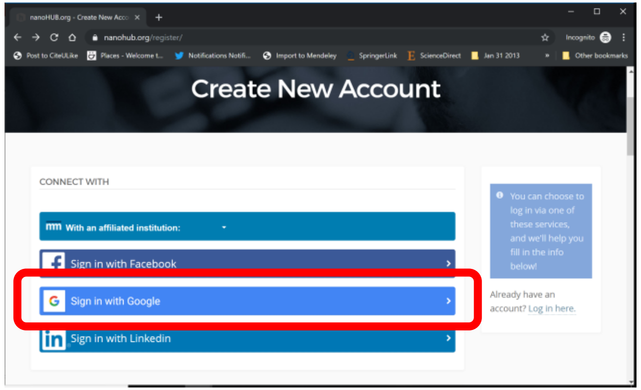
nanoHUB Account

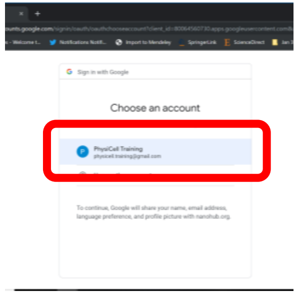
- These tutorials use cloud-hosted PhysiCell models on nanoHUB.org.
- nanoHUB is **free**, but it requires a one-time registration.
- **Steps**:
  1. Visit https://nanohub.org/register
  2. Choose "Sign in with Google"
  3. Choose a Google account
  4. Click "No" (so it doesn't try to associate with some other nanoHUB account)
  5. Finish filling in details, and you're done!
  6. Use your google account for future sign-ins.

2



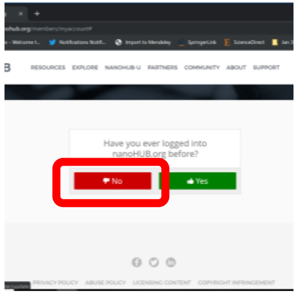
You can choose to log in via one of these services, and we'll help you fill in the info below!

3



To continue, Google will share your name, email address, language preference, and profile picture with nanohub.org.

4


RESOURCES EXPLORE NANOHUB-U PARTNERS COMMUNITY ABOUT SUPPORT
No
Yes
PRIVACY POLICY ABUSE POLICY LICENSING CONTENT COPYRIGHT INFRINGEMENT

Have you ever logged into nanoHUB.org before?

A mini course in PhysiCell

Get the lecture and codes here!

Paul Macklin, Ph.D.

Intelligent Systems Engineering
Indiana University

revised: August 23, 2019

Thanks: Partners

- **Colon cancer metabolism:**
  - Stacey **Finley** (USC, U01 Contact PI)
- **Colon cancer organoids:**
  - Shannon **Mumenthaler** (USC, U01 PI)
- **HPC via EMEWS:**
  - Jonathan **Ozik**, Nicholson **Collier**, Justin **Wozniak**, Charles **Macal** (Argonne National Lab), Chase Cockrell,
  - Gary **An** (University of Vermont)
- **Cloud-deployed PhysiCell models:**
  - Gerhard **Klimeck**, Lynn **Zentner**, Martin **Hunt**, Steve **Clark**, others (NanoHUB Cyberplatform at Purdue)
  - Geoffrey **Fox** (IU PI, nanoBIO Node)
- **PhysiCell:**
  - Randy Heiland (IU)
  - **alumni:** S.H. Friedman (OKSI), A. Ghaffarizadeh (USC)
- **IU PhD students:**
  - John Metzcar (hypoxia, invasion),
  - Yafei Wang (liver mets, nanotherapy)
  - Furkan Kurtoglu (multicellular metabolism, hypoxia)
  - Aneequa Sundus (cyanobacteria, synthetic multicellular systems)
- **IU Undergraduates:**
  - **Metastasis:** B. Fischer, D. Murphy, K. Konstantinopoulos, B. Duggan
  - **Nanotherapy:** T. Mahjan
  - **Jupyter GUIs:** E. Bower, D. Mishler, T. Zhang
  - **PhysiCell technologies:** E. Freeman, A. Connor

Thanks: Funders

- **NIH** (current)
  - NIH CSBC U01 (1U01CA232137), **PIs** Finley* / Macklin / Mumenthaler
  - High-End instrumentation grant (1S10OD018495-01), **PI** Foster
- **NIH** (past support that helped PhysiCell)
  - Provocative Questions grant (1R01CA180149), **PIs** Agus / Atala / Soker
  - NIH PS-OC center grant (5U54CA143907), **PIs** Agus / Hillis
  - HuBMAP CCF contract (OT2 OD026671), **PI** Börner
- **NSF:**
  - Engineered nanoBIO Hub (1720625), **PI** Fox. **Co-PIs** Douglas, Glazier, Macklin, Jadhao
  - Cyanobacteria / Synthetic Biology (1818187), **PI** Kehoe, **Co-PI** Macklin
- **Breast Cancer Research Foundation & JKTGF**, **PI** Macklin
  - projects with **PIs** Agus, Gilkes, Peyton, Ewald, Newton, Bader

JAYNE KOSKINAS TED GIOVANIS | Foundation for Health and Policy

NIH NATIONAL CANCER INSTITUTE

NSF

BCRF BREAST CANCER RESEARCH FOUNDATION

Cancer is a systems problem

Interconnected systems and processes:

- Single-cell behaviors
- Cell-cell communication
- Physics-imposed constraints (e.g., diffusion)
- Systems of systems (e.g., immune system)

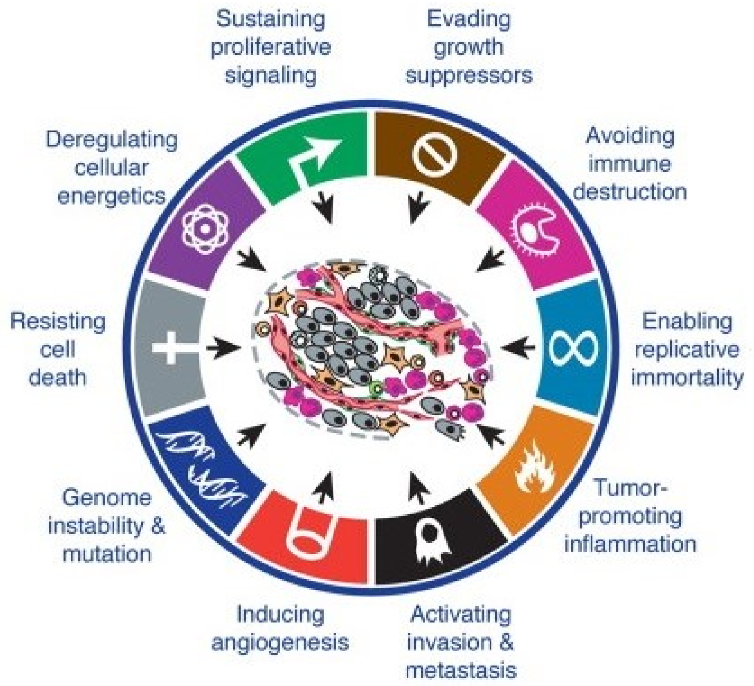
In cancer, these systems become dysregulated.

Treatments target *parts* of these systems.

Cancer is a **complex system:**
changing one part can have **surprising effects**!

Modeling can help **understand** this system.
This is **multicellular systems biology.**

If we can **control** these systems, we've arrived at **multicellular systems engineering**.


