In Silico Reconstitution of Actin-Based Symmetry Breaking and Motility

Supplementary Material

Mark J Dayel[1] Orkun Akin[2] Mark Landeryou[3]
Viviana I Risca[4] Alex Mogilner[5] R. Dyche Mullins[6]

Note: Figures S1–S12 contain movies and 3D models that require a recent version of Adobe Acrobat and Quicktime to view (click on the movies to play and the 3D figures to rotate, zoom etc.).

Contents

[1]Miller Institute for Basic Research in Science, University of California Berkeley
[2]Department of Cellular and Molecular Pharmacology, University of California San Francisco
[3]Department of Mechanical Engineering, University College London
[4]Biophysics Graduate Group, University of California, Berkeley
[5]Departments of NPB and Mathematics, University of California Davis
[6]Department of Cellular and Molecular Pharmacology, University of California San Francisco

S1 Model parameter names

For aesthetic reasons, we refer to several model parameters in the text with subscripted letters. These correspond the simulation model parameters listed in table S1.

S2 Supplemental Movies

(removed to reduce file size)

Figure S1: *In vitro* symmetry breaking and motility for beads uniformly coated with ActA

Figure S3: 3D view of simulation showing links colored by tensile stress (color bar range represents zero to breakage stress)

Figure S7: Ellipsoidal beads break symmetry sideways

Figure S2: Computer simulation of symmetry breaking and motility (2D projections convolved with Gaussian)

Figure S4: Shell deformations during symmetry breaking, (left) circumferential and (right) radial

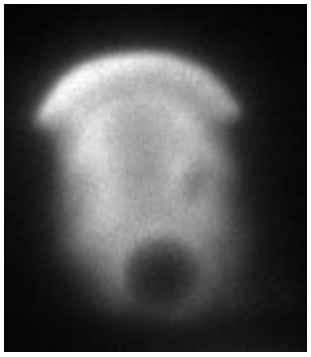
Figure S5: Strain buildup and release by link breakage. Images show node tracks, link breaks and transverse forces. Graph on right shows corresponding transverse and radial link force buildup and broken links as functions of distance from the surface of the bead.

Figure S6: Network deformations during smooth motion, (left) circumferential and (right) radial

Parameter in text	Parameter in Model	Description
P_{XL}	`P_XLINK`	Probability of forming crosslink
F_L	`LINK_FORCE`	Spring constant for node-node links
F_{BL}	`LINK_BREAKAGE_FORCE`	Force threshold above which node-node links break

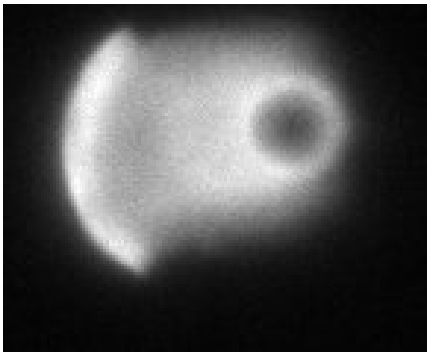
Table S1: Corresponding simulation parameter names in the main text and in the code

S3 3D reconstructions of beads

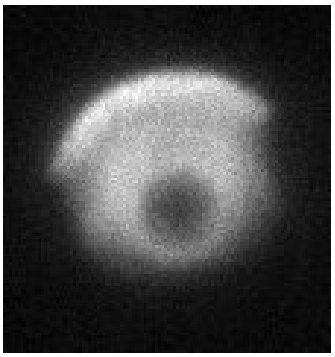


(3D interactive model)

(3D interactive model)

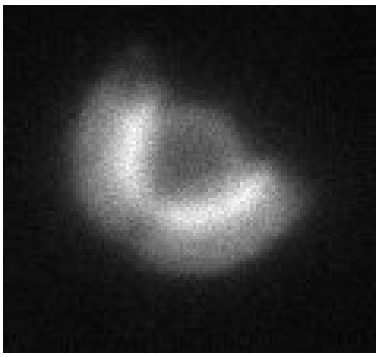


(3D interactive model)

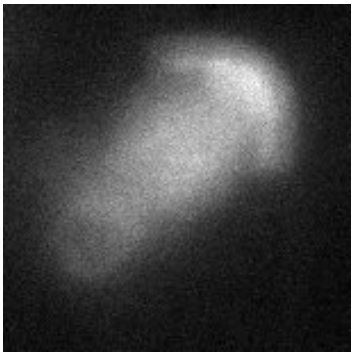


(3D interactive model)

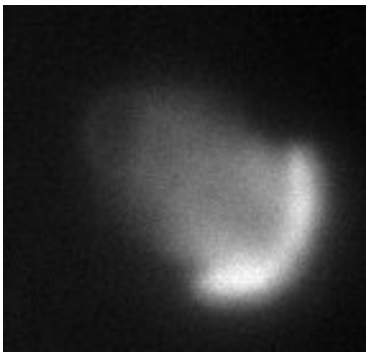
Figure S8: 2D projections (left) and corresponding 3D reconstructions (right) of constrained beads (5µm spacers) showing smooth opening of shell without bi-lobed structure



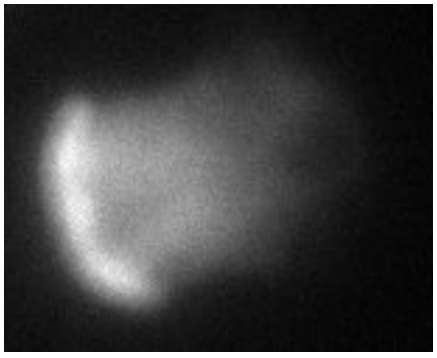
(3D interactive model)



(3D interactive model)

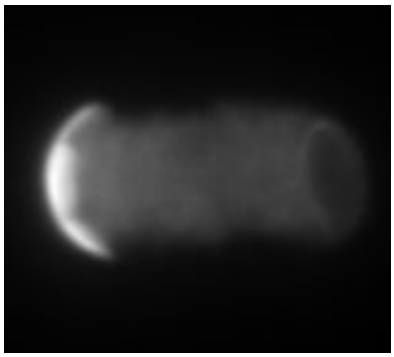


(3D interactive model)

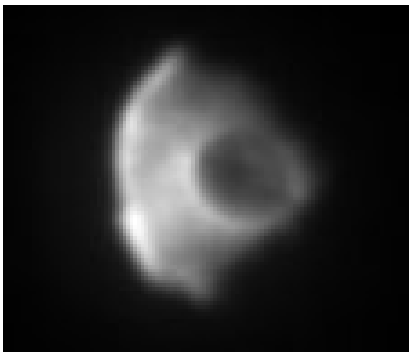


(3D interactive model)

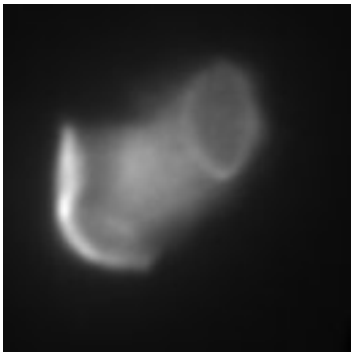
Figure S9: 2D projections (left) and corresponding 3D reconstructions (right) of unconstrained beads (15μm spacers) showing bi- and tri-lobed structure



(3D interactive model)



(3D interactive model)



(3D interactive model)

Figure S10: 2D projections (left) and corresponding 3D reconstructions (right) of shells and tails from unconstrained elliptical beads showing sideways symmetry breaking and motility

(3D interactive model)

(3D interactive model)

Figure S11: 3D reconstructions of *in silico* shells and tails from unconstrained elliptical beads showing linear crack and arc or bi-lobed structure

(3D interactive model)

Figure S12: 3D view of *in silico* network trajectory relative to bead during smooth motion

S4 Model Robustness

To determine how the model behaviors depend on the parameters, we took our default parameter set (section S7.2) and varied each parameter one by one. Figures S13 to S22 show the effect of varying the parameters, with images on the left showing timepoints during the run, and the corresponding bead velocities on the right (exact parameters are included in tiny writing below the left images). Some of the runs are cut short for some values of particular parameters (e.g. figure S16 when high LINK_BREAKAGE_FORCE) because the run essentially stalls.

Overall motility and pulsatile motion are extremely robust, but the smoothness of motion is fragile—changing many of the parameters will cause a transition to pulsatile motion.

Note, the automated algorithm for determining the symmetry break axis occasionally fails and projects the data orthogonal to the symmetry breaking axis (e.g. figure S22 FORCE_SCALE_FACT=0.536, in this case it is because the algorithm has picked up the axis from the second shell break at frame 200).

Some particularly interesting points to note:

S4.1 Increasing Radius produces pulsatile motion

Figure S13 shows that increasing the bead radius causes a transition from smooth to pulsatile motion, mimicking that seen in experiments (Bernheim-Groswasser et al. 2002). This run was performed with constant nucleator inertia (i.e. specifically not varying the inertia as a function of radius) to demonstrate that this transition is due to an effect of the radius of the bead on the network. At smaller bead radii, two things operate: First there the curvature is higher, so the network expansion is effectively faster (Bernheim-Groswasser et al. 2002), and secondly the ratio bead size to the network mesh size is smaller. This means that it is harder for tension to build up around the bead (through the effective mesh size of the network) because the bead can effectively go through the mesh. Another way to look at this is as analogous to reducing the probability of crosslinking P_XLINK. This produces smooth motion by increasing the effective mesh size when there are fewer links (few links, loose connections, larger effective mesh size) and the bead can move smoothly through the mesh. Reducing the radius does the same thing—the mesh size is the same, but now the smaller bead can move through it. Effective meshwork size is hard to control here, so it is difficult to tease out the relative contributions of these two factors on the transition to smooth motion for smaller beads.

S4.2 Shell thickness

The shell is relatively constant thickness for all parameters except for LINK_FORCE, i.e. the spring constant of the network (figure S17). When the spring constant is low, the network stretches a lot before offering a significant restoring force. Since this (circumferential) stretching is the cause of the symmetry break, decreasing the spring constant increases the thickness of the shell.

