ISSN 1213-6972 Journal of WSCG http://www.wscg.eu/ Vol.26, 2018, No.1

A Dynamic Non-Manifold Mesh Data Structure to Represent Biological Materials

Endre Somogyi
Dept. of Intelligent Systems Engineering, Indiana University Bloomington, IN 47405
{somogyie} @ indiana.edu

ABSTRACT

Computational models of biological materials enable researchers to gain insight and make testable predictions of quantitative dynamic responses to stimuli. These models are particularly challenging to develop because biological materials are (1) highly heterogeneous containing both biological cells and complex substances such as extra-cellular medium, (2) undergo structural rearrangement (3) couple biological cells with their environment via chemical and mechanical processes. Existing numerical approaches excel at either describing biological cells or solids and fluids, but have difficulty integrating them into a single simulation approach. We present a novel dynamic non-manifold mesh data structure that naturally represents biological materials with coupled chemical and mechanical processes and structural rearrangement in a unified way.

Keywords
Physically Based Modeling, Biological Simulation, Dynamic Meshing, Finite Element Simulation.

1 INTRODUCTION

Researchers increasingly build computational models of biological materials to gain insight and make testable predictions about responses to stimuli. Mechanistic models of biological tissues are particularly challenging to develop because biological materials are highly heterogeneous across a broad range of scales. Biological cells exist in a dynamic, spatial fluid environment and create and respond to a range of physical and chemical stimuli with complex behaviors, including movement, changes in morphology and mechanics, proliferation, death, differentiation and modification of the local environment. Biological materials combine *active agents* such as biological cells with highly heterogeneous visco-elastic substances such as extra-cellular medium (ECM), fluids and solids. We use the term *physical agent* to refer to parcels of biological material. Physical agents may be active or passive, may or may not have sub-structures, and may or may not have natural boundaries.

As a key enabling component of a tool for modeling biological materials, we have generalized existing manifold mesh data structures into a novel dynamic **non-manifold** mesh data structure that can consistently represent smooth deformations and dynamic topological rearrangement. This mesh forms the basis of our *Mechanica* [13] environment for simulation of cells and tissues. Our mesh represents physical agents as polytopes in contact through shared surfaces, where both sides of the surface maintain their own identity. We perform smooth mesh deformations via the explicit finite element method; we use three mesh update operations, *radial edge split*, *radial edge collapse*, and *vertex split* to perform discrete topological changes.

Biological material models are diverse, so tools to build these models must be extremely flexible and able to easily accommodate new constructs. Computational biomaterial model development tools thus need to be able to conveniently represent the interactions and dynamics of a wide range of different agents with varied material properties. Most numerical methods naturally represent only a limited and fixed subset of agent behaviors. In engineered applications designed for modularity and testability, this specialization does not usually pose a problem. For example, simulations of mechanical parts such as an aircraft wing, usually need to account primarily for mechanical and thermal properties; traditional finite-element simulators suffice for such simulations. Even complex engineered systems, such as combustion dynamics tend to involve a limited number of pre-specified physical processes such as reaction-advection-diffusion, for which finite-volume simulation is appropriate. Biological materials often lack strict modularity, and tend to be highly interdependent with large numbers of coupled chemical and mechanical processes. Fig.1, illustrates cell and tissue rearrangement and dynamics that occur during body elongation (*epiboly*) in early embryonic development. Epiboly's com-

https://doi.org/10.24132/JWSCG.2018.26.1.3

plex dynamics requires coordinated material transfer and rearrangement, formation and loss of intercomponent boundaries and intra- and extra-cellular regulation of coupled chemical and mechanical processes.

While building computational models of the complex behaviors and interactions found in nature will inevitably be cognitively challenging, the lack of tools that describe naturally the mechanics of materials composed of continually rearranging agents creates an additional and unnecessary computational burden on model developers. Many computational-mechanics

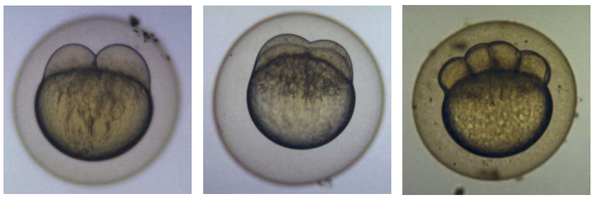
(a): 0.75 hours (b): 1 hours (c): 1.2 hours

Figure 1: **Cell rearrangement and identity change in embryonic development.** Time series of images $(0.8\,mm \times 0.8\,mm)$ of epiboly in a developing zebrafish embryo. During epiboly, ectodermal cells proliferate and flow from the dorsal to the ventral side of the embryo, as cell layers fold inwards and the yolk is incorporated into the embryo. Epiboly illustrate the complex interplay between biochemistry and mechanics during embryogenesis and the changes in agent number, shape, properties and relative positions which a tissue modeling platform must describe (from [4]).

simulation methodologies require the creation of meshes to approximate the shape and structure of physical agents. A *manifold mesh* is locally smooth and flat (homeomorphic to a disc), so a tessellated manifold surface mesh contains only edges of degree two; that is, exactly two faces share each edge. A *non-manifold mesh* is not restricted to locally-smooth surfaces, so a tessellated non-manifold surface can contains edges of degree one, two, three, or higher.

Existing dynamic manifold mesh data structures [10] often suffice to represent single cells in isolation, as long as their movement dynamics are not too complex. However biological tissues consist of large numbers of cells in contact, and these contacts continually form and disappear as cells rearrange, change shapes and adjacency, divide, merge and disappear. In *mesenchyme* (three dimensional connective tissue), both the topology and shape of cell-cell contacts can be complex. In *epithelia* (sheet-like tissues), cells contact each other via numerous locally flat surfaces, whose intersections often define geometric edges and vertices. In both cases, manifold mesh data structures are a poor match to the underlying physical reality.

