chemical fields, and Cell Type Maps. BIOLOGO automatically generates plugins implemented as C++ code.

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 [63]. 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++ [64]. 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 specific 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 define the offset as the size (in bytes) of the total number of DCNs registered thus far. Figure 4 shows a schematic of the contiguous memory allocation for a cell, assuming 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 fit 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 Cells, 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 fit into one page. Thus data sets for Cells 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. 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. Each watcher defines a method field3DChange() which defines the appropriate actions to take after 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 Cells. 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 Cells 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 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
Algorithm 1 Pseudo-code for offset neighbor evaluation.

Neighbor Finder:

1) Pre-processing: Initialize $x := 0$ and $\text{neighbor\_array}$ to be empty.
   
   $\text{neighbor\_array}$ is an array of pairs of points and integer distances, and $n := 0$;

2) $\text{getNeighbor}(\text{int } n, \text{ double } &D)$
   
   a) while length of $\text{neighbor\_array} < n$
      
      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 $\text{neighbor\_array}$;

   b) $D := \text{neighbor\_array}[n].\text{distance}$;
   
   c) return $\text{neighbor\_array}[n].\text{point}$;

3) To look at level 1 neighbors, distance 1 from a point $P$:
   
   a) do
      
      i) $\text{neighbor} = \text{getNeighbor}(n, D) + P$;
      
      ii) ... Do something with neighbor ... 
      
      iii) $n := n + 1$;

   while $D <= 1$

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:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>Ratio A/B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution Time For 100 CPM Flips</td>
<td>1959 s</td>
<td>501 s</td>
<td>3.91</td>
</tr>
<tr>
<td>Memory Usage</td>
<td>70656 KB</td>
<td>6564 KB</td>
<td>10.76</td>
</tr>
</tbody>
</table>

Therefore even without 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 [65].

D. Cell Type Map

COMPUCELL3D includes a cellular automaton to implement a Cell Type Map. 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).
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. Boundary Conditions

COMPUCELL3D implements NoFlux and Periodic boundary conditions on each individual axis. Calculating neighboring pixels using NoFlux boundary conditions involves discarding pixels outside the lattice. If a neighbor pixel lies outside the axis using Periodic boundary conditions, modular arithmetic implements wraparound.

G. Flexibility

We can extend COMPUCELL3D by encapsulating new functionality (e.g., new energy functions, new cell-change watchers, or new rules for cell differentiation 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 [50] provides complete COMPUCELL3D configuration file syntax along with configuration files for this paper’s validation simulation and three others: a basic cell sorting [1], an avian limb with domain growth [23], and Dictyostelium discoideum [11].

COMPUCELL3D can also read a Potts Initial File (PIF) for initial cell positions. The format of the PIF is straightforward, with repeated lines of the form:

```
<cell #> <cell type> x1 x2 y1 y2 z1 z2
```

Resulting in a rectangular cell extending from \((x1, y1, z1)\) to \((x2, y2, z2)\). The same cell # can be used multiple times if a cell is not perfectly rectangular.

V. Validation Simulation

Chaturvedi et al. [66] and Izaguirre et al. [58] used the systems-biology approach of integrating discrete and continuum models for biological mechanisms to describe a reduced, 2D model of vertebrate limb development. Figure 5 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.
Here, we use discrete models to describe cell movement, division, and interactions, and differentiation from multipotent cells into specific cell types. We use the modified form of the equations of Hentschel et al. as mentioned extending the originally 2D RD model to 3D. The concentration of the activator 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 fibronectin. As soon as the TGF-β concentration at a grid pixel exceeds a threshold, the corresponding cell secretes fibronectin at that pixel location, at a user-specified 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 RD can occur, moves proximo-distally within the grid. Figure 6(a) shows pictorially a 2D example of one step of the growth algorithm starting from a uniform cell distribution.

3D simulations add biological realism. For example, the Apical Zone is crucial to the simulation, and allows cells to move perpendicularly to the proximo-distal direction. Without the Apical Zone, the noncondensing zone is not connected any more, and one part of the limb always grows faster than the rest, see Figure 7. For our 3D simulation, with or without the Apical Zone, the noncondensing zone is always connected.