S4.3 Shell Flatness

Varying LINK_BREAKAGE_FORCE changes the force required to break links of the network. Figure S16 shows that for very low LINK_BREAKAGE_FORCE the network is incoherent, similar to the network with very few links (c.f. low values of P_XLINK). Unlike varying P_XLINK, high values of LINK_BREAKAGE_FORCE produce a very flat shell after symmetry breaking (looking at these results in 3D show the shell indeed to be planar). This supports the model of symmetry breaking: when the LINK_BREAKAGE_FORCE is very high, no outer shell links break except when the shell rips right through in the catastrophic symmetry break rip. When the shell relaxes, since no links broke in the outer shell, the equilibrium area of this outer shell is still exactly the same as the inner shell, so the shell relaxes to a flat plane.

S4.4 Pulsatile motion with no bead-network friction

Figure S23 demonstrates that increasing P_XLINK in the absence of any bead-network friction still induces a transition from smooth to pulsatile motion.

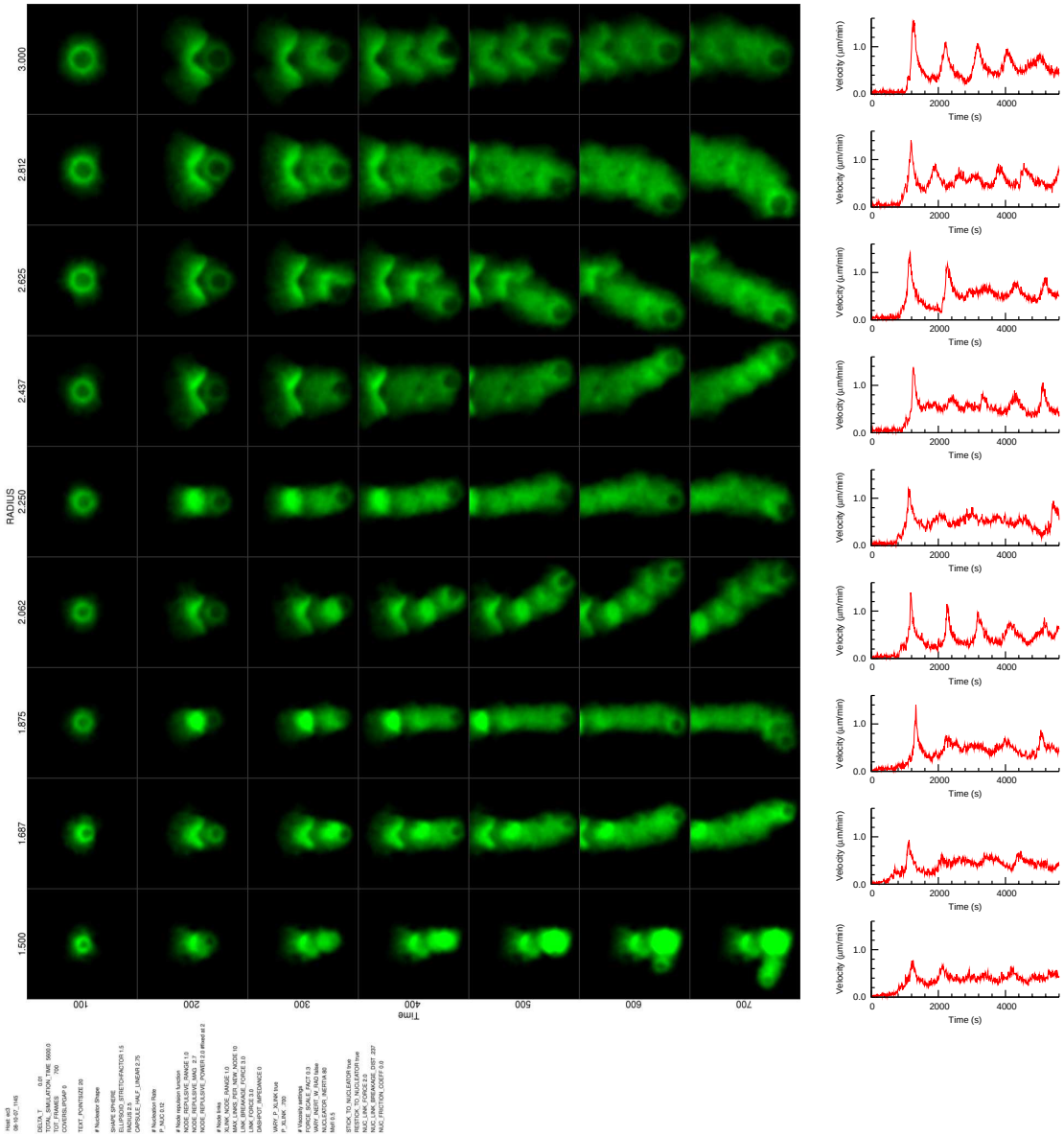


Figure S13: Effect of varying RADIUS

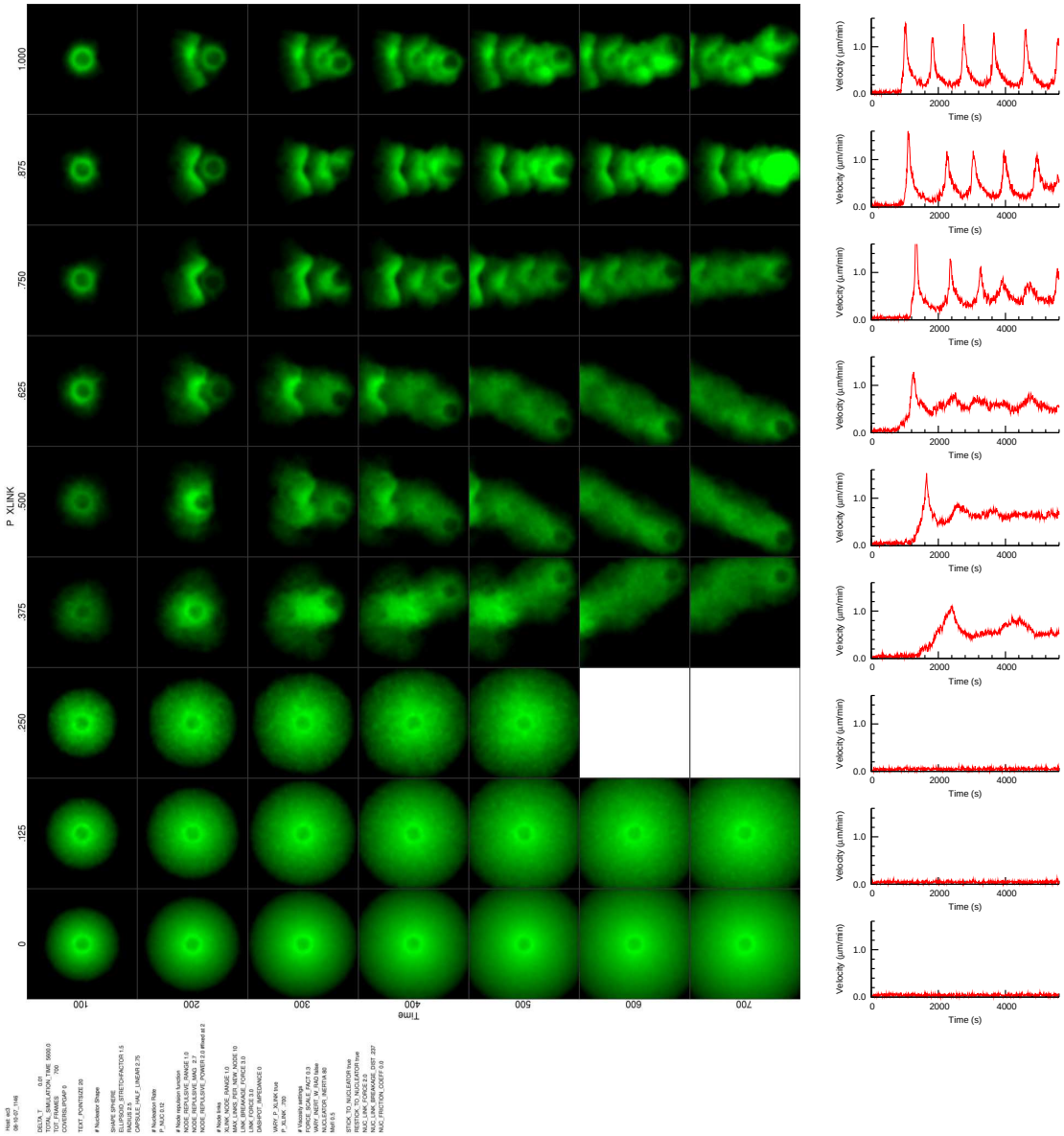


Figure S14: Effect of varying `P_XLINK`

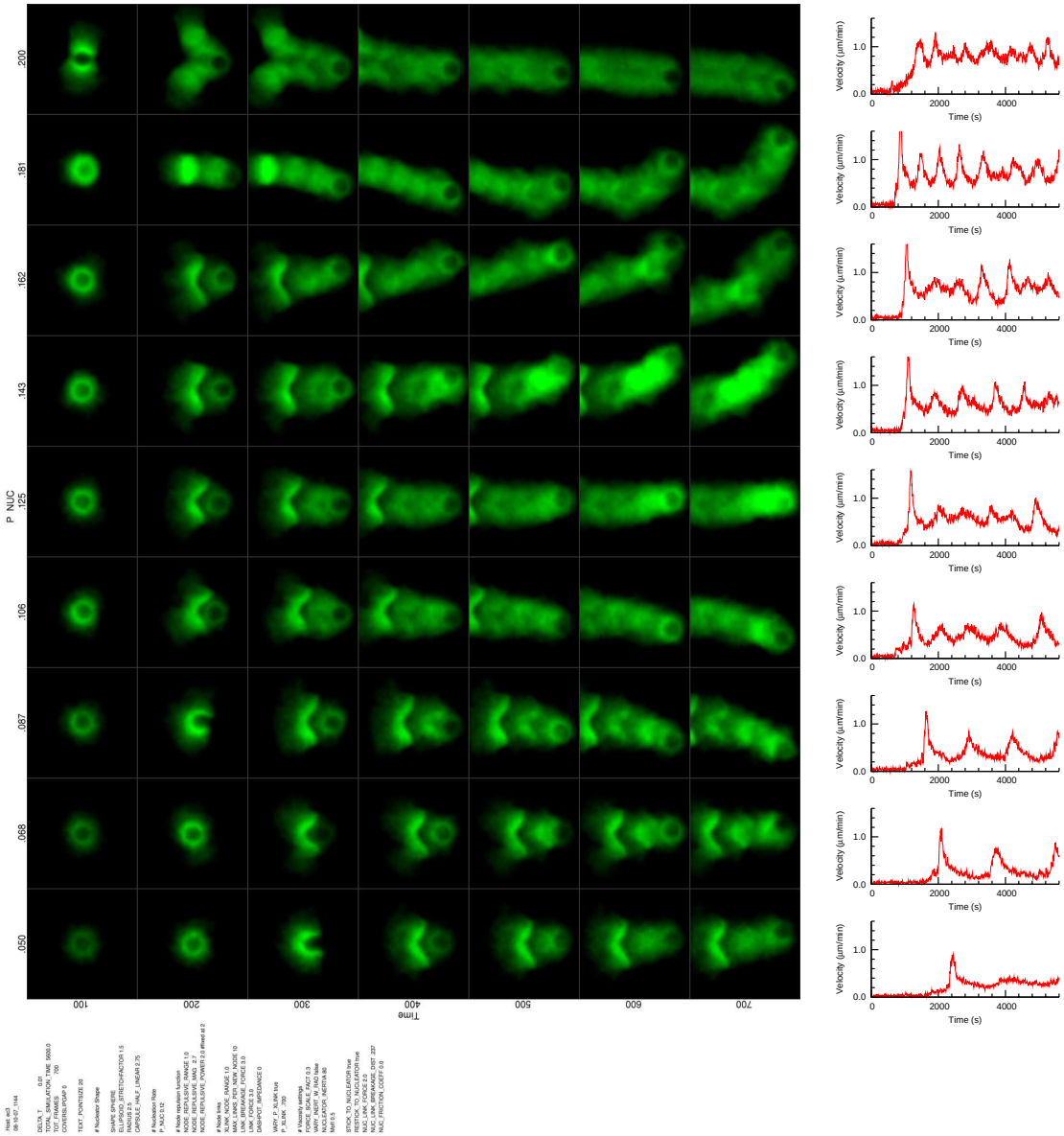


Figure S15: Effect of varying P_NUC

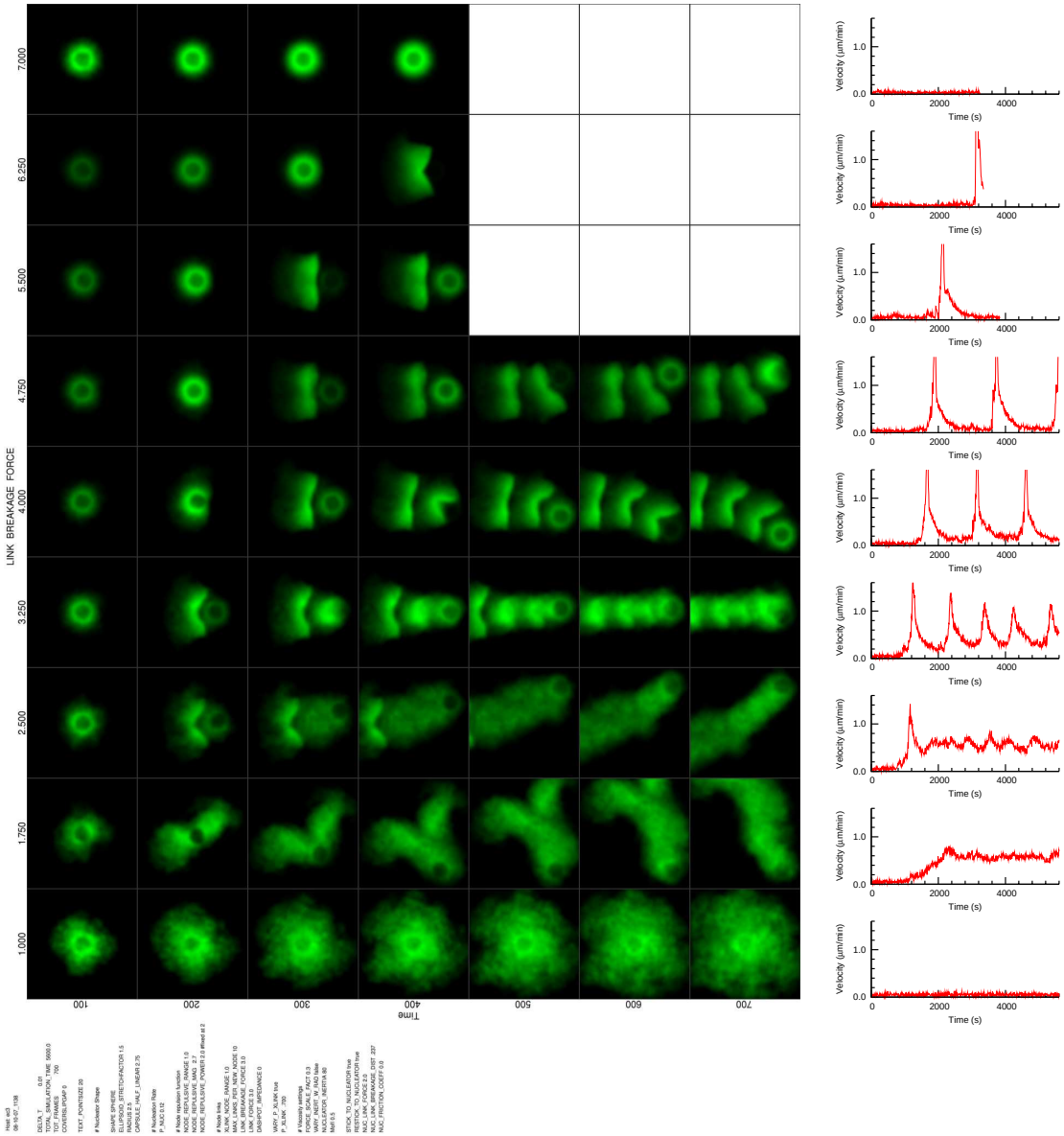


Figure S16: Effect of varying LINK_BREAKAGE_FORCE