When a biological cell contacts its neighboring cells and its environment Fig.1, its volume and surface both maintain their identities. With few exceptions, a membrane (and possible additional cell wall structures) separate cells from other cells and the surrounding environment. This membrane is thin relative to the size of the enclosed cell cytoplasm, with a typical size ratio of $\sim 2{,}000 : 1$. In many cases, when we are modeling cell and tissue-scale phenomena, we can approximate the membrane as a quasi-two-dimensional manifold surface embedded in three-dimensional space. The properties of living materials depend on the interplay of quasi-one-dimensional fibers (*e.g.*, in the extracellular matrix and intracellular cytoskeleton), quasi-two-dimensional sheets and membranes and fully three-dimensional structures (*e.g.*, biological cells, organelles, micelles or fluid droplets). Treating one- two- and three- dimensional agents consistently in the same mathematical and computational framework is challenging and defines the **aspect ratio problem**. This paper will focus on how our data structure represents the interaction of three-dimensional volumes with quasi-two-dimensional surfaces. We will discuss the use of the data structure to represent fibers in a subsequent paper.

In Fig.2, we categorize key identity changes that occur when physical agents consisting of a three-dimensional volume with a two-dimensional surface change adjacency. When two such initially separate agents contact each other, either: (1) the agents behave like water droplets: the contact surface between the agents disappears, while single volume and surface replace the original agents' volumes and remaining surfaces, (2) the agents behave like soap bubbles: the contact surface between the agents persists and the agents' volumes maintain their identity, but a single surface replaces the original agents' surfaces, or (3) the agents behave like biological cells: the contact surface between the agents persists and both agents maintain their volumes and surfaces. Traditional data structures usually naturally support only one of these adjacency-change processes (either 1) or 3)) and require awkward manipulations to implement the others. A single agent can also split partially or completely through the inverse of any of these processes. Our data structure enables us to efficiently represent all three types of adjacency-change.

2 RELATED WORK

Many numerical approaches address some of the computational challenges of representing biological materials, however no single existing approach can represent the variety of these materials in a self-consistent way.

The Finite Element Method (*FEM*) [14] is convenient for modeling solids under small deformations. The FEM discretizes materials into "elements" representing finite regions of space. An FEM solver generates

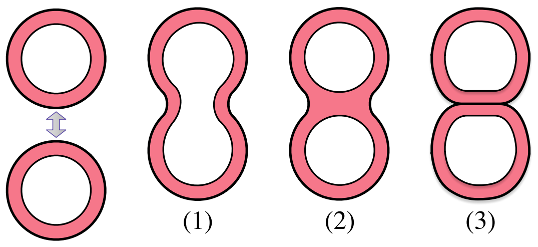

Figure 2: **Topological and identity changes on adjacency change.** When two physical agents come into contact: (1) They may merge both volumes and surfaces and lose their respective identities, like water droplets. (2) They may merge surfaces but maintain independent volumes, like soap bubbles. (3) They may maintain independent volumes and surfaces, like biological cells.

a set of nodes connecting adjacent elements, and a set of algebraic equations that define the time evolution of the nodal positions, velocities and properties at each node. Standard FEM discretizations can describe complex geometries and constitutive relations. However, most FEM methods can only naturally describe materials with a fixed dimensionality (one-, two or three dimensions). *E.g.*, to describe a composite agent with membrane and volume, standard FEM would represent the membrane using a very large number of small tetrahedra, which is computationally costly. Most FEM packages do not naturally handle large-scale material rearrangement or changing neighbor relationships.

The Finite Volume Method (*FVM*) [5] partitions space into a set of finite, connected volumes, and integrates governing equations for fluxes between volumes to calculate time evolution. FVM can represent fluid transport and reactions in complex geometries. However, most FVM approaches cannot naturally describe dynamic geometries or changing neighbor relationships.

Agent-based simulations of biological tissues typically employ lattice-, particle-, vertex-, or more recently FEM-based approaches to represent cells. Some of these approaches explicitly represent cell membranes, but many use *implicit* representations. Here we use the term *implicit* to mean data that is derived from explicit quantities. Because these are derived values, they do not have state variables, hence they can not define their own time evolution. Implicit surfaces have difficulty representing important biological processes such as surface chemistry, advection-transport, surface and edge contraction and adhesion, local signaling, etc. Explicit representation do not *ipso facto* mean that that they support these biological processes, but rather that they *can* support them.

Lattice based approaches have explicit volumes and can represent both implicit and explicit surfaces, but edges and vertices are implicit. The Cellular Potts Model (CPM) usually defines an implicit representation of membranes, edges and vertices between voxels of different cells. Volume advection can be challenging in CPM as well.

Particle based approaches include center models [5] and sub-cellular element models [11]. Center models treat cells as single point particles that interact via non-bonded forces, and define the boundary of a cell implicitly. Subcellular element models represent cells as collections of point particles and permit both explicit and implicit surface representations.

Vertex models [6] represent biological cells as connected, relatively simple ($\sim 6-15$) faceted convex polyhedra, with implicit surfaces, but explicit vertices and volumes. Modern FEM approaches [1] have explicit vertices, edges and surfaces, with volumes explicit or implicit depending on the representation. With few exceptions [9], most vertex or FEM type biological cell simulations are hard-coded to solve specific biological problems and are not available as simulation environments.

