Slides, videos, links and more:
https://github.com/physicell-training/ws2021

Session 9: Contact and Pressure in PhysiCell

Paul Macklin, Ph.D.
@MathCancer

PhysiCell Project

July 28, 2021

Goals

- Mechanical Pressure
- Example: pressure-based proliferation (function only)
- Testing for cell contact
- Cell ingestion
- Example: Predator rule (function only)

Mechanical Pressure (Compression)

- **Mechanical pressure** is the total force on a cell's surface, divided by its surface area.
- For the *i*th cell with surface area S_i, we can create a "pressure" p_i based on the interaction potentials ψ of neighboring cells (with indices in N_i):

$$p_i = \frac{1}{S_i} \sum_{j \in N_i} -\nabla\psi(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

- In PhysiCell, we calculate this as ***nondimensionaled*** cell.state.simple_pressure
  - Normalized for confluence in 3D:
    - ♦ pressure = 1 in 3-D confluence (12 neighbors of similar size)
    - ♦ pressure = 0.5 in 2-D confluence (6 neighbors of similar size)
    - ♦ eliminates the need for calculating surface area since we don't model cell morphology

Note: A ***confluent tissue*** is one with no gaps.
The cells' volumes are "squeezed" into the areas that render as white triangles.

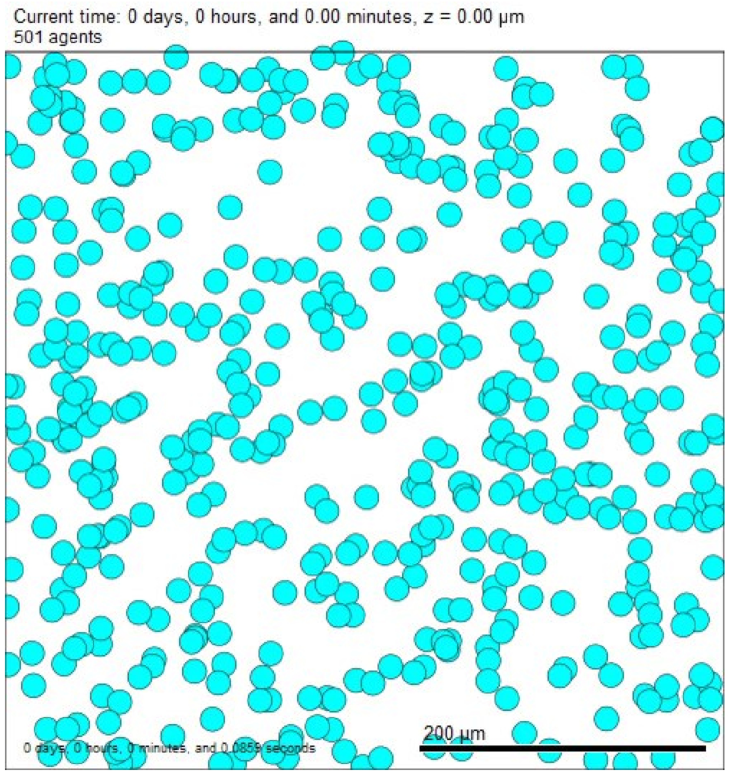
Examle: Pressure-dependent phenotype

```
void pressure_phenotype( Cell* pCell, Phenotype& phenotype , double dt )
{
   // get my cell definition
   static Cell_Definition* pCD = find_cell_definition( pCell->type );

   // exit early if dead
   if( phenotype.death.dead == true )
   {
      pCell->functions.update_phenotype = NULL;
      return;
   }

   // compare my pressure to the threshold
   // allow cycling if pressure is below threshold.
   if( pCell->state.simple_pressure < pCell->custom_data["pressure_threshold"] )
   {
      phenotype.cycle.data.transition_rate(0,1) = pCD->phenotype.cycle.data.transition_rate(0,1);
      pCell->custom_data["arrested"] = 0;
   }
   else
   {
      phenotype.cycle.data.transition_rate(0,1) = 0;
      pCell->custom_data["arrested"] = 1;
   }
   return;
}
```

Sample result

```
// Set number of cells to 500.
// Set simulation domain to [-250,250]^2
// output SVG every 15 minutes

make
./project
make jpeg
make movie
```

- ● Pressure-arrested cell
- ● Non-arrested, not cycling
- ● Non-arrested, actively cycling
- ● Apoptotic



Other details:

Flow cytometry model (separated):

$\frac{1}{r_{01}} = 30$ min $\quad \frac{1}{r_{12}} = 90$ min

$\frac{1}{r_{23}} = 120$ min $\quad \frac{1}{r_{30}} = 30$ min

Example: Mechanofeedback & chemical communication

- Suppose:
  - Cycling cells secrete a diffusible signal s
  - Cell's down-regulate cycle entry if $p > 0.5$ or if $s > s_{\text{stop}}$

- Here's the expected behavior:
  - Apoptosis events reduce pressure on 6 neighbors (in 2D)
  - All 6 neighbors could proliferate to fill the gap opened by the 1 dead cell
  - Stochastically, once cell "chooses" to divide first
  - This cell secretes s to prevent the other 5 from dividing.