Sustaining proliferative signaling
Evading growth suppressors
Avoiding immune destruction
Enabling replicative immortality
Tumor-promoting inflammation
Activating invasion & metastasis
Inducing angiogenesis
Genome instability & mutation
Resisting cell death
Deregulating cellular energetics

Source: Hanahan & Weinberg (2011)
DOI: 10.1016/j.cell.2011.02.013

Design target for multicellular modeling

- **Model multiple diffusing chemical factors**
  - Growth substrates and metabolites
  - Signaling factors
  - Drugs
- **Model many cells in these chemical environments**
  - Environment-dependent behavior
  - Mechanical interactions
  - Heterogeneity:
    - individual states
    - individual parameter values
    - individual model rules
- **Run many copies of the model in high throughput**

BioFVM: Simulating 3-D biotransport

Design goal: Simulate multiple diffusing substrates in 3D with desktops or single HTC/HPC nodes

Typical use: pO_2, glucose, metabolic waste, signaling factors, and a drug, on 10 mm^3 at 20 µm resolution

Features:

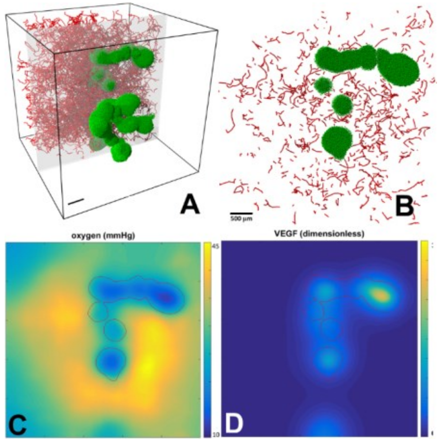
- Off-lattice cell secretion and uptake
- 2nd-order accurate (space), 1st-order accurate (time), numerically stable

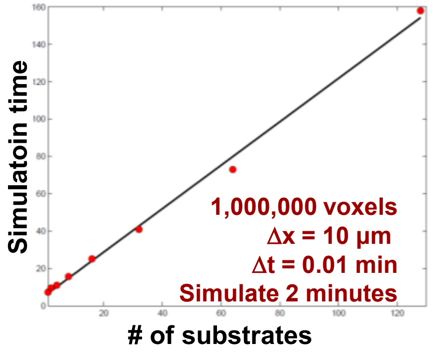
Method:

- Operator splitting, LOD, customized Thomas solvers, etc.
- Standard C++11, cross-platform
- OpenMP parallelization
- O(n) cost scaling in # substrates, # voxels
- Easy to simulate 5-10 substrates on 10^6 voxels

Reference: Ghaffarizadeh et al., *Bioinformatics* (2016)

DOI: 10.1093/bioinformatics/btv730





PhysiCell: A multicellular framework

Design goal: Simulate 10^6 or more cells in 2D or 3D on desktops or single HPC nodes

Features:

- Off-lattice cell positions
- Mechanics-based cell movement
- Cell processes (cycling, motility, …)
- Signal-dependent phenotype
- Can dynamically attach custom data and functions on a cell-by-cell basis
- **Deployed from Raspberry Pi to Crays**

2019 PLoS Computational Biology Research Prize *for* Public Impact

Method:

- Standard C++11, cross-platform
- OpenMP parallelization
- O(n) cost scaling in # cells

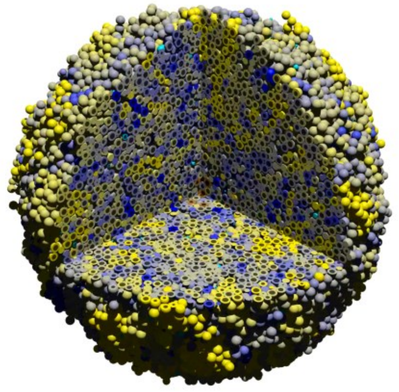
Reference: Ghaffarizadeh et al., PLoS Comput. Biol. (2018)

DOI: 10.1371/journal.pcbi.1005991

Try this model yourself!

nanohub.org/tools/pc4heterogen

Current time: 7 days, 0 hours, and 0.00 minutes
53916 cells

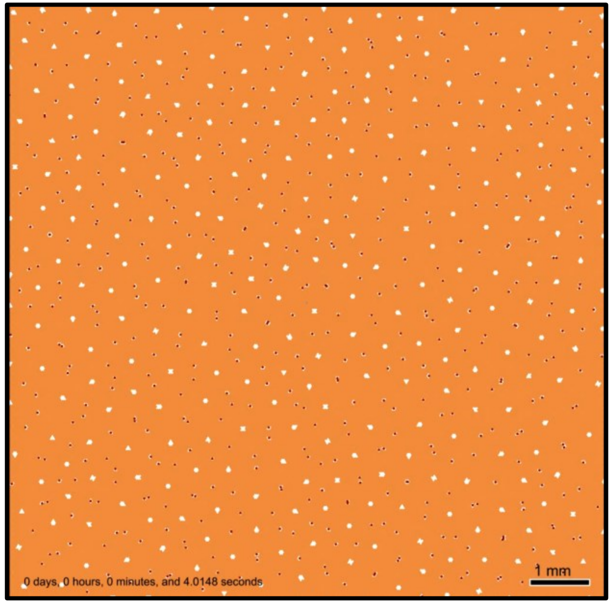


Competition in a 3-D tumor

[View on YouTube (8K)]

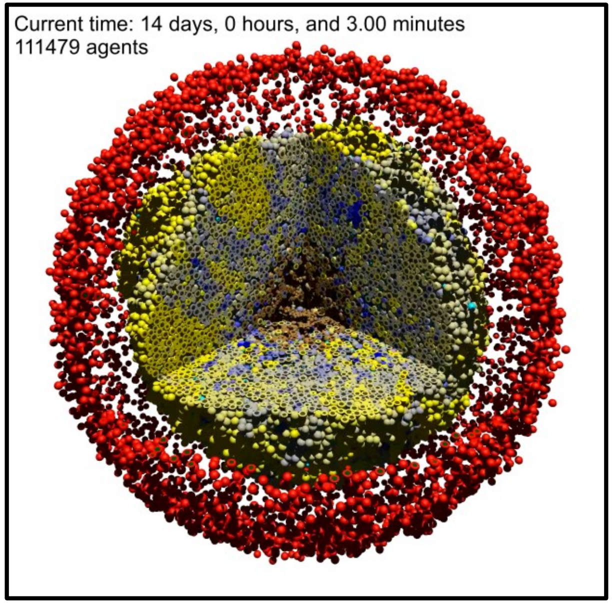
Some examples

Tumor-parenchyma mechanical feedbacks



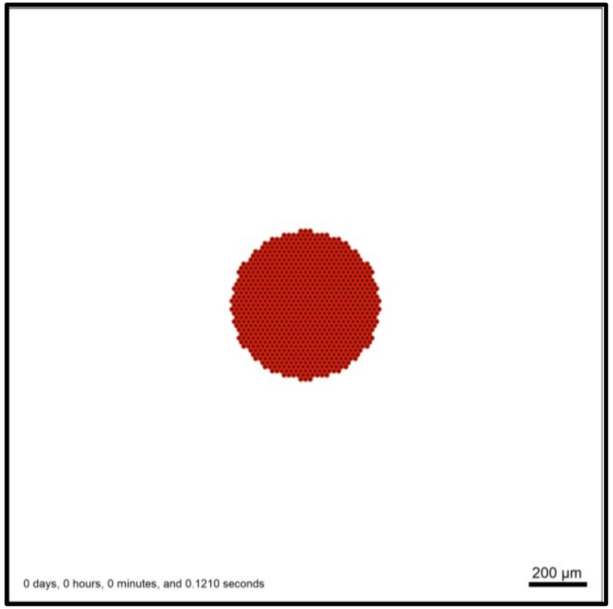
Y. Wang (IU), with Mumemthaler (USC), Sparks (Miami U), Frieboes (U Louisville)