Surface Evolver [3] is a program which determines the minimal energy configurations of surfaces such as soap films. Surface Evolver represents surfaces using a dynamic non-manifold mesh. However it can only implement case (2) in Fig. 2 and cannot handle explicit membranes in contact. It also does not support descriptions of the complex biochemistry of biological cells.

While existing dynamic manifold mesh data structures [10] can conveniently represent individual physical agents and sets of agents with a limited number of contacts, they typically do not directly support the variety of identity changes which occur when agents change adjacency. Most do not explicitly track cell neighbor relationships and contact areas, but calculate them as needed, which can be slow. We previously developed a numerical simulation engine using the deformable manifold mesh from the Bullet Physics library. We found that this data structure was able to calculate deformations for at most 20 cells in contact, before performance dropped to unacceptable levels. Benchmarking showed that because all cells are in physical contact with each other, collision detection was computationally expensive. Furthermore, determining cell neighbor relationships and contact areas between cells (which Bullet Physics required us to at each time step) was computationally costly. Compute time using our non-manifold boundary representation mesh scales linearly with the number of vertices plus the number of triangles; essentially, performance is proportional to the total surface area rather than the total volume.

3 APPROACH

To represent and simulate physical agents, we must consider two related questions: (1) How do we represent the *structure* of these agents when the physical properties of the agent's constitutive elements may all differ? and (2) How do we represent *dynamics* of these physical agents including deformation and structural rearrangement.

3.1 Physical Structure

Traditional numerical approaches do not adequately address agents with changing adjacency. Tissues contain many cells (composite agents) which frequently change their adjacency, so we need to efficiently represent: (1) multiple agents in contact, (2) individual cell surface chemical process occurring on each cell's surface, and (3) chemical processes occurring between neighboring cells. Physical agents (fluid droplets, soap bubbles, biological cells and tissues) can have a well-defined boundary that has intrinsic material and chemical properties distinct from the agents they envelop. We represent biological materials with an explicit boundary, dynamic non-manifold mesh data structure, inspired by Hun and Lee's partial entity [8] and Weiler's radial edge [16] structures. This mesh data structure enables us to faithfully represent changing neighbor relationships and chemical and mechanical processes in a unified way.

Consider two biological cells are in contact as in Fig.2.3. When we take a section of this contacting region, we can make the abstraction that there are basically three kinds of physical materials here: (1) the membrane that belongs to the first cell, (2) the interstitial material between the cells, and (3) the membrane of the second cell. Each of these regions is thin relative to the size of the cells, but still has finite thickness. We represent this stack of physical materials with the *Triangle* data structure. The triangle itself is a composite type that can be thought of as a "sandwich" formed from a left *Partial Triangle*, the triangle itself, and a right partial triangle. The triangle represents a complete section of this contact region, where the partial triangles represent one side of a boundary, i.e. the biological cell's membrane.

Our mesh data structure consists of four key data types: vertices, partial triangles, triangles and cells. Vertices represent a position in space, maintain a list of incident triangles and cells, and they presently do not store any other state variables. We *measure* mass at each vertex as the barycentric area (1/3 the area) weighted sum of each incident triangle and partial triangle's mass.

The partial triangle Fig.3 represents one side of a physical boundary, i.e., a biological cell membrane. Each partial triangle belongs to the boundary of a specific cell, and the cell's boundary is defined as a set of connected partial triangles that form a closed manifold surface. Each partial triangle has pointers to its base triangle, the cell that it belongs to, its opposing partial triangle, and to its three neighboring partial triangles that are also part of the same cell's boundary. A partial triangle has an explicit mass, and can contain other state variables such as chemical amounts. The triangle data structure represents a section of shared boundary between physical objects, and is itself a composite structure of two partial triangles. Each triangle contains pointers to three vertices. Like the partial triangle, the triangle has an explicit mass, and can contain a vector of attached chemical amounts. We call the edge where two or more triangles intersect a *radial edge*.

A cell represents a closed physical region of space demarcated by an explicit boundary composed of partial triangles. A cell type can represent a physical agent, such as fluid droplets, fluid volumes, soap bubbles, or biological cells. A cell contains pointers to the partial triangles that comprise the cell's boundary. Presently, we approximate physical agents as homogeneous deformable solids, but we plan to add more complex internal structures in future versions. The cell has an explicit mass and can contain a vector of chemical amounts. Two cells in contact form a manifold surface, three or more cells can intersect at a radial edge as in Fig.3.b forming a non-manifold intersection between three or more surfaces. The partial triangles in a non-manifold intersection are adjacent on only one side of the triangles incident to the radial edge.

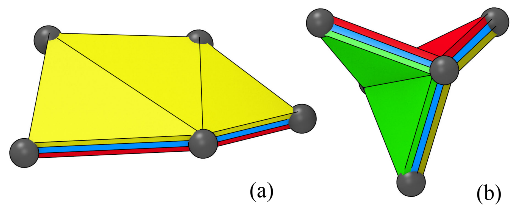

Figure 3: The partial triangle data structure enables us to represent both (a) manifold and (b) non-manifold meshes. (a) shows a manifold surface between a yellow and a red cell, (b) shows a non-manifold edge between a red, green and yellow cells.