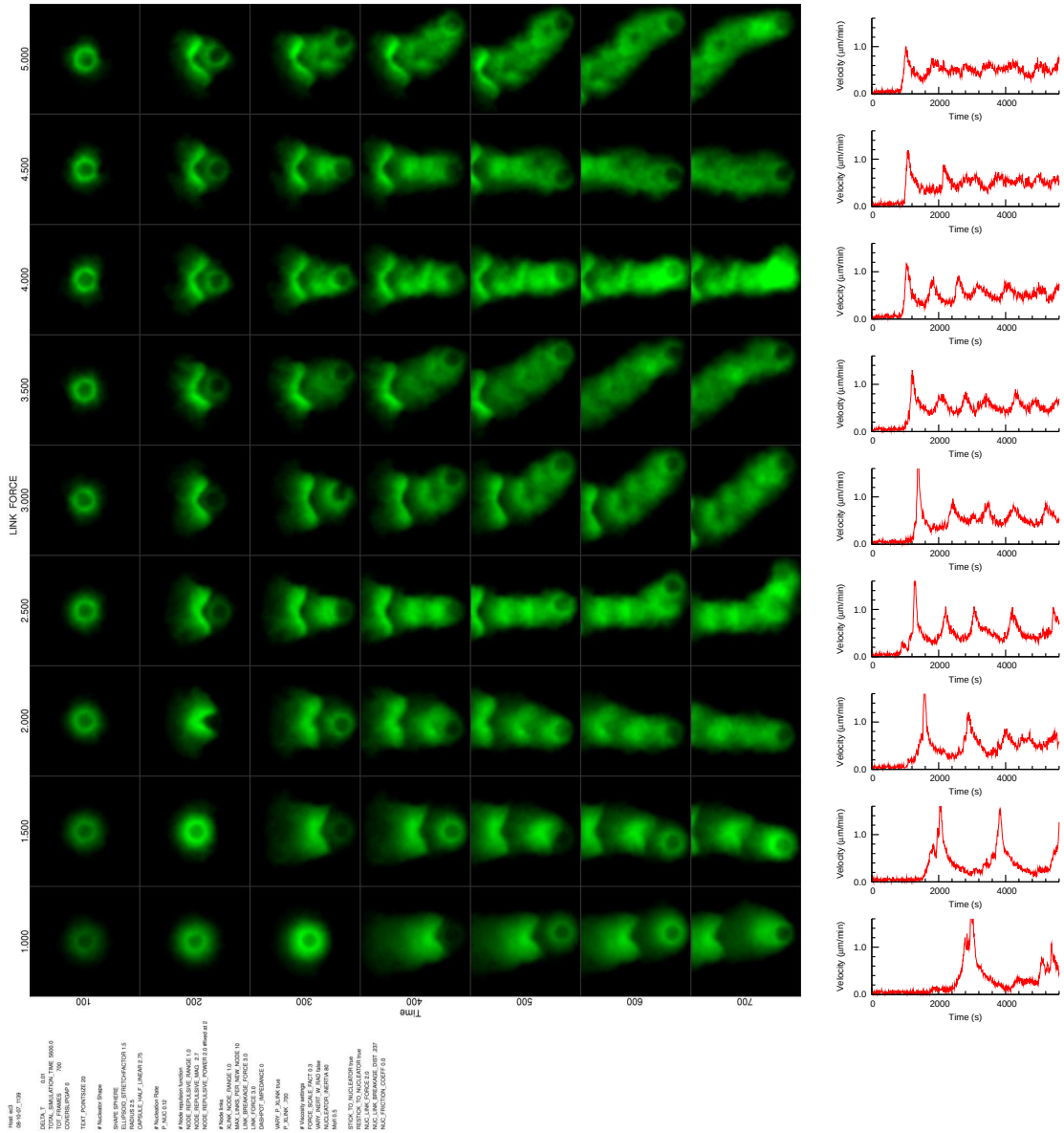


Figure S17: Effect of varying `LINK_FORCE`

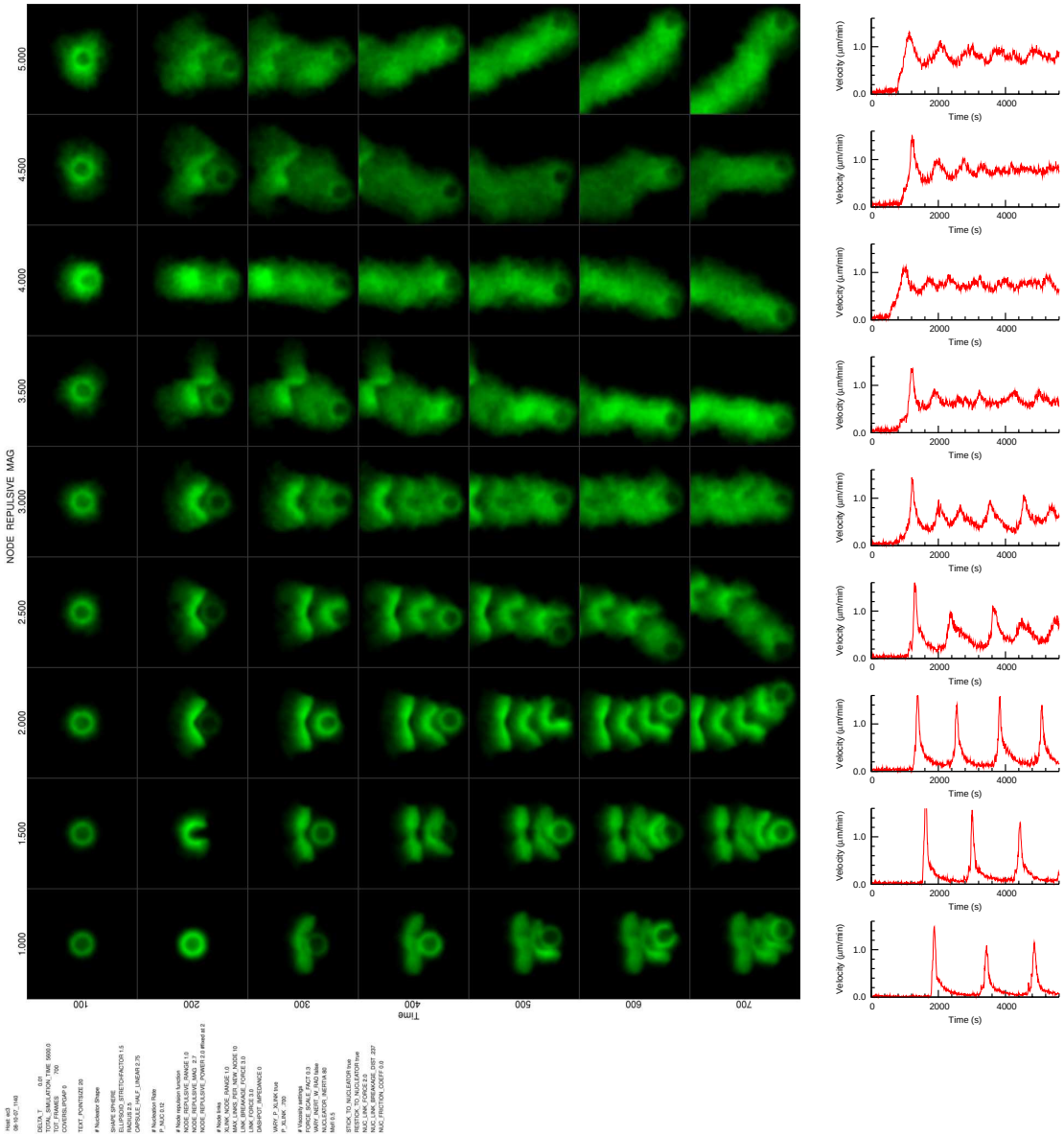


Figure S18: Effect of varying `NODE_REPULSIVE_MAG`

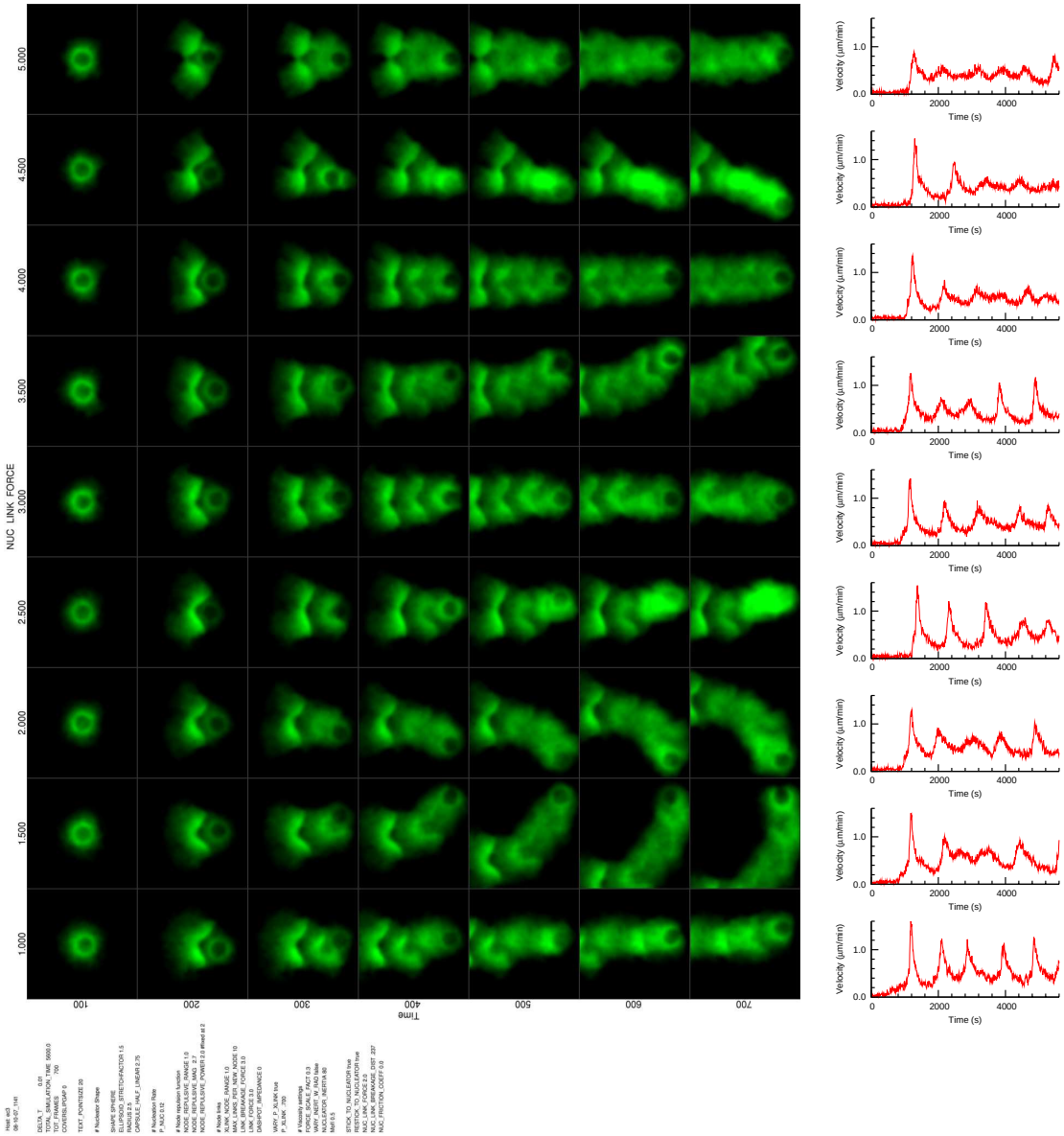


Figure S19: Effect of varying NUC_LINK_FORCE

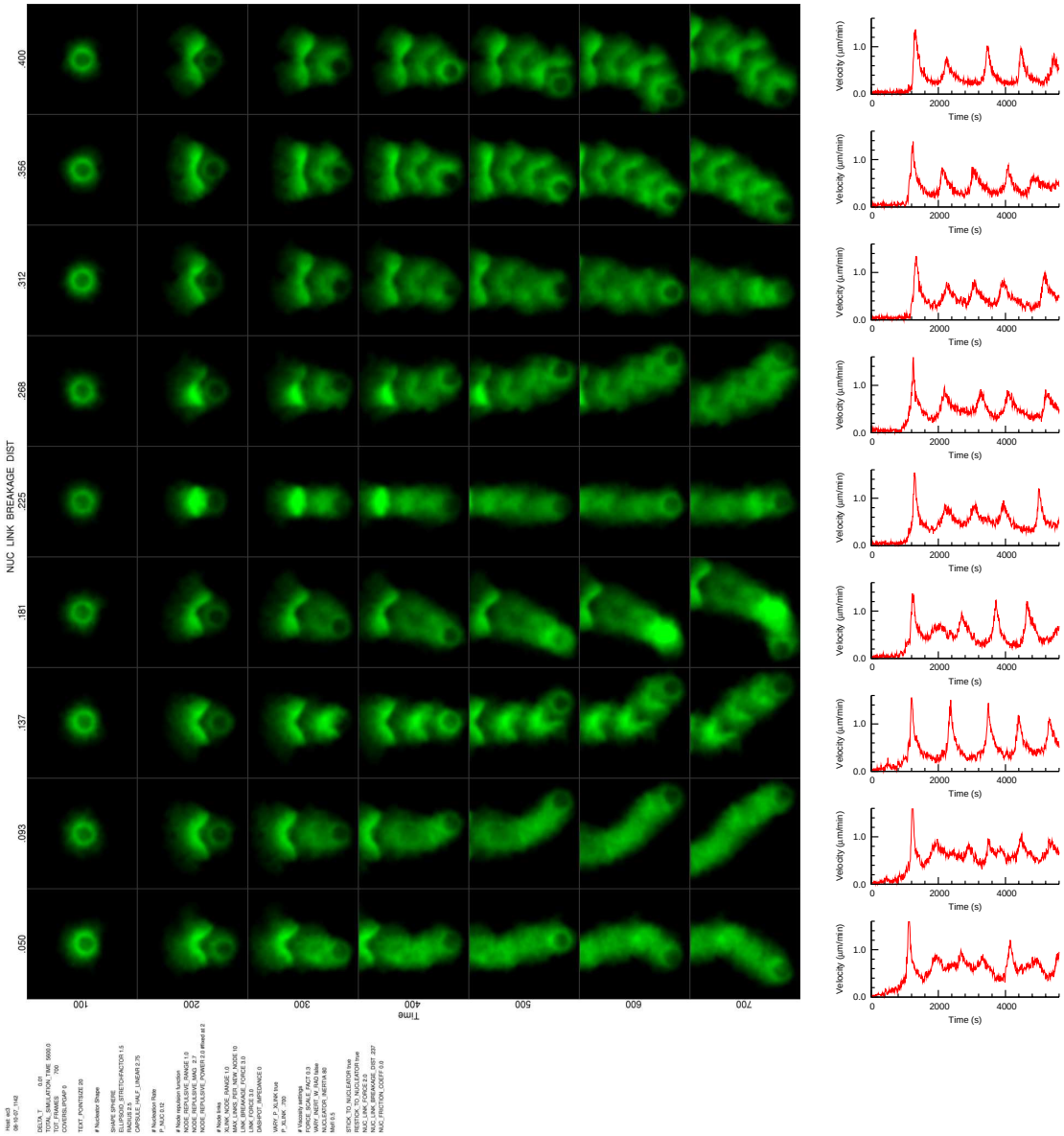


Figure S20: Effect of varying NUC_LINK_BREAKAGE_DIST

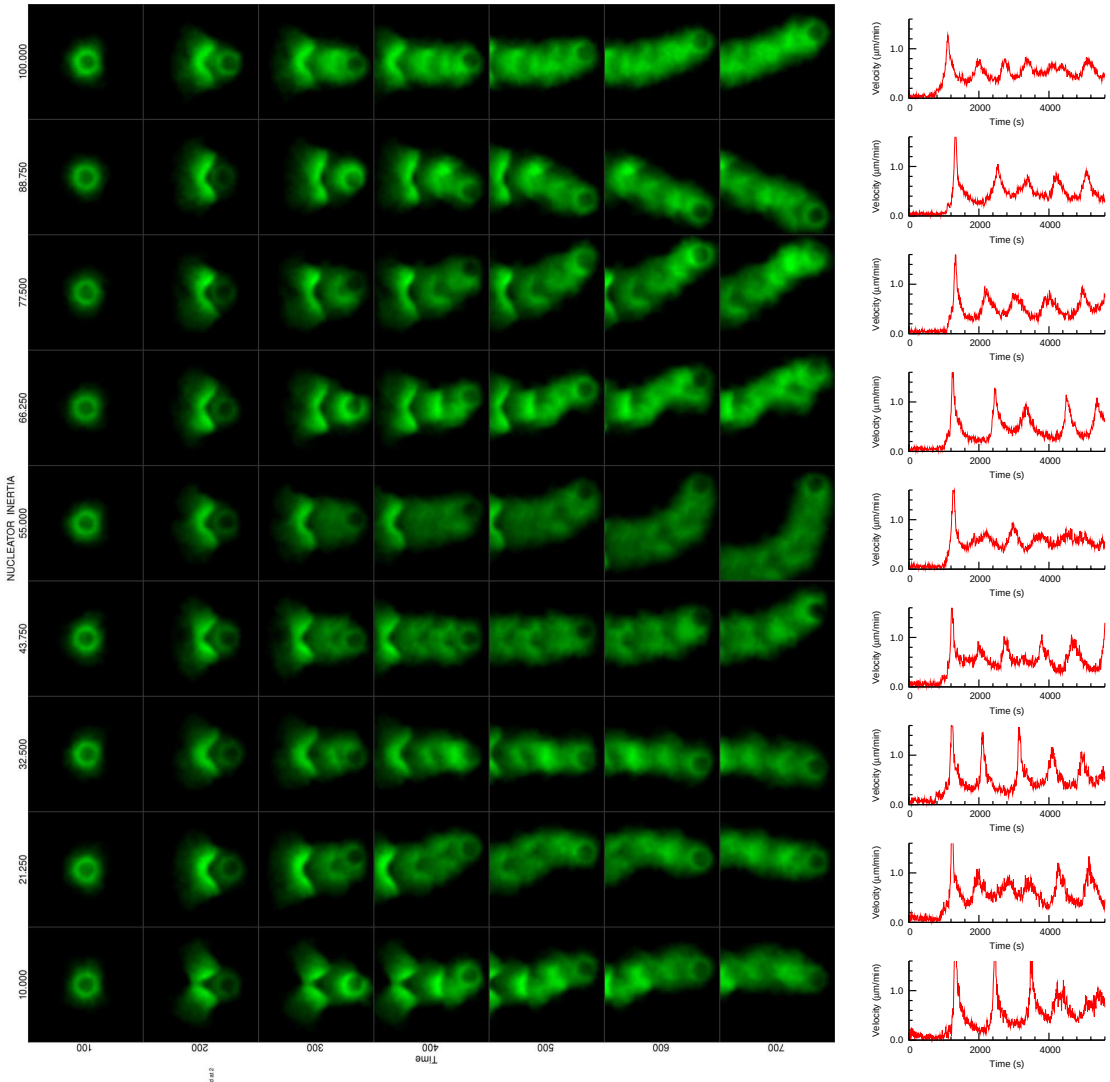


Figure S21: Effect of varying `NUCLEATOR_INERTIA`

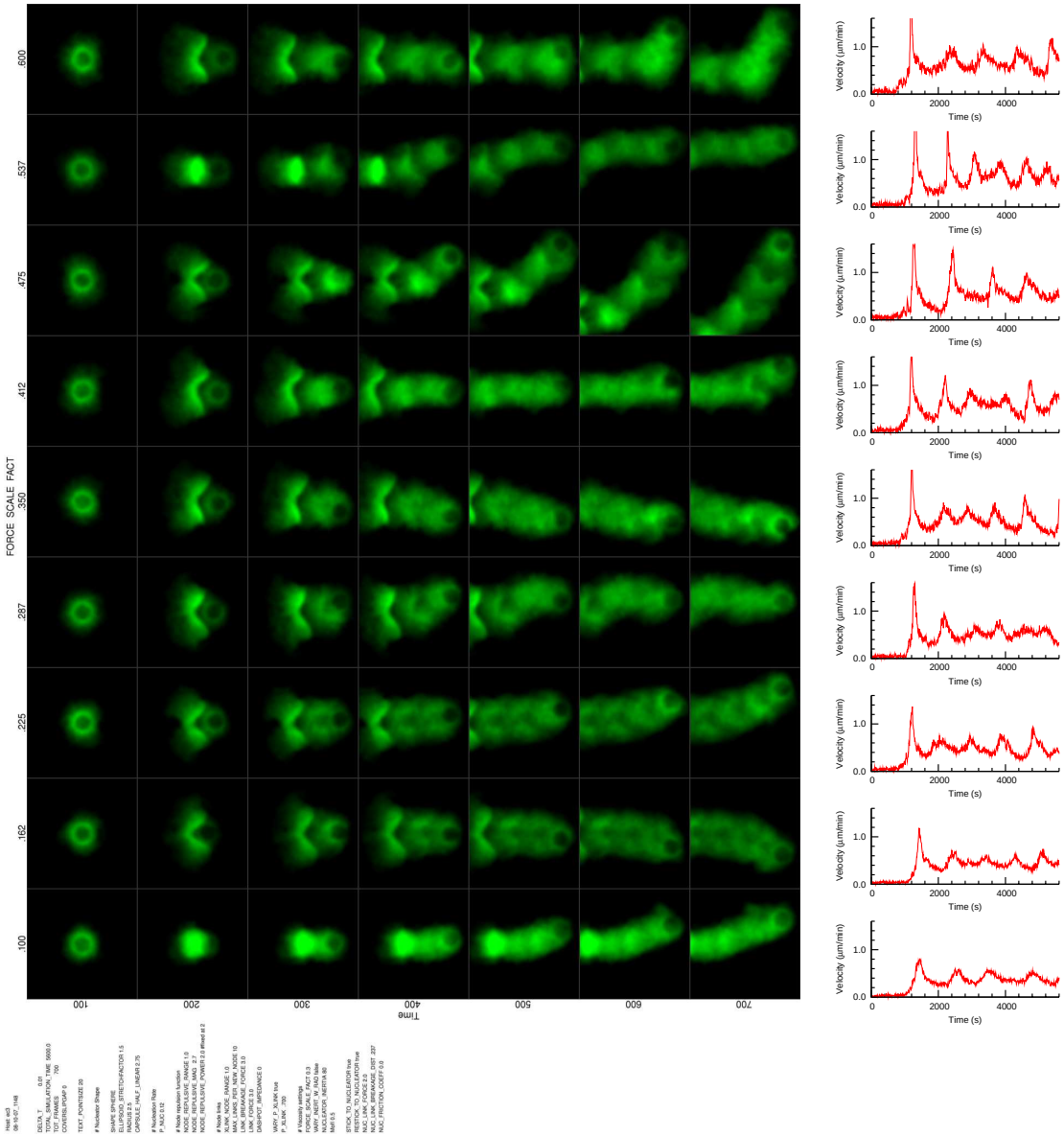


Figure S22: Effect of varying FORCE_SCALE_FACT

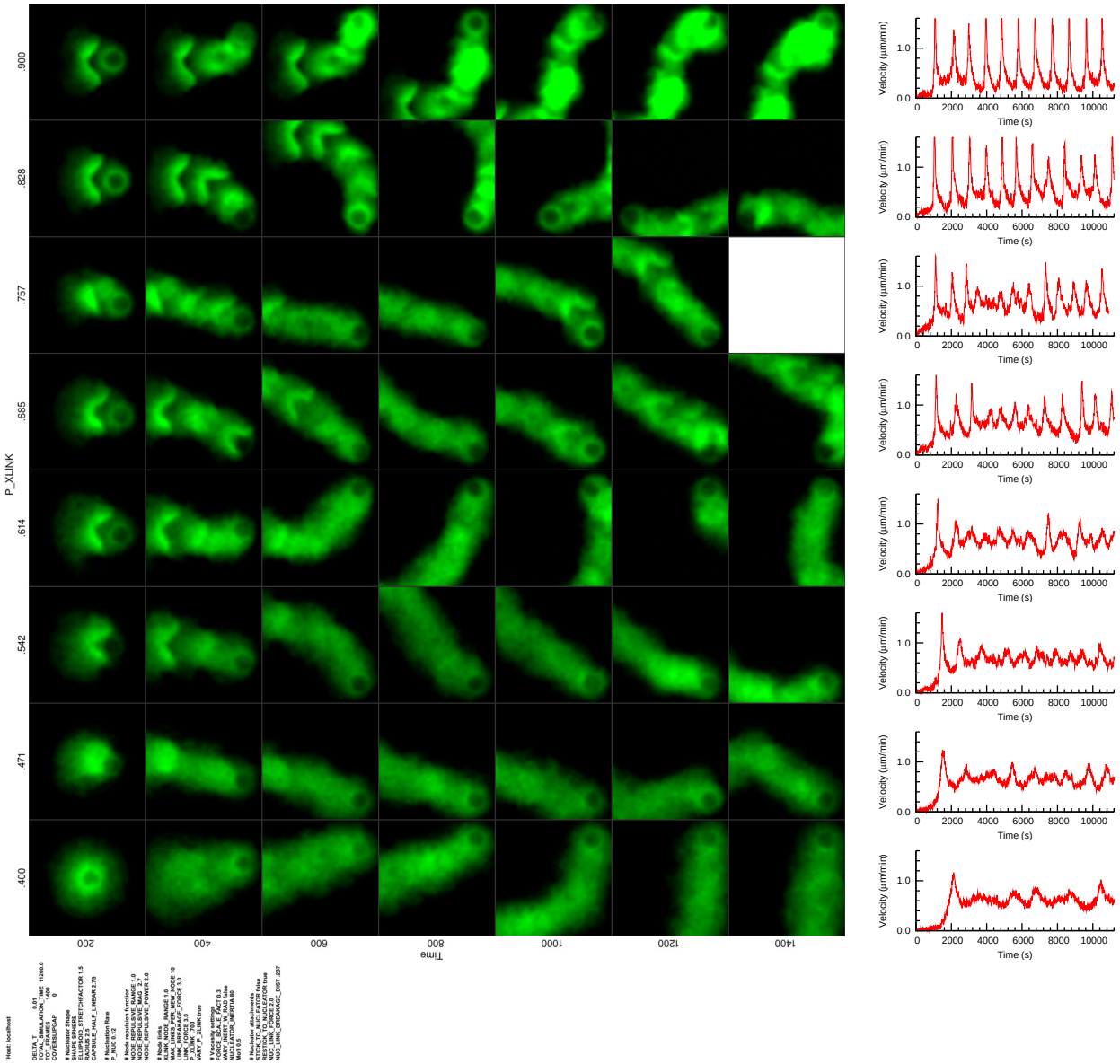


Figure S23: Effect of varying `P_XLINK` with no bead-network friction

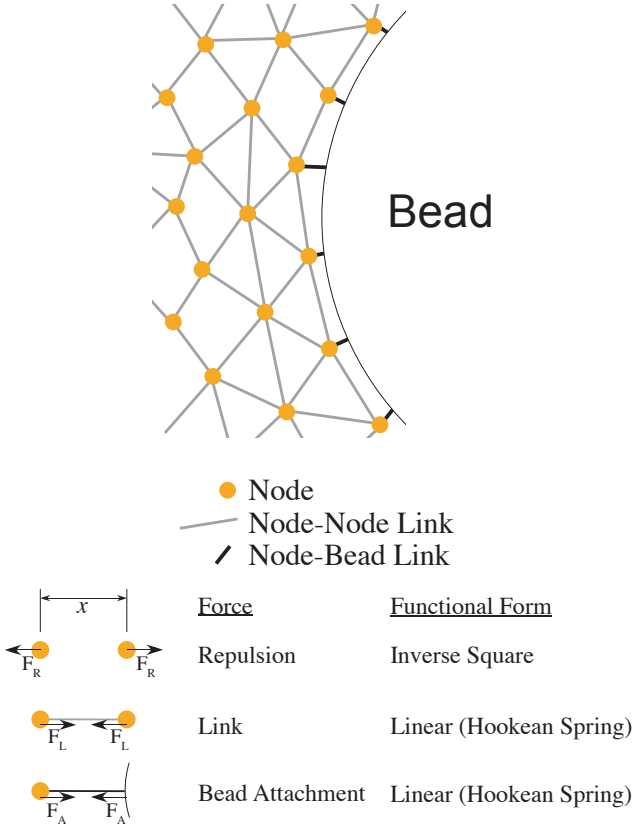


Figure S24: Diagram of network and functional forms of the forces

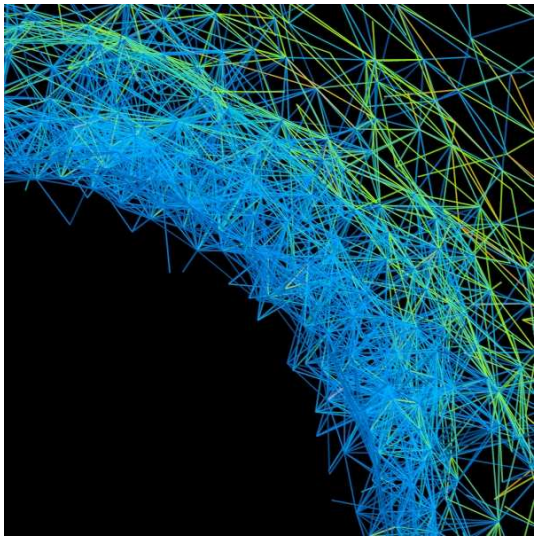


Figure S25: Cross-section of network showing links around bead (bead removed for clarity)

S5 Outline of the Model

The `comet` program is a Monte-Carlo/Lagrangian model that calculates the 3 dimensional positions of a large number of 'nodes' representing material in an actin network (diagrammed in figure S24 and an example shown in figure S25). For each timestep `DELTA_T`, nodes move a displacement proportional to the force acting upon them. There is no inertia, since this is a low Reynolds number regime. The forces acting on each node are as follows:

- Repulsive forces between nodes
- Link forces between nodes
- Link forces between node and nucleator

The nucleator object is treated as incompressible i.e. if during an iteration a node enters the nucleator, then in the next iteration it is simply moved out of the nucleator along a normal to the nucleator surface.

Nodes are nucleated at a constant rate, proportional to `P_NUC`, at the nucleator surface. To allow it to find an equilibrium position before being crosslinked into the network, a new node has its `harbinger` flag set when created, it experiences only repulsive forces for `CROSSLINKDELAY` iterations. Crosslinks are then formed as follows: All nodes with within `XLINK_NODE_RANGE` are counted, and links are either formed in random order, or if `XLINK_NEAREST` nearest first, until the number of crosslinks reaches `MAX_LINKS_PER_NODE`. Once a link is formed, its original distance is stored and used to calculate link forces. If the link is stretched or compressed away from its original length it behaves as a Hooke's Law spring and excerts a force proportional to, and opposing, the displacment. The scale multiple for this force is `LINK_FORCE`. This is to simulate an actin filament acting as an entropic spring by flexing motions. If the link force exceeds `LINK_BREAKAGE_FORCE` then the link breaks.

The nucleator is allowed to move and rotate, subject to displacement and torque vectors from the summed node repulsion from the nucleator, and the nucleator-node link forces. A full treatment of nucleator inertia is beyond the scope of the current model, and drag is simply scaled by a supplied parameter `NUCLEATOR_INERTIA` multiplied by the node inertia, and similarly the nucleator moment of inertia is scaled by the supplied parameter `MofI`. As a first approximation of how this should change with nucleator size, we scale the inertia and moment of inertia by the radius (or radius and length for long axes of the ellipsoids and capsules) if the `VARY_INERT_W_RAD` parameter is set. Given that this is not drag through a Newtonian fluid, but largely a product of complex fluid and network drag forces, this may not be very accurate. On the other hand figure S21 shows that the behavior is not very sensitive to the `NUCLEATOR_INERTIA` parameter anyway.

Output files are saved as jpgs for the x,y and z projections (convolved with a gaussian to make it look like a microscope image). Post processing routines can produce 3D rendering jpgs, or interactive 3D renderings on-screen. Also, post-processing 3D rendering of a single image will trigger the program to also write a vrml file to allow the 3D view to be imported into other software (e.g. Acrobat 3D etc.). Note: the program calls the Imagemagick `convert` program to add text to the images and save as jpgs and calls `bzip2` to compress the data files.

S6 Installing the program

The code is open source and available for download via svn (temporarily housed at `https://kinglab.berkeley.edu/svn/comet/model/comet/src/`). We provide a precompiled binary for Mac OS X `http://kinglab.berkeley.edu/public/mark/` (username:reviewers password:cometprogram), and instructions for compiling from source for OS X, Linux and Windows. The program requires that `ImageMagick` be installed for writing images (we recommenced using macports to install `ImageMagick` on OS X, and cygwin to install ImageMagick on windows.), and `bzip2` is required to compress the data files.

S6.1 Compiling from source

The code has two optional dependencies, the Gnu Scientific Library (GSL) which provides the Mersenne Twister random number generator (more statistically valid than the standard `rand()` function), and The Visualization Toolkit (VTK) which provides the 3D visualization routines. If these libraries are not available, you can compile without them by changing the `#define`'s `USE_GSL_RANDOM` and `LINK_VTK` in the file `stdafx.h` from 1 to `0` respectively.

S6.1.1 OS X

First install the Apple Developer Tools, then open the `Xcode` project file supplied. Include the GSL and VTK libraries in the search path, or disable before compiling (see above).