Cancer immunotherapy



with G. An (U. Vermont), Ozik, Wozniak, and Collier (Argonne National Lab)

Phenotypic persistence in breast cancer



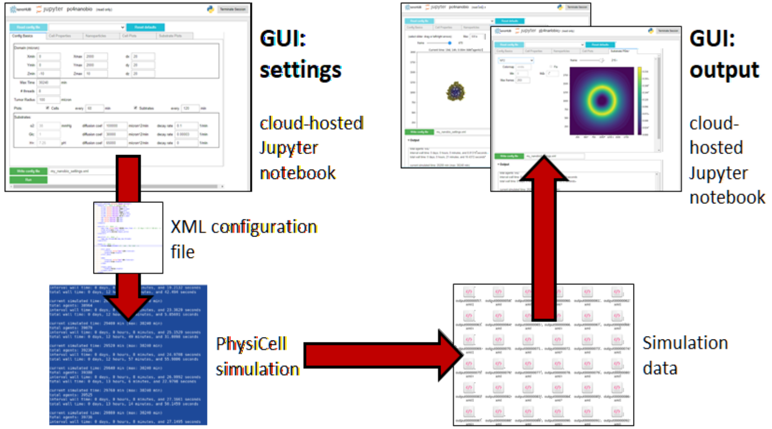
with D. Gilkes (Johns Hopkins U.)

Some key extensions

- Incorporated Boolean signaling networks by combining MaBoSS with PhysiCell to create ***PhysiBoSS*** (Letort et al., *Bioinformatics* 2019)
  - **DOI:** 10.1093/bioinformatics/bty766.
- Performed high-throughput model investigations on supercomputers with Argonne National Lab (Ozik et al., *BMC Bioinformatics* 2018)
  - **DOI:** 10.1186/s12859-018-2510-x
- Extended work to use machine learning to accelerate parameter investigation and interpret model results. (Ozik et al., *Molec. Syst. Des. Eng.* 2019)
  - **DOI:** 10.1039/c9me00036d.
- Automated creation of Jupyter-based GUIs and cloud-hosted PhysiCell-powered models (Heiland et al., *J. Open Source Soft.* 2019)
  - **DOI:** 10.21105/joss.01408

Let's try one!

- **cancer biorobots:**
  - **green:**
    - ♦ cycle entry scales with O2
    - ♦ O2 depletion causes necrosis
    - ♦ cumulative drug exposure causes apoptosis
  - **blue:**
    - ♦ drug-loaded "cargo"
  - **red:**
    - ♦ worker cells that seek and haul cargo towards hypoxic zones



XML+Jupyter architecture

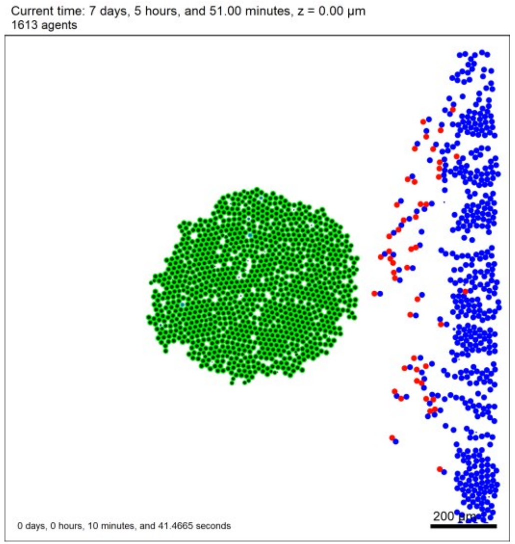
Try this model yourself!

https://nanohub.org/tools/pc4cancerbots

Use case: Research in the classroom

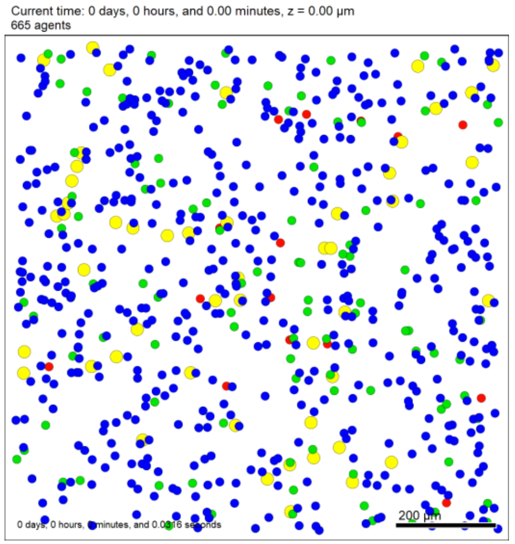
- Because it's easy to convert a research-grade model to a graphical app, we can **quickly integrate new research into the classroom.**

Synthetic cell-based therapy

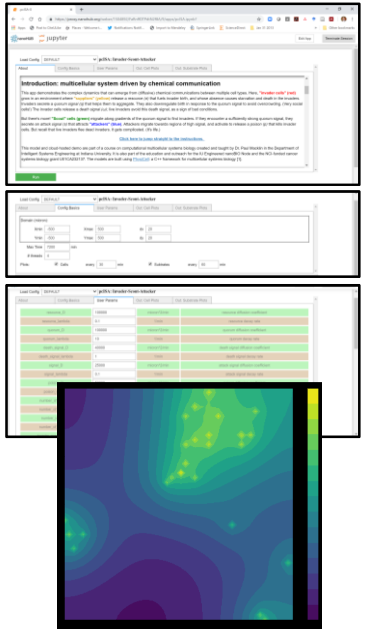


https://nanohub.org/tools/pc4cancerbots

Chemokine demo



https://nanohub.org/tools/pcisa



Introduction: multicellular system driven by chemical communication

This app demonstrates the complex dynamics that can emerge from (diffusive) chemical communications between multiple cell types. Here, "invader cells" (red) grow in an environment where "suppliers" (yellow) release a resource (r) that fuels invader birth, and whose absence causes starvation and death in the invaders. Invaders secrete a quorum signal (q) that helps them to aggregate. They also downregulate birth in response to the quorum signal to avoid overcrowding. (Very social cells!) The invader cells release a death signal (d); live invaders avoid this death signal, as a sign of bad conditions.

But there's more! "Scout" cells (green) migrate along gradients of the quorum signal to find invaders. If they encounter a sufficiently strong quorum signal, they secrete an attack signal (a) that attracts "attackers" (blue). Attackers migrate towards regions of high signal, and activate to release a poison (p) that kills invader cells. But recall that live invaders flee dead invaders. It gets complicated. (It's life.)

Click here to jump straight to the instructions.

This model and cloud-hosted demo are part of a course on computational multicellular systems biology created and taught by Dr. Paul Macklin in the Department of Intelligent Systems Engineering at Indiana University. It is also part of the education and outreach for the IU Engineered nanoBIO Node and the NCI-funded cancer systems biology grant U01CA232137. The models are built using PhysiCell, a C++ framework for multicellular systems biology [1].