3.2 Dynamics

The physical agents that we represent are not static – both their positions and their states evolve in time. They move around and past each other, and regularly come into and out of contact with each other (i.e., change neighbor relationships). We represent cell motion, membrane deformation, and chemical processes such as cell-signaling, membrane transport as continuous, ordinary differential equation defined processes. We implement large-scale structural rearrangement and topological change, such as large shape changes and cell attachment/detachment as discrete rule-based events. Vertices move according to the laws of classical mechanics, and we calculate their time evolution using the *Propagator* data type. Chemical reaction-transport processes are well-studied, and we re-use our existing solver [12] to implement them. The *Rules Engine* performs large scale rearrangement and topological transformations with rule triggered mesh reconnect operations. Mesh reconnect operations alter the underlying mesh, they define how elements split and merge - that is, they define local topological rearrangement. Mesh operations can create and delete mesh objects (triangles, and ultimately cells) and alter their neighbor pointers and vertex positions. Exactly when to apply a mesh update operation is highly material and simulation specific. Users need the ability to define custom triggering rules for mesh updates in order to model a range of different physical materials. Thus, we have developed a general rule-based mesh update system that separates the mesh update operation from the triggering rule, and enables the user to readily add new mesh update operations and to write custom, material-specific trigger rules.

The Propagator calculates the continuous-time evolution of the material vertices, as well as the fluid materials and state variables located on the triangle, partial triangle and cell elements. In other words, the propagator calculates the subsequent simulation state based on the current state. The propagator queries the model for the net force on each vertex and the rate of change of each fluid or other state variables associated with the cells and surfaces. The propagator presently uses a Runge-Kutta integrator, however we plan on adding more sophisticated integrators in future versions.

In order to enforce constraints, the propagator implements a Position Based Dynamics (PBD) constraint solver [2]. The PBD solver enforce constraints by adjusting the positions of the mesh vertices. The solver applies a correction displacement to the vertex positions. The basic idea of the PBD constraint solver is to first project all of the mesh vertices forward in unconstrained motion, according to the net force acting on each vertex. The unconstrained motion most likely violates some constraints. Each constraint object contains two functions: a) a constraint function of the current model state that yields a scalar constraint value, and b) a function that calculates the rate of change of the constraint function relative to the mesh vertices, i.e. the Jacobian of the constraint function. The constraint is satisfied when the constraint function evaluates to zero. The propagator sequentially adjusts the vertex positions in a Gauss-Siedel fashion, according the constraint Jacobian until the constraint function reaches a tolerance. We have found that volume constraints are satisfied with only 2-3 iterations.

The Rules Engine is responsible for discontinuous time state changes in the mesh and other state variables. The rules engine maintains a list of rules that associate a trigger condition and energy function with a discrete mesh update operation. The Rules Engine monitors the mesh for positional and topological changes and looks for mesh configuration patterns that match a trigger condition. When a configuration matches a trigger, the rules engine applies the corresponding mesh operation to the mesh, thus modifying the mesh. We present three mesh update operations: radial edge split, radial edge collapse, and vertex split. Each mesh update rule also defines an energy function. We associate an energy with each mesh topological configuration, analogous to the way a potential energy is associated with different configurations in physics. For example, users may wish to define a rule that penalizes excessively long edges, where the energy function could be proportional to the square of the edge length, say kx^2, and then associate the edge split operation with this energy function. Here, the rules engine might find an edge, and perform a trial edge split. The initial energy of this edge would be kx^2, and the energy of the split edge would be $2 * k(x/2)^2 = (1/2)kx^2$, thus the change in energy is $-k/2x^2$. The rules engine determines that this is an energetically favorable operation and applies it. The rules engine stores all triggered rules in a priority queue, ordered energy value, such that the most energetically favorable rules are evaluated first. This approach is similar to [6], where they perform triangle to edge and edge to triangle operations, and store all pending operations in a priority queue ordered on edge length. Because the rules engine successively applies rules sorted by energy level, each time a rule is evaluated, the rules engine looks at which mesh objects were altered by the rule and removes any pending operation that also depends on these objects. If that pending operation is still valid, it will be triggered and queued in the next time step. These rules enable us to to represent objects separating and rearranging, and we intend to add more rules as needed.

3.2.1 *Radial Edge Split*

The triangles in an evolving mesh can become too large to adequately represent an object's surface which can result in increased numerical error. In order to accurately sample and represent spatially variant physical quantities, we must refine large triangles. Triangle refinement replaces a single large triangle with two or more smaller triangles. Numerous approaches subdivide a single triangle into three triangles, but the more common approach is split a single triangle into two triangles. In a manifold mesh, when a single triangle is split in two, the neighboring triangle incident to the split edge must also be split, hence the name, "edge split". The radial edge split operation is a generalization of the commonly used manifold edge split, with the manifold edge split being a special case of the radial edge split. The radial edge split creates a new vertex at the midpoint of edge, and identifies all incident triangles around this edge, and splits each one of them into two triangles. The new radial triangles share the space as the original triangles did. Because the radial edge split does not move any existing vertices and only inserts a new vertex, the operation does not need to check for any topological or geometric violations - the radial edge split operation is always topologically valid. As such, it is one of the simplest mesh operation to implement.

For each triangle in a radial edge, the radial edge split operation identifies the outer-most vertex, and creates two new triangles and removes the original triangle. Each of the two new triangles share an edge formed by the new center vertex and the outer vertex. Radial edge split then reconnects these new triangle neighbor pointers to the triangles adjacent to the removed triangles.