Pressure + Signal phenotpe

```
    // etc etc etc etc

    static int nSignal = microenvironment.find_density_index( "signal" );

    // if cycling, secrete signal
    if( phenotype.cycle.data.current_phase_index > 0 )
    { phenotype.secretion.secretion_rates[nSignal] = 1; }
    else
    { phenotype.secretion.secretion_rates[nSignal] = 0; }

    // compare my signal to the threshold (store result in signal_arrested)
    if( pCell->nearest_density_vector()[nSignal] < pCell->custom_data["signal_threshold"])
    { pCell->custom_data["signal_arrested"] = 0; }
    else
    { pCell->custom_data["signal_arrested"] = 1; }

    // compare my pressure to the threshold (store result in pressure_arrested)
    if( pCell->state.simple_pressure < pCell->custom_data["pressure_threshold"] )
    { pCell->custom_data["pressure_arrested"] = 0; }
    else
    { pCell->custom_data["pressure_arrested"] = 1; }

    // if either condition holds, arrest cycle entry
    if( pCell->custom_data["pressure_arrested"] > 0.5 && pCell->custom_data["signal_stop"] > 0.5 )
    { phenotype.cycle.data.transition_rate(0,1) = 0; }
    else
    { phenotype.cycle.data.transition_rate(0,1) = pCD->phenotype.cycle.data.transition_rate(0,1); }

    return;
}
```

Sample result

```
// Set number of cells to 1400.
// Set simulation domain to [-250,250]^2
// set SVG output interval to 3 minutes
// set max time to 3000 minutes

make
./project
make jpeg
make movie
```

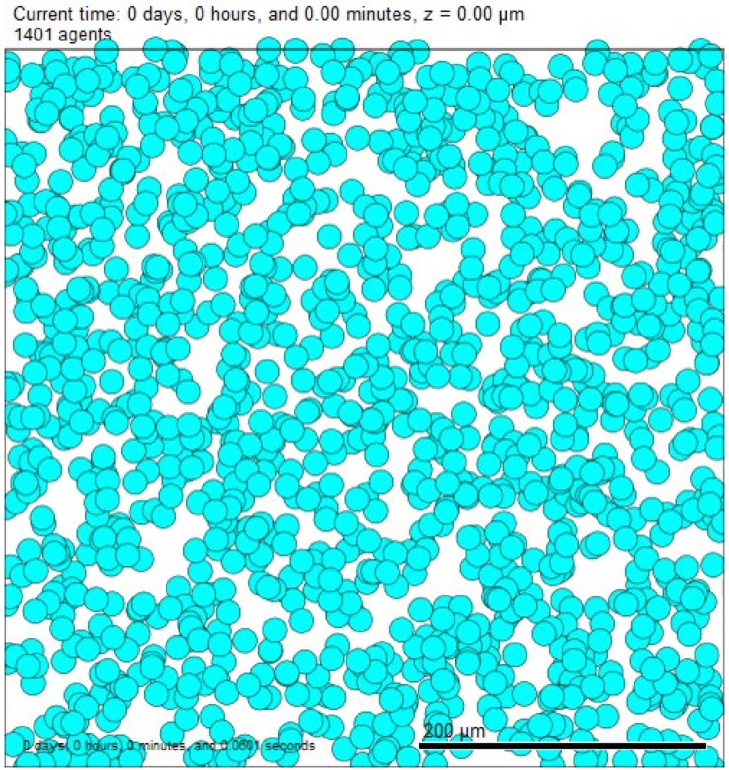
● Signal-arrested cell
● Signal & Pressure-arrested cell
● Pressure-arrested cell
● Non-arrested, not cycling
● Non-arrested, actively cycling
● Apoptotic

Other details:

Flow cytometry model (separated):

$\frac{1}{r_{01}} = 30$ min $\frac{1}{r_{12}} = 90$ min

$\frac{1}{r_{23}} = 120$ min $\frac{1}{r_{30}} = 30$ min


Current time: 0 days, 0 hours, and 0.00 minutes, z = 0.00 μm
1401 agents
0 days, 0 hours, 0 minutes, and 0.0801 seconds
200 μm

Testing for Contact

- As of PhysiCell 1.8.0, each cell actively tracks a vector of Cell pointers **`pCell->state.neighbors`**:
  - Interaction potential records all cells with non-zero adhesion/repulsion
  - Updated every mechanics time step
  - ***Note:*** Requires that you use the default cell velocity function