Let's download PhysiCell

- Two download options to get the latest numbered release:
  - **GitHub:**
    - https://github.com/MathCancer/PhysiCell/releases/latest
  - **SourceForge:**
    - https://sourceforge.net/projects/physicell/files/latest/download
- Unzip the download, and enter the PhysiCell root directory
- Of particular note, go to ./documentation and open the User_Guide.pdf

A note about time steps

- PhysiCell is designed to account for the multiple time scales inherent to these problems, and has 3 time scales:

$\Delta t_{\text{diffusion}}$	diffusion, secretion, and uptake	(default: 0.01 min)
$\Delta t_{\text{mechanics}}$	cell movement	(default: 0.1 min)
Δt_{cell}	phenotype and volume changes	(default: 6 min)

- This allows some efficiency improvements: not all functions need to be evaluated at each time step.

- See the PhysiCell method paper. (Oddly, not in the User Guide (yet).)

PhysiCell agents

- Each PhysiCell agent has:
  - functions to **sample the microenvironment**
  - **State** elements (things that are measured from an external frame of reference)
    - ID, type, position, velocity, base-to-apex orientation, …
  - **Phenotype**
    - hierarchically-ordered behavioral state and parameters (much more detail soon)
  - user-defined **functions**
    - functions that act on the cell's phenotype (or state)
    - can replace the built-in mechanics and other behaviors
  - user-defined **custom data**
    - used to track additional details specific to user models, if they aren't already

Documentation: User Guide, Sec. 9

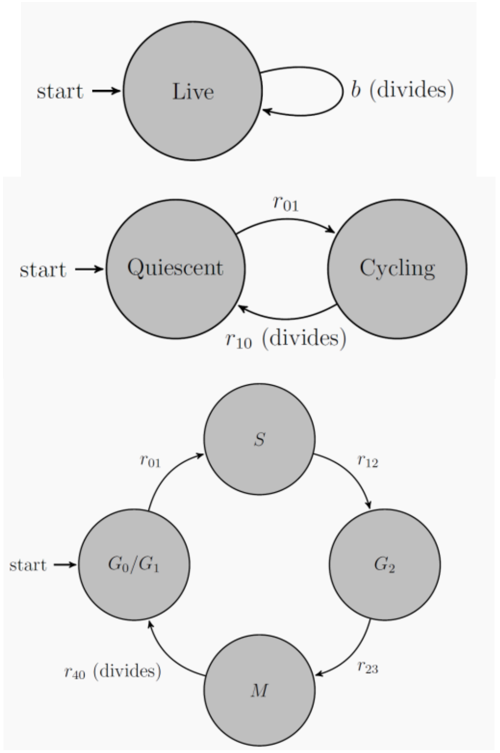
Cell phenotype

- One of the most critical data elements in a PhysiCell Cell is ***phenotype***
- Hierarchically organize key behavioral elements:
  - Phenotype
    - **cycle**: advancement through a cell cycle model
    - **death**: one or more types of cell death
    - **volume:** cell's volume regulation
    - **geometry:** cell's radius and surface area
    - **mechanics:** adhesion and resistance to deformation ("repulsion")
    - **motility:** active motion (other than "passive" mechanics)
    - **secretion:** both release and uptake of chemical substrates. Interfaces with BioFVM
    - **molecular:** a place to store internalized

Documentation: User Guide, Sec. 10

Phenotype: Cycle

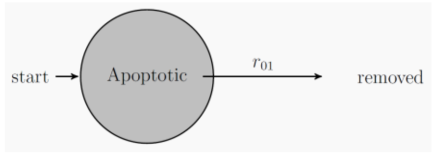
- Each agent's **phenotype** had a **cycle** with:
  - Cycle **model**
    - A directional graph: *nodes* are cycle **phases** $\{P_i\}$ and *edges* are **transition rates** $\{r_{ij}\}$
    - r_{ij} is the transition rate from phase P_i to phase P_j
    - One of the transitions must be marked as a *division transition*
    - Users can attach **arrest condition** functions to these transitions (e.g., size checks)
  - Cycle **data**
    - stores the cell's current transition rates
- **Documentation:** User Guide, Sec. 11.1



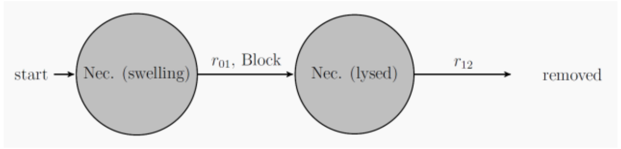
Phenotype: Death

- **Death** has one or more death models:
  - A specialized cycle model with a *removal* transition rate
  - Extra parameters to help govern cell volume
  - Each death model has an associate death rate
  - Also stores an easy Boolean ***dead*** to easily check if the cell is alive.

- PhysiCell has built-in apoptosis and necrosis death models



Apoptosis



Necrosis

Documentation: User Guide, Sec. 11.2

Phenotype: Volume

- **volume** records the cell's sub-volumes:
  - nuclear and cytoplasmic
  - solid vs. fluid
  - calcified fraction
  - key parameters
- a very simple **default model** to regulate volume based on ODEs
  - Change the parameters, target values based on environment and cell state

$$\frac{dV_F}{dt} = r_F(V_F^* - V_F)$$

$$\frac{dV_{NS}}{dt} = r_N(V_{NS}^* - V_{NS})$$

$$\frac{dV_{CS}}{dt} = r_C(V_{CS}^* - V_{CS})$$

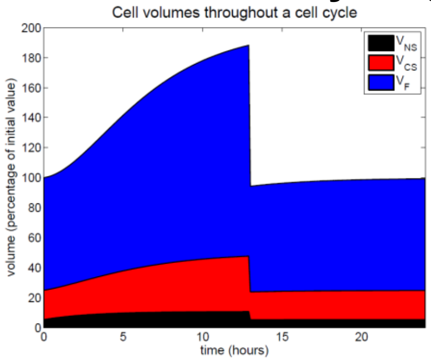
Documentation: User Guide 11.3

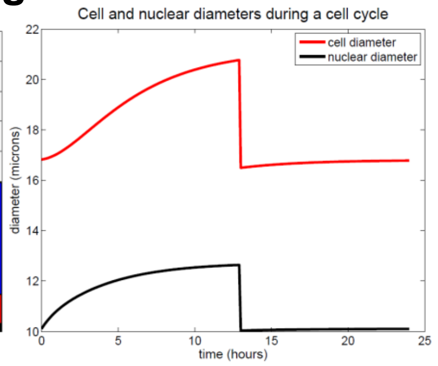
Phenotype: Geometry

- **geometry** records:
  - surface area (not actively tracked)
  - equivalent spherical radius
  - equivalent nuclear radius