3.2.2 Radial Edge Collapse

As a simulation evolves in time, a triangles can become excessively small. This over-refinement leads to wasted compute resources and performance degradation. The radial edge collapse operation removes small triangles and re-connects these triangles' neighbors. The radial edge collapse is a generalization of the conventional manifold edge collapse operation as studied by [15, 7], where the manifold edge collapse is a special case of the radial edge collapse. In a manifold edge collapse, an edge can be incident to exactly two triangles. A radial edge however can have any number of triangles arranged radially around an edge as in Fig.3. The idea however is the same: we want to remove the edge and re-connect all of the neighboring triangles. A variety of different trigger conditions may be appropriate for initiating a radial edge collapse. Say one would like to coarsen a mesh in areas of low curvature, or, say one wants to remove all triangles below a certain edge length threshold, as is the case in [6]

A radial edge collapse traverses every triangle in a radial edge and collapses the triangle along the edge side, and re-connects the neighboring triangles on the collapsed triangle's two remaining edges. The edge collapse removes a region of mesh which consists of the edge itself and its two incident triangles and enlarges the neighboring triangles to fill the void. The edge collapse operation reduces the dimensionality of a mesh. As such, it is only applicable under specific conditions. For an edge collapse (or any other operation) to be valid, it must not invert any triangles, i.e., it must not change any triangle's normal by more than 90 degrees. Vieira et al. [15] identified that a manifold edge collapse is valid if it does not violate the *link condition*. The link condition sates that an edge $e = \{u, v\}$ can be collapsed if and only if $link(u) \cap link(v) = link(e)$. In a triangular mesh, the star of a vertex v is the set of triangles and edges that are incident to v. The link of a vertex is the frontier of the star. The frontier of the star is the set of vertices that are incident to every edge and triangle in the star, not including v. The links of the edge and each vertex for non-manifold meshes are slightly more complex to calculate than the manifold case. We first build the edge link by iterating over every triangle in the radial edge and inserting the outer vertices into the edge link set. Because we only need to test that the intersection of the u and v link sets are equal to the edge link set, we do not need to build the entire link set for each vertex. Rather, we only need to build the $link(u)$ set out of the vertices adjacent to u, that are not in the $link(e)$ set. If we find a vertex adjacent to the other edge vertex v that is not in $link(e)$ and not in the $link(u)$, the radial edge violates the link condition.

In a radial edge collapse, the link condition alone does not guarantee that a vertex move will not invert a triangle. We must also explicitly test triangle incident to each of the radial edge endpoints. We test each of these triangles by test-moving the edge endpoint vertex into a trial position and testing for a change in the triangle normal direction. Presently, we also test to ensure that a radial edge collapse will not collapse a tetrahedron down to a triangle. We check this by looking at the partial triangles on each face of a radial edge triangle. If that partial triangle's two outer neighbors are adjacent to each other, then this partial triangle and its two neighbors define three faces of a tetrahedron, with an open base. The collapse of this triangle will result its two neighboring triangles collapsing down to each other - that is, collapsing a tetrahedron to a triangle.

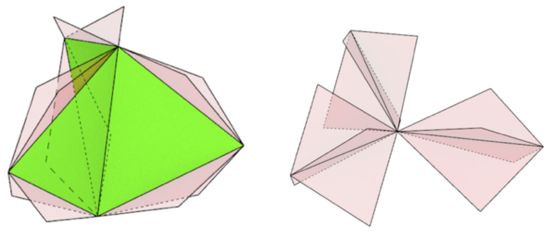
Figure 4: The radial edge collapse operation collapses a set of radial triangles (green) down to a single vertex, pulls in the remaining triangles (pink) to fill the newly created hole, and connects each remaining triangle to maintain mesh connectivity.

3.2.3 Vertex Split

The cell valence count of a vertex is the number of cells that share that vertex. The previously described radial edge collapse removes a vertex from a mesh and attaches all of this vertex's cells to another vertex, thus potentially increasing the cell valence count of the remaining vertex. Many mesh generation packages, tend to produce meshes that are predominantly pentahedron and hexahedron, thus contain a large number of vertices with eight cell valences. Eight-valence vertices are unusual in nature, and high or unbalanced vertex cell valences also tend increase computational cost and numerical error. The vertex split operation here is a non-manifold generalization of Hoppe's [7] manifold vertex split. The vertex split operation here splits a single vertex into two vertices and detaches a cell from a vertex in order to reduce vertex cell valence count. We can see in in Fig. 5 an example of a vertex with a high cell valence count, (five in this case), and relaxed configuration after five vertex splits. The vertex split operation enables cells to reconnect to each other. In a manifold mesh, a vertex split is the exact inverse of the edge collapse operation. In our non-manifold mesh, the vertex split operation shares a similar relationship with the radial edge collapse, though they are not exact inverses of each other.

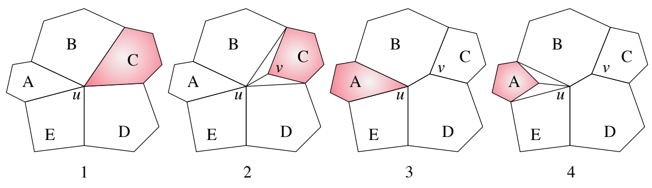

Figure 5: A sequence of vertex split operations in 2D. The central vertex initially has a cell valence count of 5, and the C cell (in red) is selected for ejection. The vertex split locates the shared face between C and its neighboring cells, creates new vertex towards the center of C, pulls the neighboring face of C to this new vertex, and creates a set of triangles to fill this void. The original vertex now has a cell valence count of 4. The vertex split then selects the A cell for ejection, creates a new vertex, moves the shared face, and fills the void again. The final images show the mesh after two subsequent vertex split operations. Note, the vertex split is a localized operation, the frontier elements of these cells are not modified.