Initially, cells are uniformly distributed in a user-specified 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 $\text{FGFThreshold}$ which sets the lower boundary of the Apical Zone. The $z$ value in the grid where the FGF concentration is equal to this threshold defines the lower boundary of the Apical Zone, and the $z$ value at which the FGF concentration is at its maximum defines 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 than or equal to $\text{FGFThreshold}$ (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 6(b)): 
is the z-coordinate of the lower boundary of the Apical Zone, $L_z$ the z-dimension of the grid itself, $F_i$ the FGFThreshold, and $z$ is the proximo-distal position.

TGF-$\beta$ also plays a role in our customized Cell Type Map. Cells can be of type NonCondensing or Condensing, with Condensing cells being more adhesive. When a cell outside the Apical Zone occupies a point in the grid where the TGF-$\beta$ 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 8 shows the state diagram for these cell types. Only Condensing cells have an energy term for haptotaxis to fibronectin (Eq. (5)).

\[
FGF(z) = \begin{cases} 
\frac{z-A_z}{L_z-A_z}(1 - F_i) + F_i, & z \geq A_z, \\
\frac{z}{A_z}(F_i - 0.1) + 0.1, & z < A_z. 
\end{cases} 
\]  

Fig. 8. Cell Type Map implemented using COMPUCELL3D for the sample simulation.

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

\[
Density = \frac{C}{T} \times 100,
\]
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 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 9 shows the domain growth algorithm in COMPUCELL3D.

![Growth algorithm](image)

**Fig. 9.** Growth algorithm.

Mitosis rates come from [67]. The volume of a mesenchymal cell of approximately $15 \mu m$ in diameter is $1.8 \times 10^3 \mu m^3$, and its surface area is approximately $1.8 \times 10^2 \mu m^2$. The simulation starts with 8 cells, and eventually grows to contain 13,902 cells.

Figure 10 shows our simulation of 3D limb growth and pattern formation at 2250 Monte Carlo Steps (MCS), 3250 MCS and 4250 MCS, visualized using Ogle [68]. We ran the simulation for 4250 MCS, which took 7 hours, 1 minute and 13 seconds on a 2.4 GHz Intel Xeon processor with 1 GB of RAM. We
show three screenshots, and have superimposed the Condensing cells (grey) on the grid containing all
the cells for clarity. NonCondensing cells are red. Note the Apical Zone (pink cells) is the distal end of
the limb, where no cell condensation can occur. Related studies we show that changes in the parameters
对应的 known genetic expressions in the core components lead to biologically observed outcomes,
such as fusion of skeletal elements (see Figure 2 in [23]).

Fig. 10. 3D chicken limb growth and patterning visualized with Ogle. On a visualization of all cells, we superimpose one showing only
Condensing cells (in grey) for clarity. Apical Zone cells are pink. No condensation occurs in the Apical Zone. NonCondensing cells
outside the apical zone are red.

VI. Conclusions

We have presented an extended 3D version of the software COMPUCELL. This version is faster
and uses much less memory. COMPUCELL3D is a useful tool to compute the time evolution of
differentiating cells in 3D space using a stochastic Cellular Potts Model and a Cell Type Map implemented
as a discrete cellular automaton. Continuum reaction-diffusion equations model chemical signalling.
We have validated COMPUCELL3D by simulating the skeletal pattern formation in the avian limb. Our RD equations model a core cellular-biochemical network involving an activator (TGF-β), an inhibitor, the ECM protein Fibronectin, and the FGF growth factors. From this model, and using realistic spatial scales for our lattice, we observe the emerging spatiotemporal development of the skeleton. The Cell Type Map includes realistic biological information about chondrogenesis to model the transition from mesenchymal to condensing cells. Finally, the CPM uses relative cell adhesion parameters that can determine surface-tension experiments on different cell types. The literature provides volumes and mitosis rates. In related work we have shown how our model can predict skeletal element fusion.

We can extend COMPUCELL3D directly using C++ plugins, or more abstractly by describing arbitrary Cell Type Maps, chemical fields, and energy functions using BIOLOGO, an XML-based domain-specific language for morphogenesis.

We are currently extending COMPUCELL3D to integrate more realistic geometry, specifically irregular-shaped grids and moving boundaries. These involve more careful pixel and neighbor selection algorithms, which must discard pixels outside the moving boundary.

We are also creating a parallel version of COMPUCELL3D, which will more easily accommodate very large and long simulations. Parallelization is somewhat challenging because an event-driven, kinetic Monte Carlo process underlies COMPUCELL3D.

REFERENCES


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

**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-specific 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 occurring 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 Biology, 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. He is a fellow of the Institute of Physics.

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 from the National Science Foundation in 2001.