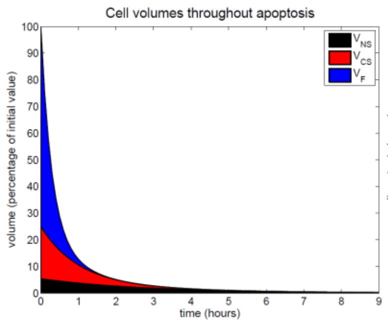
Documentation: User Guide 11.4

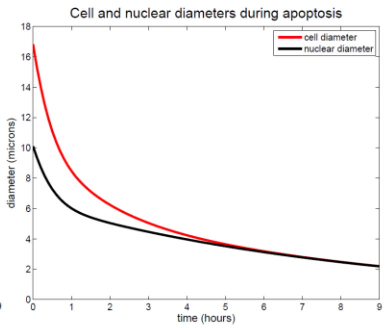
Cycling cells





Apoptotic cells

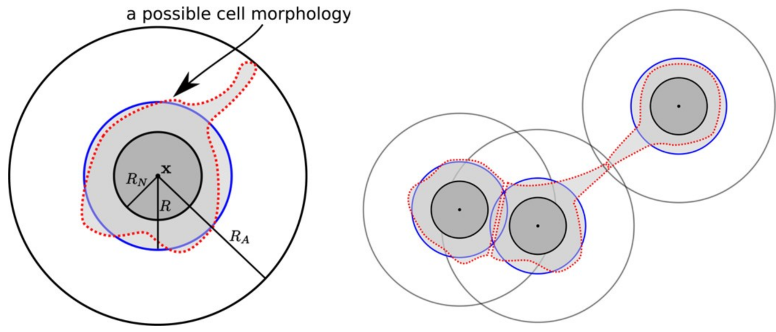


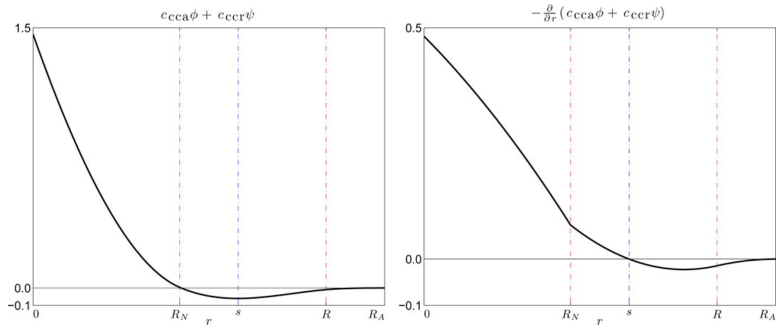


Phenotype: Mechanics

- **Mechanics** keeps parameters for adhesion and "repulsion"
  - Key parameter: maximum adhesion distance
    - ♦ a multiple of the cell's radius
  - (as a multiple of the cell's radius)
- Default model uses potential functions, but this can be supplemented or replaced.

Documentation: User Guide 11.5

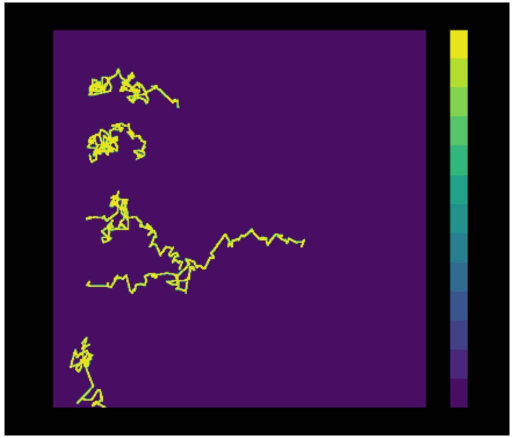




Phenotype: Motility

- **Motility** controls biased random migration
  - Migration speed s
  - Bias direction $\mathbf{d}_{\text{bias}}$
  - Migration bias $0 \leq b \leq 1$
    - If $b = 1$, deterministic motion
    - If $b = 0$, purely Brownian motion
  - Persistence time T_{per}

$$\mathbf{v}_{\text{mot}} \sim s(b\mathbf{d}_{\text{bias}} + (1 - s)\mathbf{d}_{\text{rand}})$$



Documentation: User Guide Sec. 11.6

Try this model yourself! (2D)

https://nanohub.org/tools/motility

Phenotype: Secretion

- **Secretion** stores parameters for secretion and uptake of diffusing substrates

$$\frac{\partial \boldsymbol{\rho}}{\partial t} = \nabla \cdot (\boldsymbol{D} \nabla \boldsymbol{\rho}) - \boldsymbol{\lambda} \cdot \boldsymbol{\rho} + \sum_i \delta(\boldsymbol{x} - \boldsymbol{x}_i) V_i (\boldsymbol{S}_i \cdot (\boldsymbol{\rho}_i^* - \boldsymbol{\rho}) - \boldsymbol{U}_i \cdot \boldsymbol{\rho})$$

Future: Will add an additional "Export" term since the **S** form isn't right for all types of measurements and assays.

Documentation: User Guide Sec. 11.7

Phenotype: Molecular

- **Molecular** is where total internalized substrates are tracked. (optional)
  - A fraction (or all) of substrates can be released at cell death
  - A fraction (or all) of substrates can be transferred when a cell is ingested
  - Internalized substrates are divided among daughter cells

- Eventual support for molecular-scale models will be attached here.

Documentation: User Guide Sec. 11.8

Phenotype-centric programming

- The core cell behaviors are implemented:
  - Cell cycling (with user-selectable models)
  - Cell death
  - Cell adhesion / repulsion
  - Cell motility
  - Cell secretion / uptake

- Modelers can focus on writing functions that control these behaviors.
- This is **phenotype-centric programming.**

Functions in PhysiCell

- Almost all functions in PhysiCell have this form:

```
void my_function( Cell* pCell, Phenotype& phenotype, double dt );
```

All cells have the following key functions (`in pCell->`**`functions`**):

- `volume_update_function`
- `update_migration_bias`
- `custom_cell_rule` (default NULL, evaluated at each mechanics time step)
- `update_velocity`
- `set_orientation`

Documentation: User Guide Sections 9.4.3, 19.1

Cell Definitions

- A **Cell Definition** is a convenient way to set the parameters and functions for a whole class of cells
  - Users can instantiate cells of a specific type using create_cell()
  - With no argument, new cells use the **cell_defaults** definition

- <u>Best practice</u>: set up the **cell_defaults** definition with the correct custom data and functions, then copy this to make new cell definitions

- <u>Tip</u>: Refer back to the phenotype in your agent's cell definition as a reference parameter set (i.e., to get the initial parameter values)

Documentation: User Guide Sec. 9.4.5

Okay, let's try some models

Sample projects

- It would be inefficient (insane) to code new projects from scratch.
- So, we provide sample projects:
  - 2D and 3D template projecgts
  - Sample cancer models

- **make [project-name]:** populate a sample project
  - Then use **make** to compile it
- **make data-cleanup**: clean up the output data
- **make reset**: return to a "clean slate" (depopulate the project)

Documentation: User Guide Sections 6, 7.

Typical model implementation

- Plan!
- Create a new project (start with 2D or 3D template)
- Set up the XML (./config/PhysiCell_settings.xml).
  - Set up the chemical microenvironment
  - Set up user-defined parameters
- Define cells (create_cell_types())
  - Add custom data to the default cell definition
  - Create any needed custom function (for model rules)
  - Create and configure cell definitions
- Place cells (setup_tissue())
  - Place different cell types in their initial positions
- Compile and run

Our first project

Populate the biorobots sample:

```
make biorobots-sample
```

Compile the project:

```
make
```

run the project:

```
./biorobots
```

cleanup:

```
make data-cleanup
make reset
```

Our Project

- Modify heterogeneity sample:
  - Dimensionless oxygen, glucose, and metabolic waste
  - Energy-dependent cycle entry
  - Energy-dependent necrosis
  - Waste-dependent apoptosis
  - Cells have an initial distribution of parameters

- Maybe later, if we have time:
  - A new cell type (interloper) that;
    - use the "glycolytic" energy, secrete waste, resist waste-based death
    - chemotax towards low oxygen