S6.1.2 Linux

A makefile is included for compilation with GNU Make. This should be edited to point to the GSL and VTK libraries, or disable them before compiling (see above).

S6.1.3 Windows

First install cygwin, then use cygwin to install ImageMagick, bzip2 and gcc, then compile as for Linux.

S7 Running the program

S7.1 Command line syntax

The program expects to be run from a new directory containing a copy of the control file `cometparams.ini`, an example of which is included in the source code and explained in detail below. Typing '`comet`' without any parameters returns the command line syntax:

For a new simulation setup the parameter file 'cometparams.ini' in current directory and type: `comet <numThreads>` where `<numThreads>` is the number of CPUs to use e.g. typing '`comet 4`' will start a new run using 4 simultaneous threads and parameters read from the `cometparams.ini` control file.

To process an existing dataset type: `comet <command> <frame range>` where `<command>` is 'post' to write bitmap images, 'vtk' to write 3D images or 'view' to enter 3D interactive mode. e.g `comet post 1:300` writes bitmaps for frame 1–300, `comet view 300:300` enters 3D interactive mode for frame 300 (the range '0:0' can be used to process all frames).

S7.2 The `cometparams.ini` parameter control file

Here are the core settings in the `cometparams.ini` file (explained below):

```
# Run time
DELTA_T                  0.01
TOTAL_SIMULATION_TIME    5600.0
TOT_FRAMES               700

# Nucleator
SHAPE                    SPHERE
ELLIPSOID_STRETCHFACTOR  1.5
RADIUS                   2.5
CAPSULE_HALF_LINEAR      2.75

# Nucleator attachments
STICK_TO_NUCLEATOR       true
RESTICK_TO_NUCLEATOR     true
NUC_LINK_FORCE           2.0
NUC_LINK_BREAKAGE_DIST   .237

# Node repulsion function
NODE_REPULSIVE_RANGE     1.0
NODE_REPULSIVE_MAG       2.7
NODE_REPULSIVE_POWER     2.0

# Node links
P_NUC                    0.12
XLINK_NODE_RANGE         1.0
MAX_LINKS_PER_NEW_NODE   10
LINK_BREAKAGE_FORCE      3.0
LINK_FORCE               3.0
P_XLINK                  .700
VARY_P_XLINK             true

# Drag
FORCE_SCALE_FACT         0.3
NUCLEATOR_INERTIA        80
MofI                     0.5
VARY_INERT_W_RAD         false
```

S7.2.1 Run Time

`TOTAL_SIMULATION_TIME` defines the run length in simulation time (uncalibrated, nominally seconds). `DELTA_T` defines the time step between iterations, i.e. for the given `TOTAL_SIMULATION_TIME` of 5600 and `DELTA_T` of 0.01, there will be a total of 560000 iterations. `TOT_FRAMES` defines

the number of snapshots to be taken during the run, i.e. 700 snapshots would mean one snapshot every 800 iterations.

S7.2.2 Nucleator

SHAPE can be SPHERE, CAPSULE or ELLIPSOID. For SPHERE, only the RADIUS matters. For CAPSULE, RADIUS and CAPSULE_HALF_LINEAR are used, and for ELLIPSOID, RADIUS and ELLIPSOID_STRETCHFACTOR define the shape. (Arbitrary shapes can be defined in the code, given a function that for a supplied point, returns a vector normal to the nearest point on the surface to the given point.)

S7.2.3 Nucleator attachments

When nodes are created, STICK_TO_NUCLEATOR defines whether they stick to their point of creation on the nucleator surface. Stuck nodes exert a force proportional to NUC_LINK_FORCE multiplied by the distance from the surface stuck point until they are extended beyond NUC_LINK_BREAKAGE_DIST when the link breaks. If RESTICK_TO_NUCLEATOR is true, *unstuck* nodes will re-stick if they come into contact with the surface again.

S7.2.4 Node repulsion function

The repulsion force between nodes is of the form:

$$F_R = M_R\left(\left(\frac{d_R}{d}\right)^{P_R} - 1\right), \quad 0 < d < d_R$$

where d is the distance between nodes, M_R (NODE_REPULSIVE_MAG) is a magnitude scale factor, and d_R (NODE_REPULSIVE_MAG) is maximum range of the repulsive force. The power factor P_R (NODE_REPULSIVE_POWER) is 2, so this is a simple inverse square repulsive force and is plotted in figure S26.

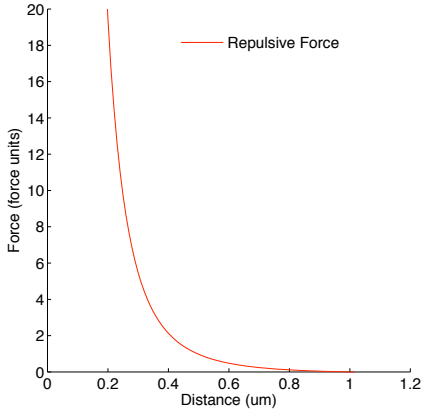


Figure S26: Repulsive force function

S7.2.5 Node links

P_NUC defines the rate of nucleation of new nodes per unit area per unit time. i.e. for one iteration, the number of new nodes added over the whole of the nucleator surface is P_NUC * DELTA_T * surf_area, where surf_area is in μm^2. The nodes are added at random positions on the surface, with an even distribution unless the ASYMMETRIC_NUCLEATION variable is set.

New nodes are crosslinked to nearby nodes within XLINK_NODE_RANGE. The links then behave as Hookean springs, exerting a restoring force

$$F_L = -M_L\left(\frac{d - d_L}{d_L}\right)$$

where d is the distance between nodes, M_L is a magnitude scale factor, and d_L is the original length of the link when it was formed (figure S25). If the link is extended so that its force goes beyond a certain limit, the link breaks. (optionally this can be strain rather than stress, i.e. a break occurs when $\frac{d}{d_L}$ exceeds a certain limit rather than when $\frac{d-d_L}{d_L}$ does)

Nodes are added to the surface and fixed there while their repulsive forces are ramped up linearly from 0 to full. This allows time for nodes already at the surface move and make room for the new node before it is crosslinked. The ramp-up occurs over CROSSLINKDELAY iterations. MAX_LINKS_PER_NEW_NODE limits the maximum number of crosslinks for each new node. LINK_FORCE is the spring constant, and when the extension forces reaches LINK_BREAKAGE_FORCE, the link breaks. P_XLINK is the probability of forming a crosslink to a node within range (still restricted by the MAX_LINKS_PER_NEW_NODE limit). The VARY_P_XLINK flag (normally on) also imposes a linear tail-off of this probability with distance. (see section S5 for more info).

S7.2.6 Drag

This section relates the forces to the actual movement of the nodes and nucleator. FORCE_SCALE_FACT scales the movement of nodes (i.e. effectively inverse of node drag). If you reduce this, you probably need to reduce DELTA_T as well. NUCLEATOR_INERTIA determines how hard it is to *displace* the nucleator and MofI determines how hard it is to rotate it. If VARY_INERT_W_RAD is set, inertia will be scaled by the size of the nucleator (see section S5 for more info).

S8 Implementation in C++

The code it's written in C++ for speed. We attempt to use an somewhat object-based approach, but a good many of the member variables are declared as static global to allow their access across threads.

Here is a breakdown of the main classes and functions in the program. There are numerous other functions but this is the core of the program:

- Main()
  - Spawns threads: `collisiondetectionthread`, `linkforcesthread` and `applyforcesthread` depending on the `USETHREAD_COLLISION`, `USETHREAD_LINKFORCES` and `USETHREAD_APPLYFORCES` parameters.
  - Parses the `comet_params.ini` file to read parameters. All of the parameters are implemented as globals (should fix at some point)
  - Creates the main `theactin` and `nuc_object` objects.
  - Runs through the main iteration loop, calling `theactin.iterate()` and saving snapshots every so often.
- Actin class
  - There is only one actin object, `theactin`, which constitutes the network, i.e. contains the nodes and the functions that deal with them.
  - The `iterate()` function does one iteration pass, calling:
    - `nucleator_node_interactions()` displaces any nodes out of the nucleator object along a normal to the nucleator surface
    - `nucleate()` adds new harbinger nodes to the surface of the nucleator
    - `crosslinknewnodes()` crosslinks harbingers once they are ready
    - `sortnodesbygridpoint()` orders nodes by gridpoint. The *only* reason for this is for the division of labor when using threads: We do repulsion by gridpoint to save re-calculating nearby nodes if there are multiple nodes on one gridpoint, and we do not want to divide nodes on one gridpoint across multiple threads.
    - `collisiondetection()` detects whether nodes are within `NODE_REPULSIVE_RANGE` of one another and adds the repulsive force to `rep_force_vec[]`.
    - `linkforces()` Calculates the forces between nodes due to links and puts into `link_force_vec[]`. If a link goes above a certain threshold force, marks it as broken and removes next time (again to prevent thread problems—since a link is removed both ways and we can't guarantee that both nodes are being processed by same thread)
    - `applyforces()` updates the positions of all the nodes. Sums over the threads for `rep_force_vec[]`, `link_force_vec[]` and `repulsion_displacement_vec[]`.
  - Numerous other functions for things like saving bmps, vrml etc.
- Nucleator class
  - There is only one nucleator object at the moment, `nuc_object`, which is closely linked to the actin object
  - The nucleator is either a sphere, a capsule (i.e. a sphere with a cylindrical segment stuck in the middle) or ellipsoid
  - `addnodes()` adds harbingers to the surface of the nucleator. The probablility of addition of nodes is normalized by surface area and is symmetric if `ASYMMETRIC_NUCLEATION` is zero, or asymmetric if 1 or 2 (stepped or linear bias)
  - `definenucleatorgrid()` sets a list of gridpoints to check in case of nodes entering the nucleator. Called once at the beginning.
  - `iswithinnucleator()` returns true if the node is within the nucleator
  - `collision()` moves a node out of the nucleator along a normal vector
- Nodes class
  - Nodes exist only as members of the actin object
  - `nodegrid` is a 3 dimensional C++ vector of node pointers. Each nodegrid entry starts a circularly linked list of nodes representing the nodes within that gridpoint voxel.
  - The actin class contains a vector of nodes. Each node has an associated `nodenum`, `x y` and `z` position, `nextnode` and `prevnode` node pointers for the nodegrid linked list, `rep_force_vec[]`, `link_force_vec[]` and `repulsion_displacement_vec[]` as described above, the grid position of the node, `harbinger` and `polymer` flags and a `listoflinks` i.e. a vector of link object which attach this node to other nodes.
  - `polymerize()` Creates a node as a harbinger. Adds its pointer to the gridpoint linked list.
  - `depolymerize()` Removes a node, deletes all links and removes from grid.
  - `setgridcoords()` Calculates new grid coordinates based on x,y,z position
  - `addtogrid()` adds the node to the current gridpoint
  - `removefromgrid()` removes node from the grid
  - `updategrid()` checks to see if node has moved gridpoints, and updates grid is needs to
  - `removelink()` removes the specified node from the list of links