A vertex split begins with a candidate vertex u and identifies a target cell C to disconnect and pull away from u. Vertex split will then split u into two vertices, u and v, where v is attached to the target cell, C, which is then no longer incident to u. As C is disconnected from u, u's cell valence count is reduced by one. The cell valence count of v is then the number of cells that that are both incident to the original vertex u, and adjacent to C, i.e. $valence\_count(v) = |incident\_cells(u) \cap adjacent\_cells(C)| + 1$. Thus, in order to reduce net valence counts, it is important for vertex split to choose C such that $valence\_count(v) < valence\_count(u)$. Otherwise, vertex split will indeed reduce the valence count of u, but it will also create another vertex v that has the same valence count that u originally had.

The most challenging task of the vertex split operation is filling the hole created by splitting and moving the vertex. This task is illustrated in Fig. 6. The vertex move creates a void between the cell being moved (C) and the set of cells adjacent to C, and incident to the original vertex u. In order to fill this void, the vertex split iterates around the triangle fan centered at v, composed of triangles that face C. Every triangle in this fan, by definition has exactly one face in the C cell surface. Consider a sequential pair of triangles in this fan. The non-C facing partial triangles either belong to a same cell, or they point to different cells. If both partial triangles point to the same cell on both sides, then there is nothing to do, these triangles simply shift slightly, as their center vertex gets attached to a new vertex v. If, however, there is a cell change, (i.e., the first triangle's non-C facing partial triangle points to a different cell than the second triangle's non-C facing partial triangle) then the vertex split inserts a new triangle at this edge. We say that triangle t_1 is incident to cells A and C, and triangle t_2 is incident to cells B and C. Vertex split then creates a new triangle, t_n with vertices v, u, and v_f, the vertex on the frontier of the fan that is incident to both t_1 and t_2. The t_n triangle faces cell A on one side, and cell B on the other. The vertex split will continue iterating around this fan until it encounters the starting triangle. Vertex split here creates a set of new triangles, one for each pair of triangles that have a cell change. All of these new triangles share the new $\{u,v\}$ edge, and when vertex split completes creating these new triangles, it then connects their partial triangles together appropriately. We can see that vertex split is essentially the inverse of the radial edge collapse, in that a radial edge collapse removes a set of triangles around a radial edge, and vertex split creates a set of triangles around a radial edge.

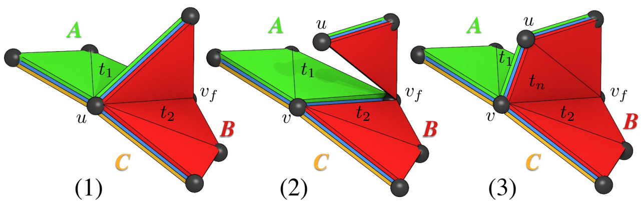

Figure 6: Three stages in a vertex split operation. This operation first identifies a target cell, C in yellow, and pulls this cell away, then splits the vertex u into u and v, and finally fills in the resulting holes with new triangles. Cell C is adjacent to cells A and B. Vertex split ensures that the partial triangles on the new triangles are connected to the correct cell boundaries.

3.3 Implementation Details

Biological cells and membranes participate in chemical processes that can often occur at significantly faster timescales than the motion of the objects themselves. In biological material simulations, the most computationally expensive task is evaluating these continuous time chemical reactions local to cells, surface elements and the extracellular milieu, as well as the chemical fluxes between neighboring spatial objects. The topological rearrangement operations previously discussed perform a significant number of mesh traversals when evaluating mesh operation rule triggers, and when performing the mesh update operations themselves. In order to maximize the performance of the chemical process solver and mesh traversals, we choose to explicitly store all neighbor relationships in the mesh at the expense of increased memory usage and software complexity.

We provide two basic mesh traversal operations: **radial edge** and **triangle fan** traversals. A radial edge is an edge connecting two or more incident triangles. To enumerate these incident triangles, the radial edge traversal (1) starts with a partial triangle and a pair of vertices that define an edge, (2) finds the next partial triangle that is both a neighbor of the current partial triangle, and incident to the edge, (3) identifies the opposite partial triangle (4) continues the iteration until it come across the starting partial triangle. A triangle fan is the set of triangles that are incident to both the same vertex and the same cell. (i.e., a triangle fan is a set of triangles, all of which face the boundary of a cell and surround a specified vertex.) The triangle fan traversal (1) starts with a triangle, vertex and cell, (2) follows the partial triangle neighbor pointers until the original triangle is encountered again.

4 RESULTS

We have created a set of simple models that demonstrate the mesh reconnect rules. These models read an initial configuration from a gmesh file. These models all implement a surface tension term (a force that acts in a direction tangent to the surface), and a volume constraint (acts perpendicular to the surface).

We use Zheng's method [17] to calculate the surface tension force. This method calculates the surface tension contribution of a triangle's three barycentric regions to each corresponding vertex. Each barycentric region pulls its' incident vertex in towards the triangle's barycenter. The magnitude of each Voronoi region's surface tension is proportional to the 1/2 the length of opposite edge. For details, see page 39 in [17].

We define the volume constraint on a per cell basis as the difference between a target volume and the present volume, $C(X) = \lambda(v_t - v)$, where v_t is the target volume, v is the current volume, and λ represents the stiffness of the constraint. Cells with a lower λ are "softer", as the constraint solver adjusts the cell positions more rapidly in cells with a larger λ. We approximate the volume constraint Jacobian, as in [2] as area weighted surface normal to each vertex. We use the barycentric area of each triangle to estimate the surface area of each vertex, and we compute the total volume of each cell using the divergence theorem.