Plan

Example: tumor competition

- Cells can use two types of "metabolism"

$$\frac{dE}{dt} = \alpha\sigma g + \beta g - uE$$

- Cells consume supplies and make waste

$$U_\sigma = 10\alpha$$

$$U_g = 0.2(\alpha + \beta)$$

$$S_w = 10r$$

- All cell behaviors consume energy:

$$u = \mu(0.1 + \alpha + 2\beta + 2r)$$

- Let's give each cell independent "gene expressions" for α, β, r

- Suppose cell cycling increases with energy:

$$b = \begin{cases} 0 & E < 0.1 \\ b_M\left(\frac{E - 0.1}{0.9 - 0.1}\right) & 0.1 < E < 0.9 \\ b_M & 0.9 < E \end{cases}$$

- Necrotic death increases below E = 0.1

$$d_N = d_{\text{nec}}\left(\frac{0.1 - E}{0.1}\right)^+$$

- Apoptotic death increases with waste:
  - But *r* gives tolerance (resistance) to the waste

$$d_A = d_{\text{apop}}\big(1 + 9w(1 - r)\big)$$

What we'll need

- Modify the microenvironment
  - oxygen, glucose, waste
- Change the custom variables in cell defaults
  - alpha, beta, resistance, energy, use_rate
  - Let's leave all the birth and death rates relatively untouched.
- A new phenotype function
  - void energy_based_cell_phenotype(Cell* , Phenotype& , dt)
- Modify the cell_defaults with:
  - Custom data and the new phenotype function.
- Modify the setup_tissue() to choose random values of alpha, beta, resistance for each cell.
- New coloring function (to tailor the SVGs)
  - energy_coloring_function(Cell*)

Create a new project

Modify an existing project

```
make data-cleanup

make reset

make heterogeneity-sample
```

Set up the XML

Let's modify the microenvironment

- Open ./config/PhysiCell_settings.xml
- Browse to **microenvironment_setup** block
- Modify the first **variable**
- Now, copy it to make **glucose** and **waste**

Documentation: User Guide Sec. 13.4

Tutorial:

http://www.mathcancer.org/blog/setting-up-the-physicell-microenvironment-with-xml/

Modifying the microenvironment (1)

Modify oxygen to be dimensionless, set initial and boundary values

```
<variable name="oxygen" units="dimensionless" ID="0">
  <physical_parameter_set>
    <diffusion_coefficient units="micron^2/min">100000.00</diffusion_coefficient>
    <decay_rate units="1/min">.1</decay_rate>
  </physical_parameter_set>
  <initial_condition units="mmHg">1</initial_condition>
  <Dirichlet_boundary_condition units="mmHg"
    enabled="true">1</Dirichlet_boundary_condition>
</variable>
```

Modifying the microenvironment (2)

Copy oxygen to make glucose and waste.

```
<variable name="glucose" units="dimensionless" ID="1">
   <physical_parameter_set>
      <diffusion_coefficient units="micron^2/min">18000</diffusion_coefficient>
      <decay_rate units="1/min">0</decay_rate>
   </physical_parameter_set>
   <initial_condition units="mmHg">1</initial_condition>
   <Dirichlet_boundary_condition units="mmHg" enabled="true">1</Dirichlet_boundary_condition>
</variable>

<variable name="waste" units="dimensionless" ID="2">
   <physical_parameter_set>
      <diffusion_coefficient units="micron^2/min">100000.00</diffusion_coefficient>
      <decay_rate units="1/min">0</decay_rate>
   </physical_parameter_set>
   <initial_condition units="mmHg">0</initial_condition>
   <Dirichlet_boundary_condition units="mmHg" enabled="true">0</Dirichlet_boundary_condition>
</variable>
```

Define cells

Let's add new custom variables

in create_cell_types() in heterogeneity.cpp (This is usually ./custom_modules/custom.cpp in most projects.)

```
// add custom data

cell_defaults.custom_data.add_variable( "alpha" , "dimensionless", 1.0 );
cell_defaults.custom_data.add_variable( "beta" , "dimensionless", 1.0 );
cell_defaults.custom_data.add_variable( "resistance" , "dimensionless", 1.0 );
cell_defaults.custom_data.add_variable( "use_rate" , "dimensionless", 1.0 );
cell_defaults.custom_data.add_variable( "energy" , "dimensionless", 1.0 );
```

Let's make a new phenotype function (1)

First, declare the new function in heterogeneity.h

```
void energy_based_cell_phenotype( Cell* pCell, Phenotype& phenotype, double dt );
```

Next, let's implement. In the function in heterogeneity.cpp

```
void energy_based_cell_phenotype( Cell* pCell, Phenotype& phenotype, double dt )
{
    // housekeeping: one-time searches for variables
        // for finding the right cycle phases
    static int cycle_start_index = live.find_phase_index( PhysiCell_constants::live );
    static int cycle_end_index = live.find_phase_index( PhysiCell_constants::live );

        // for finding the death model indices
    static int apoptosis_i =
        cell_defaults.phenotype.death.find_death_model_index( PhysiCell_constants::apoptosis_death_model );
    static int necrosis_i =
        cell_defaults.phenotype.death.find_death_model_index( PhysiCell_constants::necrosis_death_model );

        // for accessing the custom variables
    static int energy_i = pCell->custom_data.find_variable_index( "energy" );
    static int alpha_i = pCell->custom_data.find_variable_index( "alpha" );
    static int beta_i = pCell->custom_data.find_variable_index( "beta" );
    static int resistance_i = pCell->custom_data.find_variable_index( "resistance" );
    static int use_rate_i = pCell->custom_data.find_variable_index( "use_rate" );

        // for sampling the microenvironment
    static int oxygen_i = microenvironment.find_density_index( "oxygen" );
    static int glucose_i = microenvironment.find_density_index( "glucose" );
    static int waste_i = microenvironment.find_density_index( "waste" );
```

Let's make a new phenotype function (2)

Next, a fun trick: check to see if the cell's dead. If so, set uptake / secretions rates to zero, and overwrite the function pointer so that we don't keep asking dead cells to do things. :-)

```
    // if I'm dead, set secretoin rates to zero, and tell us not to
    // bother checking ever again.

  if( phenotype.death.dead == true )
  {
    phenotype.secretion.set_all_secretion_to_zero();
    phenotype.secretion.set_all_uptake_to_zero();

    pCell->functions.update_phenotype = NULL;
    return;
  }
```

Let's make a new phenotype function (3)

Let's do the energy model

```
     // do the basic energy model
     // use rate : u = alpha + beta + resistance
   double alpha = pCell->custom_data[alpha_i];
   double beta = pCell->custom_data[beta_i];
   double resistance = pCell->custom_data[resistance_i];

   pCell->custom_data[use_rate_i] = 0.1 + alpha + 2*beta + 2*resistance;
     // sample the oxygen, glucose, and waste
   double oxygen = pCell->nearest_density_vector()[oxygen_i];
   double glucose = pCell->nearest_density_vector()[glucose_i];
   double waste = pCell->nearest_density_vector()[waste_i];
     // run the ODE. Let's use backwards Euler
   double use_rate = pCell->custom_data[use_rate_i];

   pCell->custom_data[energy_i] +=
     dt*( alpha*oxygen*glucose + beta*glucose );
   pCell->custom_data[energy_i] /= ( 1.0 + dt*use_rate );
```

Let's make a new phenotype function (4)

Let's uptake secretion/ uptake

```
    // set secretion parameters
  phenotype.secretion.secretion_rates[waste_i] = beta * 10.0;
  phenotype.secretion.saturation_densities[waste_i] = 1.0;
    // set uptake rates
  phenotype.secretion.uptake_rates[oxygen_i] = alpha * 10.0;
  phenotype.secretion.uptake_rates[glucose_i] = 0.2*(alpha+beta);
```