Testing for contact (backup methods)

- In addition, the Cell class has three ways to test for nearby cells
  - `std::vector<Cell*>& Cell::cells_in_my_container( void );`
    - This returns a vector of the memory addresses of cells in the same mechanics voxel.
    - It's very fast and very cheap, but it may miss some nearby cells.
    - **Note:** This also includes your cell in the list! Make sure to test against `pCell` when using!
  - `std::vector<Cell*> Cell::nearby_cells( void );`
    - This returns a vector of the memory addresses of all cells nearby.
    - It returns all the cells in neighboring mechanics voxels.
    - It's more robust and complete, but it has a higher computational cost.
    - **Note:** This also includes your cell in the list! Make sure to test against `pCell` when using!
  - `std::vector<Cell*> Cell::nearby_interacting_cells( void );`
    - This returns a vector of the memory addresses of all cells nearby ***except*** the cell.
    - It returns all the cells in the neighboring voxels within interaction distance. (same as default potential functions)
    - It's more robust and complete, but it has a higher computational cost. But it returns fewer cells!

Testing for contact (backup methods 2)

- You can also test for nearby cells for any cell *pCell*
  - `std::vector<Cell*> Cell::nearby_cells( Cell* pCell );`
    - ♦ This returns a vector of the memory addresses of all cells nearby.
    - ♦ It returns all the cells in neighboring mechanics voxels.
    - ♦ It's more robust and complete, but it has a higher computational cost.
    - ♦ **Note:** This also includes your cell in the list! Make sure to test against `pCell` when using!

  - `std::vector<Cell*> Cell::nearby_interacting_cells(Cell* pCell);`
    - ♦ This returns a vector of the memory addresses of all cells nearby ***except*** **`pCell`*.***
    - ♦ It returns all the cells in the neighboring voxels within interaction distance.
      (same as default potential functions)
    - ♦ It's more robust and complete, but it has a higher computational cost. It returns fewer cells!
    - ♦ If you are using the default mechanics model, it returns the same list as `pCell->state.neighbors`.

Ingestion

- A cell (predator) can "eat" another cell (prey)
  - The prey cell solid volume is added to the cytoplasmic solid
  - The prey cell fluid volume is added to the fluid volume
  - The prey cell's volumes are set to zero
  - The prey cell is inactivated (all functions NULL), secretion/uptake set to zero, and any attachments set to zero.
  - The predator's volume will actively shrink back towards its target volume

- This is useful for predation, such as by macrophages.

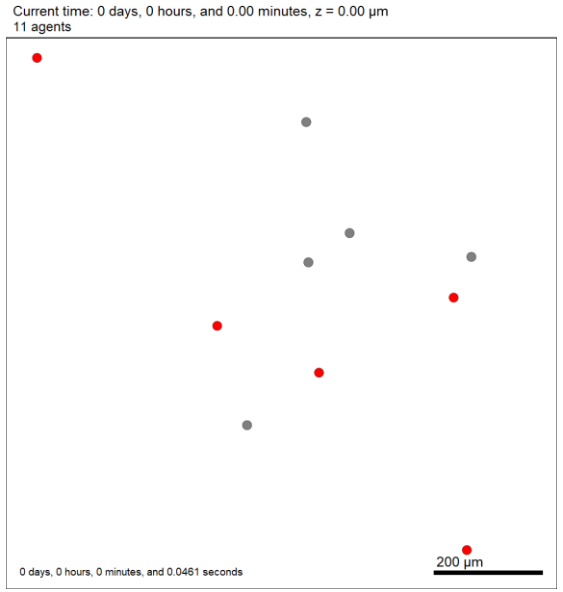
- `void Cell::ingest_cell( Cell* pCell_to_eat )`

Example: Predator rule

```
// Use this in the "custom" rule to evaluate on the mechanics time scale.
// Test for contact with prey cells and eat them.

void custom_predator_function( Cell* pCell, Phenotype& phenotype , double dt )
{
  static Cell_Definition* pPrey = find_cell_definition( "prey" );

  Cell* pC;
  for( int n = 0; n < pCell->state.neighbors.size() ; n++ )
  {
    pC = pCell->state.neighbors[n];
    if( pC->type == pPrey->type )
    { pCell->ingest_cell( pC ); }
  }

  return;
}
```

Sample result

```
## set the max run time to 3600 minutes
## set SVG output to every 20 minutes

make
./project
make gif
```



Funding Acknowledgements

PhysiCell Development:

- Breast Cancer Research Foundation
- Jayne Koskinas Ted Giovanis Foundation for Health and Policy
- National Cancer Institute (U01CA232137)
- National Science Foundation (1720625)

Training Materials:

- Administrative supplement to NCI U01CA232137 (Year 2)