To illustrate the effect of the edge collapse and vertex split operations, we performed an experiment with four cells and applied a harmonic bond force to the center of mass of two cells in order to bring the cells in contact depicted in Fig. 7. The initial condition contained the two red cells topologically connected with each other - they share triangles, and the two blue cells are disconnected.

We then apply a harmonic force to the center of mass of each blue cell which caused the blue cells to come closer, and start to push the red cells apart. We can see the edge length shrinking between the blue cells. When the edge length drops below a threshold, the radial edge collapse rule culls these short edges. This operation however increases the cell valence count on the central vertexes to four. The increased valence count causes the rules engine to invoke the vertex split operation. We specified a rule that causes the cell with the highest curvature to be ejected from the vertex. The vertex split operation splits these vertices and creates the new triangles in yellow. We can then see that as these two blue cells continue moving towards each other, these central edges continue to fall below the threshold, and continue to be culled, thus invoking further vertex splits. In the last frame, we can see that all of the shared contacts between the red cells has been eliminated. At this point, we remove the force between the two blue cells, and allow system to equilibrate. We can see that in the final frame, the material is topologically very distinct from the initial configuration. In the final frame, the blue cells are not connected and the red cells are disjoint. The mesh operations provide one possible way to maintain a memory on materials, as these are atomic operations that alter the mesh structure and topology.

To illustrate the effects of differential surface tension and demonstrate the mesh update operations when cells separate, we created a model that mimics a simple biological model of cellular mitosis. Here, we represent a biological cell that is about to undergo mitosis as a pair of Mechanica cells in contact, separated by a membrane. We first create two cubic cells in contact and set the surface tension of the membrane that separates these two cells to zero and set the surface-tension of the membrane not in contact to a positive value and allow the simulation to relax. As expected, the two cells in contact in a relaxed configuration form a nearly perfect sphere as in Fig. 8. Because the shared membrane is initially not under tension, it does not exert any force on the outside membranes. Once the system is relaxed (no net motion), we increase the surface tension on the shared membrane to a positive value. We can see that this shared membrane begins to pull in-

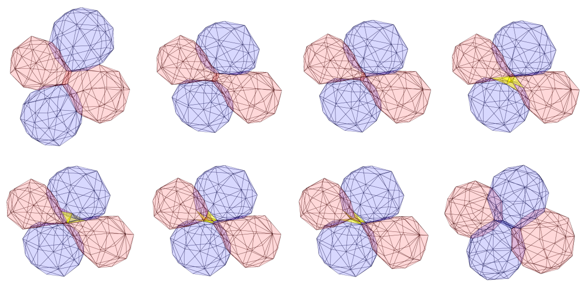
Figure 7: A series of frames from a simulation that starts with an initial configuration with the two red cells connected. We then apply a force between the two blue cells, which causes the blue cells to come towards each other and push the red cells out of the way. We can see the surface shrinking between the blue cells, as these edges shrink. The rules engine invokes the edge collapse operation to cull the short edges, which results in a number of vertices with a large cell valence count. The rules engine then invokes the vertex split operation to peel the red cells away. We then remove the harmonic bond between the blue cells, the system then relaxes on its own. The yellow regions indicate triangles that were generated by vertex split operations.

wards on the outer membrane, and the two Mechanica cells in contact begin to take the classic shape of a biological cell undergoing mitosis. We then increase the shared membrane surface tension to a value larger than the outer membrane surface tension. At this point, the shared membrane will continue pulling the outer membranes inwards, and shared contact area between the cells shrinks. As this shared area shrinks, the edge length of triangles within this shared area drop below the edge length cutoff, and the rules engine applies the radial edge collapse rule to cull these offending triangles. The shared surface area continues to shrink and pull inwards on the outer membranes until the shared area eventually disappears. The shared area disappears when the rules engine culls the last remaining triangle in that area. At this point, the two cells are no longer in contact, and their respective surface areas relax to approximate a sphere.

5 CONCLUSIONS

Researchers who simulate biological systems must take into account both mechanical and chemical processes. They must also represent objects that continually move and change neighbor relationships. Most existing numerical simulation approaches such as the finite element or the finite volume methods excel at representing certain aspects of physical objects. Finite element is ideal for simulating mechanical properties of materials with limited structural rearrangement. Finite volume methods excel at simulating complex fluids in relatively stationary geometries. Few, if any simulation platforms that can do both at once using the same simulation approach. The novel mesh data structure we have developed meets this challenge and enables us to represent a wide range of physical and biological materials in a unified way. As an example of the applicability of our platform, consider again the epiboly process depicted in Fig. 1. This biological process is challenging to represent using current simulation approaches because it involves a variety of different chemical and mechanical processes. Presently, researchers must resort to integrating a range of different numerical simulation methodologies and programs. Using our new numerical simulation engine, the representation of the epiboly process is much simpler, is done entirely self consistently, and requires no integration with external solvers. We are presently developing a cell-division mesh update operation that will split a single cell into two cells. When this new operation is ready, we will be able to directly represent this key biological process. Take, for example, Fig. 1. This image depicts a cluster of biological cells on top of a yolk sack. The cells start out as a blob on top of the yolk sack. This blob then gradually spreads out across the yolk, ultimately enveloping most of the yolk. We could represent each embryo cell as specific Mechanica cell type and represent the yolk as another cell type. We know that there are a number of chemical reaction processes that occur inside each embryonic cell. We presently support implementing chemical reaction networks inside our cell types. We also know that these embryo cells communicate with one another via intricate signaling pathways, and we implement these as chemical fluxes between cells. We also know that the embryo cells are initially blob-like, and later move to envelop the yolk. We can represent this transition with a surface tension