Next, the cell cycle rate

```
    // set cycle parameters
  double energy = pCell->custom_data[energy_i];
  double scale = ( energy - 0.1 )/( 0.9 - 0.1 );
  if( scale > 1.0 )
  { scale = 1.0; }
  if( scale  < 0.0 )
  { scale = 0.0; }
  phenotype.cycle.data.transition_rate( cycle_start_index ,cycle_end_index ) =
    6.94e-4 * scale;
  // 1/24 hr^-1 max birth rate, in units of min^-1
```

Let's make a new phenotype function (5)

Lastly, let's set the death rates

```
    // set necrotic death rate
  scale = ( 0.1 - energy )/0.1;
  if( scale < 0.0 )
  { scale = 0.0; }
  if( scale > 1.0 )
  { scale = 1.0; }
  phenotype.death.rates[necrosis_i] = scale * 0.01 ;
    // 100 minute survival time when zero energy;

    // set the apoptotic death rate
  scale = 1.0 + 9.0*(1.0-pCell->custom_data[resistance_i])*waste;
  phenotype.death.rates[apoptosis_i] = scale * 6.94e-6;
  // 1% of the max birth rate

  return;
}
```

Checkpoint:

To catch up to the code at this point, unzip checkpoint2.zip

Now, in create_cell_types, let's switch to this rule:

```
  cell_defaults.functions.update_phenotype = energy_based_cell_phenotype;
```

Place cells

Modify setup_tissue()

Some bookkeeping

```
// some bookkeeping
static int energy_i = cell_defaults.custom_data.find_variable_index( "energy" );
static int alpha_i = cell_defaults.custom_data.find_variable_index( "alpha" );
static int beta_i = cell_defaults.custom_data.find_variable_index( "beta" );
static int resistance_i = cell_defaults.custom_data.find_variable_index("resistance");
static int use_rate_i = cell_defaults.custom_data.find_variable_index( "use_rate" );
```

and in four places

```
// choose random properties
pCell->custom_data[alpha_i] = UniformRandom();
pCell->custom_data[beta_i] = UniformRandom();
pCell->custom_data[resistance_i] = UniformRandom();
```

Then clear out unneeded junk

Make a coloring function

A custom coloring function for SVGs (1)

Declare the function in heterogeneity.h

```
std::vector<std::string> energy_coloring_function( Cell* );
```

Create it in heterogeneity.cpp

```
std::vector<std::string> energy_coloring_function( Cell* pCell )
{
   // color 0: cytoplasm fill
   // color 1: outer outline
   // color 2: nuclear fill
   // color 3: nuclear outline

   // some bookkeeping
   static int energy_i = pCell->custom_data.find_variable_index( "energy" );
   static int alpha_i = pCell->custom_data.find_variable_index( "alpha" );
   static int beta_i = pCell->custom_data.find_variable_index( "beta" );
   static int resistance_i = pCell->custom_data.find_variable_index( "resistance" );
   static int use_rate_i = pCell->custom_data.find_variable_index( "use_rate" );

   // start black
   std::vector< std::string > output( 4, "black" );
```

A custom coloring function for SVGs (2)

Sample cell properties and set colors (if not dead)

```
// nucleus: color by the three "genes"
// red: alpha
// green: beta
// blue: resistance
// cytoplasm: energy (black = 0, white >= 1)
if( pCell->phenotype.death.dead == false )
{
   int red = (int) round( 255.0 * pCell->custom_data[alpha_i] );
   int green = (int) round( 255.0 * pCell->custom_data[beta_i] );
   int blue = (int) round( 255.0 * pCell->custom_data[resistance_i] );
   int grey = (int) round( 255.0 * pCell->custom_data[energy_i] );
   char szTempString [128];
   sprintf( szTempString , "rgb(%u,%u,%u)", red, green, blue );
   output[2].assign( szTempString ); // nucleus by alpha, beta, resistance "genes"
   sprintf( szTempString , "rgb(%u,%u,%u)", grey, grey, grey );
   output[0].assign( szTempString ); // cyto by energy

   return output;
}
```

A custom coloring function for SVGs (3)

Standard dead colors

```
   // if not, dead colors

   if (pCell->phenotype.cycle.current_phase().code == PhysiCell_constants::apoptotic )
      // Apoptotic - Red
   {
      output[0] = "rgb(255,0,0)";
      output[2] = "rgb(125,0,0)";
   }

   // Necrotic - Brown
   if( pCell->phenotype.cycle.current_phase().code == PhysiCell_constants::necrotic_swelling ||
      pCell->phenotype.cycle.current_phase().code == PhysiCell_constants::necrotic_lysed ||
      pCell->phenotype.cycle.current_phase().code == PhysiCell_constants::necrotic )
   {
      output[0] = "rgb(250,138,38)";
      output[2] = "rgb(139,69,19)";
   }

   return output;
}
```

A custom coloring function for SVGs (4)

Select this coloring function in main.cpp

```
   std::vector<std::string> (*cell_coloring_function)(Cell*) = energy_coloring_function;
```

Now, compile and run! (Can take 20-60 minutes, depending on your machine.)

```
make && ./heterogeneity
```

Checkpoint:

To catch up to the code at this point, unzip checkpoint3.zip

Some postprocessing tips

convert the SVGs to JPEGs

```
magick mogrify -format jpg -resize 50% snap*.svg
```

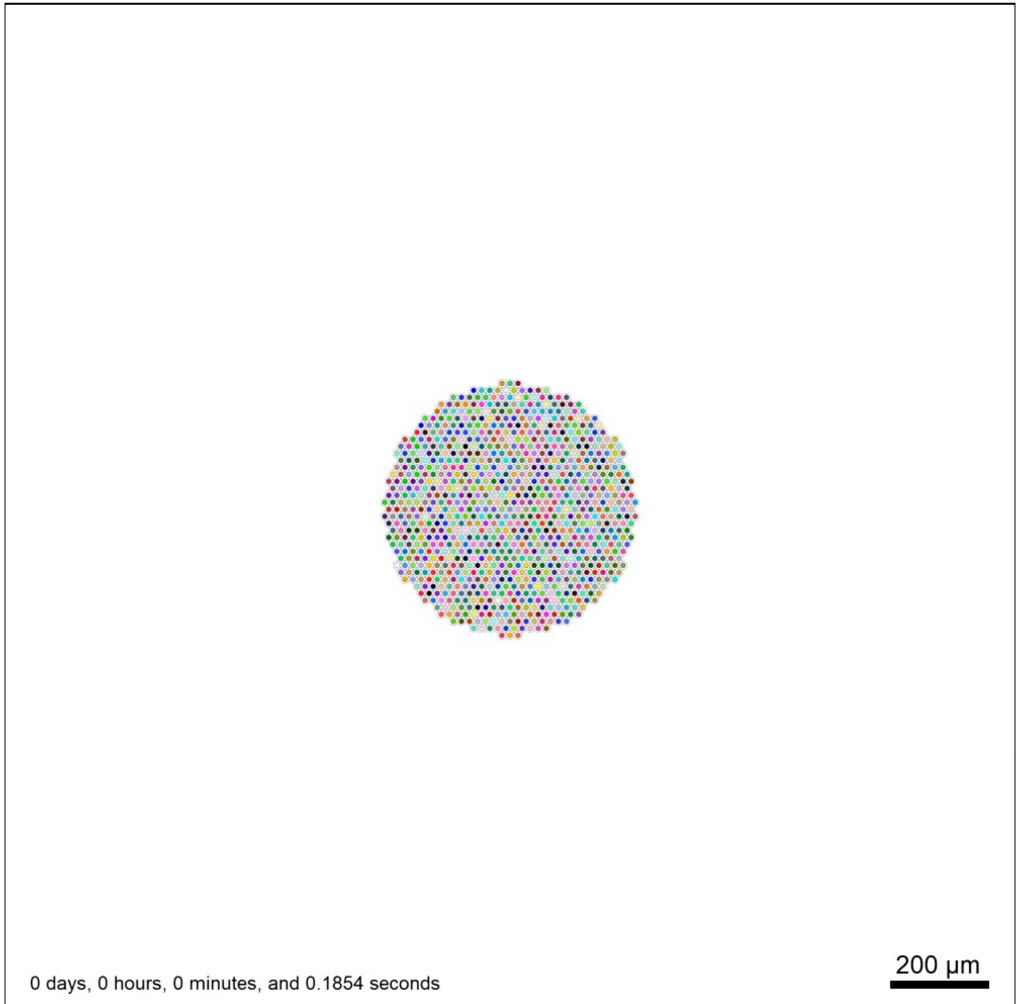
stitch those JPEGs into an animated GIF

```
magick convert *.jpg out.gif
```