- Links class
  - Links exist only as members of the node objects
  - Each link has an associated `linkednodeptr` which points to the target node that the link is to and a `broken` flag which is read by `actin::linkforces()` and tells it to delete the link if it broke.
  - `orig_dist` and `orig_distsqr` store the original distance of the link (and the square of that in a misguided attempt to avoid taking square roots.)
  - `breakcount` stores the number of consecutive iterations the link force has been above `LINK_BREAKAGE_FORCE` and is used to increase the probability of breakage
  - `getlinkforces()` returns the force acting on the link. Also sets the `broken` flag and increments `breakcount` if appropriate

S9 Relation of the model assumptions to theories of and data on actin dynamics

Our model is mesoscopic and does not consider the detailed microscopic mechanisms of force generation by actin filaments growing against a curved surface. We simply use the theories (reviewed in (Mogilner 2006)) supported by the data (Kovar and Pollard 2004; Footer, Kerssemakers et al. 2007) suggesting that individual filaments can grow against pN-range forces. Despite the fact that we do not consider respective pushing forces at the surface explicitly, their existence is crucial, because they maintain the active outward pushing stress at the inner boundary of the shell generating the passive viscoelastic radial and transverse stresses within the shell. The justification for not considering the pushing forces explicitly is as follows.

Three regimes of actin filament growth at the bead or *Listeria* surface are possible: diffusion limited (Plastino, Lelidis et al. 2004), stress limited (van der Gucht, Paluch et al. 2005), and polymerization limited. In the first case, the dense actin gel hinders diffusion of the G-actin to the surface where the polymerization takes place, and the filament growth slows down. In the second case, the radial compression of the expanding actin shell stalls the filament growth. The diffusion-limited regime, however, is only the case when the mesh size of the actin network is small enough (of the order of 30 nm or less (Mogilner and Edelstein-Keshet 2002)). In our case, estimates of the data (Akin and Mullins 2008) suggest that the actin gel mesh size is greater, $\xi \sim 0.1$ μm. In this case, and when the radius of the actin shell is of the order of the bead's radius, the radial stress at the beads surface $\sigma \sim Y$, where Y is the Young modulus of the actin gel (Sekimoto, Prost et al. 2004). The Young modulus can be estimated roughly as $Y \sim \frac{k_B T l_p}{\xi^4}$ (MacKintosh, Kas et al. 1995), where $k_B T \sim 0.004 pN \times \mu m$ is the thermal energy, and $l_p \sim 10 \mu m$ is the actin filament's persistence length. For $\xi \sim 0.1\ \mu m, \sigma \sim Y \sim 400 pN/\mu m^2$, and the force per filament is of the order of $\sigma \times \xi^2 \sim 4 pN$, well below the estimated stall force (reviewed in (Mogilner 2006)).

These estimates suggest that we can assume simply that the actin growth at the surface is equal to a constant polymerization rate. The growing filaments, of course, also produce force, which is not constant: this force balances the growing radial shell compression, but it does not slow down the growth significantly. Mathematically, this assumption translates into the constant rate with which the nascent network nodes are deposited at the random locations at the surface. Following the observations, we assume that the polymerization takes place only at the surface, and that there is no appreciable depolymerization of actin.

The assumption that the nascent nodes are attached to the surface by elastic springs and that these springs break at characteristic yield strain is equivalent, when averaged, to an effective viscous drag (Tawada and Sekimoto, 1991). The fact that the transient attachments of the actin filaments do produce such resistance to propulsion is established (Bernheim-Groswasser, Wiesner et al. 2002; Trichet, Campas et al. 2007).

Modeling of the actin gels with nodes connected by elastic springs is well established (Bottino and Fauci 1998; Shafrir and Forgacs 2002). In our model, many elastic links between the neighboring nodes oriented in random directions correspond to the isotropic elasticity of the actin gel. This is the simplest case; there are no indications of mechanical anisotropy of the Arp2/3-mediated actin gels. At small deformations, the *in silico* gel exhibits linear elasticity; existing estimates (Boal 2001) demonstrate that the mechanical properties of such gel are robust with respect to the exact orientation, number and lengths of the spring-like connections between the nodes. The dimensional magnitude of the Young modulus of our *in silico* gel, which in principle is the parameter sensitive to the springs' lengths, is not important for the model behavior, because we assume that the filaments' growth is force-independent, and that the gel breaking is strain-limited, rather than stress-limited.

We introduced the non-linearity to the springs' behavior to account for the observations that, depending on the system, the gel exhibits either stress-softening, or stress-stiffening (Boal 2001; Gardel, Shin et al. 2004; Gardel, Nakamura et al. 2006) behavior. The viscoelastic properties of the actin gels were measured (Bausch, Moller et al. 1999; Park, Koch et al. 2005; Rogers, Waigh et al. 2008). In our model, the elastic behavior arises from small deformations of the elastic springs, while the viscous behavior ensues when a characteristic yield strain is exceeded, the springs 'snap', and the respective nodes start 'flowing' relative to each other. This behavior corresponds indirectly to Kelvin model of viscoelastic materials (Bird, Armstrong et al. 1977). The yield-strain-limiting behavior of the actin gel was detected many times, recently in (Gardel, Nakamura et al. 2006). It corresponds most likely not to breaking of individual filaments (Tsuda, Yasutake et al. 1996) or proteins connecting the filaments (Fujiwara, Suetsugu et al. 2002), which would be stress-limiting and occur at greater forces, but to disentanglement of stretching filament arrays, which is a geometric phenomenon and therefore is strain-limiting.

Supplemental Material References

Akin, O. and R. D. Mullins (2008). "Capping protein increases the rate of actin-based motility by promoting filament nucleation by the Arp2/3 complex." Cell **133**(5): 841-51.

Bausch, A. R., W. Moller, et al. (1999). "Measurement of local viscoelasticity and forces in living cells by magnetic tweezers." Biophys J **76**(1 Pt 1): 573-9.

Bernheim-Groswasser, A., S. Wiesner, et al. (2002). "The dynamics of actin-based motility depend on surface parameters." Nature **417**(6886): 308-11.

Bird, R. B., R. C. Armstrong, et al. (1977). Dynamics of polymeric liquids. New York ; London, Wiley.

Boal, D. H. (2001). Mechanics of the cell. Cambridge, Cambridge University Press.

Bottino, D. C. and L. J. Fauci (1998). "A computational model of ameboid

deformation and locomotion." Eur Biophys J **27**(5): 532-9.
Footer, M. J., J. W. Kerssemakers, et al. (2007). "Direct measurement of force generation by actin filament polymerization using an optical trap." Proc Natl Acad Sci U S A **104**(7): 2181-6.
Fujiwara, I., S. Suetsugu, et al. (2002). "Visualization and force measurement of branching by Arp2/3 complex and N-WASP in actin filament." Biochem Biophys Res Commun **293**(5): 1550-5.
Gardel, M. L., F. Nakamura, et al. (2006). "Stress-dependent elasticity of composite actin networks as a model for cell behavior." Phys Rev Lett **96**(8): 088102.
Gardel, M. L., J. H. Shin, et al. (2004). "Elastic behavior of cross-linked and bundled actin networks." Science **304**(5675): 1301-5.
Kovar, D. R. and T. D. Pollard (2004). "Insertional assembly of actin filament barbed ends in association with formins produces piconewton forces." Proc Natl Acad Sci U S A **101**(41): 14725-30.
MacKintosh, F. C., J. Kas, et al. (1995). "Elasticity of semiflexible biopolymer networks." Phys Rev Lett **75**(24): 4425-4428.
Mogilner, A. (2006). "On the edge: modeling protrusion." Curr Opin Cell Biol **18**(1): 32-9.
Mogilner, A. and L. Edelstein-Keshet (2002). "Regulation of actin dynamics in rapidly moving cells: a quantitative analysis." Biophys J **83**(3): 1237-58.
Park, S., D. Koch, et al. (2005). "Cell motility and local viscoelasticity of fibroblasts." Biophys J **89**(6): 4330-42.
Plastino, J., I. Lelidis, et al. (2004). "The effect of diffusion, depolymerization and nucleation promoting factors on actin gel growth." Eur Biophys J **33**(4): 310-20.
Rogers, S. S., T. A. Waigh, et al. (2008). "Intracellular microrheology of motile Amoeba proteus." Biophys J **94**(8): 3313-22.
Sekimoto, K., J. Prost, et al. (2004). "Role of tensile stress in actin gels and a symmetry-breaking instability." Eur Phys J E Soft Matter **13**(3): 247-59.
Shafrir, Y. and G. Forgacs (2002). "Mechanotransduction through the cytoskeleton." Am J Physiol Cell Physiol **282**(3): C479-86.
Trichet, L., O. Campas, et al. (2007). "VASP governs actin dynamics by modulating filament anchoring." Biophys J **92**(3): 1081-9.
Tsuda, Y., H. Yasutake, et al. (1996). "Torsional rigidity of single actin filaments and actin-actin bond breaking force under torsion measured directly by in vitro micromanipulation." Proc Natl Acad Sci U S A **93**(23): 12937-42.
van der Gucht, J., E. Paluch, et al. (2005). "Stress release drives symmetry breaking for actin-based movement." Proc Natl Acad Sci U S A **102**(22): 7847-52.