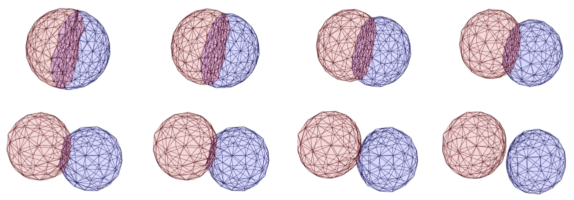
Figure 8: A simple cellular mitosis model. Two cells are initially in contact with a shared membrane, and each cell's individual membrane has a surface tension of σ, and the the shared membrane's net surface tension is zero. As the shared membrane is completely relaxed, the two cells in contact form a sphere. We then increase the surface tension of the membrane to a positive value which causes the membrane to shrink. As the membrane shrinks, the mesh update operations automatically cull the triangles with short edges and split large triangles. The mesh update operations enable the shared membrane to shrink and eventually disappear, at which point the cells separate.

like force between the cells and the yolk. Initially we could have a high surface tension between the embryo cells and yolk, and lower surface tension between embryo cells of the same type. Later, possibly based on some chemical state variable in each embryo cell, we could have the embryo-yolk surface tension reduce, and the embryo-embryo surface tension increase. Here, we would expect the embryo cells to then decrease their affinity towards each other, and their affinity towards the yolk to increase and spread out over the yolk. The embryo development process exemplifies how biological processes couple chemical and mechanical interactions - a feature that the novel numerical simulation engine we have developed uniquely captures, enabling researchers to model these kinds of processes. The mesh data structure presented here lays the groundwork for this numerical simulation engine.

Our simulation code is currently under active development, all source code and binaries *will be made* freely available on the Mechanica website, (`http://www.mechanica.org`) under a open source (GPL) license.

6 ACKNOWLEDGMENTS

We acknowledge generous financial support the National Institutes of Health, National Institute of General Medical Sciences, grant R01 GM122424, and National Science Foundation grant 1720625. We thank Dr. David Umulis, Dr. Marie Gingras, Dr. Amit Hagar and Dr. James P. Sluka for their discussion and insights.

7 REFERENCES

[1] S. Alt, P. Ganguly, and G. Salbreux. Vertex models: from cell mechanics to tissue morphogenesis. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 372(1720):20150520, May 2017.

[2] J. Bender, M. Müller, M. A. Otaduy, M. Teschner, and M. Macklin. A Survey on Position-Based Simulation Methods in Computer Graphics. In *Computer Graphics Forum*, pages 228–251, Sept. 2014.

[3] K. A. Brakke. The surface evolver. *Experimental Mathematics*, 1(2):141–165, 1992.

[4] T. Braunbeck and E. Lammer. Fish Embryo Toxicity Assays. Technical report, German Federal Environment Agency, 2006.

[5] A. Ghaffarizadeh, R. Heiland, S. H. Friedman, S. M. Mumenthaler, and P. Macklin. Physicell: An open source physics-based cell simulator for 3-d multicellular systems. *PLoS computational biology*, 14(2):e1005991, 2018.

[6] H. Honda, M. Tanemura, and T. Nagai. A three-dimensional vertex dynamics cell model of space-filling polyhedra simulating cell behavior in a cell aggregate. *Journal of Theoretical Biology*, 226(4):439–453, 2004.

[7] H. Hoppe. *Progressive meshes*. ACM, New York, New York, USA, Aug. 1996.

[8] S. H. Lee and K. Lee. Partial Entity Structure: A Compact Boundary Representation for Non-Manifold Geometric Modeling. *Journal of Computing and Information Science in Engineering*, 1(4):356–365, 2001.

[9] R. Merks, M. Guravage, and D. Inzé. VirtualLeaf: an open-source framework for cell-based modeling of plant tissue growth and development. *Plant physiology*, 155:656–666, 2011.

[10] T. Odaker, D. Kranzlmueller, and J. Volkert. View-dependent simplification using parallel half edge collapses. In *Proceedings of the WSCG*, pages 63–72, 2015.

[11] S. A. Sandersius, C. J. Weijer, and T. J. Newman. Emergent cell and tissue dynamics from subcellular modeling of active biomechanical processes. *Physical biology*, 8(4):045007, July 2011.

[12] E. T. Somogyi, J.-M. Bouteiller, J. A. Glazier, M. König, J. K. Medley, M. H. Swat, and H. M. Sauro. libRoadRunner: a high performance SBML simulation and analysis library. *Bioinformatics*, 31(20):3315–3321, June 2015.

[13] E. T. Somogyi and J. A. Glazier. A modeling and simulation language for biological cells with coupled mechanical and chemical processes. In *Symposium on Theory of Modeling Simulation*, Virginia Beach, Apr. 2017. Society for Computer Simulation International.

[14] M. Verschoor and A. C. Jalba. Elastically Deformable Models based on the Finite Element Method Accelerated on Graphics Hardware using CUDA. *Journal of WSCG*, 2012.

[15] A. W. Vieira, L. Velho, H. Lopes, G. Tavares, and T. Lewiner. Fast Stellar Mesh Simplification. *SIBGRAPI*, pages 27–34, 2003.

[16] K. Weiler. *The radial edge structure: a topological representation for non-manifold geometric boundary modeling* . Geometric modeling for CAD . . . , 1988.

[17] W. Zheng, B. Zhu, B. Kim, and R. Fedkiw. A new incompressibility discretization for a hybrid particle MAC grid representation with surface tension. *Journal of Computational Physics*, 280:96–142, Jan. 2015.