create a movie

```
mencoder "mf://snapshot*.jpg" -ovc lavc -lavcopts vcodec=mpeg4:vbitrate=10000:mbd=2:trell
-mf fps=24:type=jpg -nosound -o out.avi
```

Current time: 0 days, 0 hours, and 0.00 minutes, z = 0.00 μm
889 agents



0 days, 0 hours, 0 minutes, and 0.1854 seconds

Other work that should be done

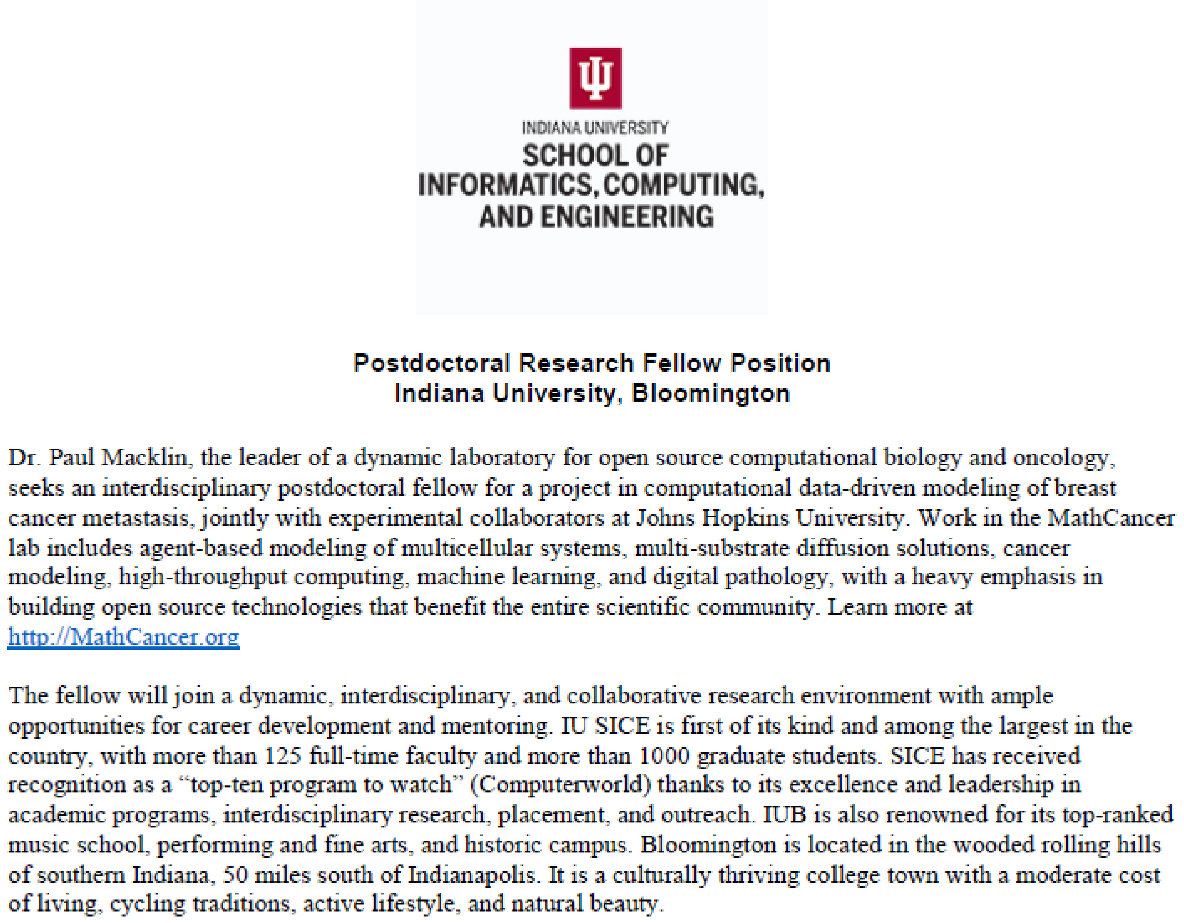
- Remove hard-coded parameters
  - Use "user_parmameters" in the XML to set the values
- Use internalized substrates for the energy model
- Create cloud-hosted version of the model

Opportunities

- **Winter 2019:** All-new training materials (Thanks, NCI!)
  - A series of ~10-15 minute modules with webapps to illustrate key concepts
  - New website and wiki at PhysiCell.org
  - Basic online development environment on nanoHUB

- **Spring 2020:** funded hackathon at Indiana University (Thanks, NCI!)
  - Contribute to the PhysiCell ecosystem

- **Summer 2020:** two NCI-funded 6-week visitors to IU (Yay, NCI!)
  - Likely recruited from the hackathon participants
  - Build a model or new PhysiCell capability, learn to create nanoHUB apps


INDIANA UNIVERSITY
SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

Postdoctoral Research Fellow Position
Indiana University, Bloomington

Dr. Paul Macklin, the leader of a dynamic laboratory for open source computational biology and oncology, seeks an interdisciplinary postdoctoral fellow for a project in computational data-driven modeling of breast cancer metastasis, jointly with experimental collaborators at Johns Hopkins University. Work in the MathCancer lab includes agent-based modeling of multicellular systems, multi-substrate diffusion solutions, cancer modeling, high-throughput computing, machine learning, and digital pathology, with a heavy emphasis in building open source technologies that benefit the entire scientific community. Learn more at http://MathCancer.org

The fellow will join a dynamic, interdisciplinary, and collaborative research environment with ample opportunities for career development and mentoring. IU SICE is first of its kind and among the largest in the country, with more than 125 full-time faculty and more than 1000 graduate students. SICE has received recognition as a "top-ten program to watch" (Computerworld) thanks to its excellence and leadership in academic programs, interdisciplinary research, placement, and outreach. IUB is also renowned for its top-ranked music school, performing and fine arts, and historic campus. Bloomington is located in the wooded rolling hills of southern Indiana, 50 miles south of Indianapolis. It is a culturally thriving college town with a moderate cost of living, cycling traditions, active lifestyle, and natural beauty.

[link]

Postdoc!

PhysiCell models of metastatic breast cancer

INDIANA UNIVERSITY
SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

Macklin lab
@MathCancer
MathCancer.org