

Slides, videos, links and more:

<https://github.com/physicell-training/ws2021>

Session 10: Examples of Contact Functions in PhysiCell

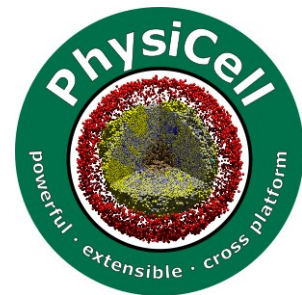


Paul Macklin, Ph.D.

 @MathCancer

PhysiCell Project

July 28, 2021



LUDDY

SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

PhysiCell Project

PhysiCell.org

 @PhysiCell

Goals

- Introduce cell contact functions
- Built-in spring adhesion functions
- Example: "worms"

Attaching cells

- PhysiCell allows you to manually connect (attach) cells for interactions:
 - `void Cell::attach_cell(Cell* pAddMe);`
 - ♦ This attaches `pAddMe` to the `Cell`.
 - ♦ It also attaches the `Cell` to `pAddMe`. (Attachments are tracked symmetrically.)
 - ♦ This operation checks to make sure they are not already attached.
 - ♦ The operation is thread-safe.
 - `void attach_cells(Cell* pCell_1, Cell* pCell_2)`
 - ♦ This attaches `pCell_2` to `pCell_1`
 - ♦ This attaches `pCell_1` to `pCell_2`
 - ♦ This operation checks to make sure they are not already attached.
 - ♦ The operation is thread-safe.
- Each cell tracks a vector of attached cells in `Cell.state.attached_cells`.
 - Currently no functions in PhysiCell use this data structure.
 - You have complete freedom to manage and use it.
- `Cell.state.number_of_attached_cells()` gives the number of attached cells.

Detaching cells

- PhysiCell allows you to manually disconnect (detach) cells for interactions:
 - `void Cell::detach_cell(Cell* pRemoveMe);`
 - ♦ This detaches `pRemoveMe` from the `Cell`.
 - ♦ It also detaches the `Cell` from `pRemoveMe`. (Attachments are tracked symmetrically.)
 - ♦ The operation is thread-safe.
 - `void detach_cells(Cell* pCell_1, Cell* pCell_2)`
 - ♦ This detaches `pCell_2` from `pCell_1`
 - ♦ This detaches `pCell_1` from `pCell_2`
 - ♦ The operation is thread-safe.
 - `void Cell::remove_all_attached_cells(void)`
 - ♦ This runs `detach_cell()` on each attached `Cell`
- Cell division removes all attached cells
- Cell death removes all attached cells (at the end of death through the destructor `~Cell()`)

Example

- Suppose a cell is attempting to aggregate with similar cells
- We can have it search for nearby cells, and attach them if they are close.
- We can also have them detach if they sense a diffusible signal

```
void custom( Cell* pC , Phenotype& p , double dt )
{
    // test all nearby cells
    for( int k=0; k < pC->state.neighbors.size() ; k++ )
    {
        Cell* pThem = pC->state.neighbors[k];
        std::vector<double> displacement = pThem->position - pC->position;
        double distance = norm( displacement );

        // attach if (1) closer than sum of radii and (2) not attached to more than 6 cells.
        if( norm < pThem->phenotype.geometry.radius + p.geometry.radius &&
            pC->state.number_of_attached_cells() < 7 )
        { pC->attach_cell( pThem ); }
    }

    // scatter signal makes cells detach from everyone
    static int nS = microenvironment.find_density_index("signal");
    if( pC->nearest_density_vector()[nS] > 0.1 )
    { pC->remove_all_attached_cells(); }
    return;
}
```

Contact functions

- If Cell A and Cell B are attached, they can interact by contact functions
 - Contact functions are evaluated once per mechanics time step.
 - ♦ **Warning:** This is embedded in an OpenMP loop, so do consider thread safety!
 - The functions can act on both A and B.
 - Supposing A calls the function:
 - ♦ Writing to Cell A is thread-safe.
 - ♦ Writing to Cell B is not thread-safe. use `#pragma omp critical`
- Each cell has a contact function (default: NULL):
 - `Cell.functions.contact_function;`
- At each mechanics time step, each Cell does this:
 - For each `pCell` in `Cell.state.neighbors()` :
 - ♦ evaluate `contact_function(this , pCell);`

Contact functions: format

- Contact functions are a variation on the standard function syntax:
 - They need to reference both interacting cells:

```
some_contact_function( Cell* pA, Phenotype& phenotypeA, Cell* pB, Phenotype& phenotypeB, double dt );
```

- Cell A (first argument) tends to call the function
 - ♦ pA is pointer to Cell A, with phenotype phenotypeA.
 - ♦ pB is pointer to Cell B, with phenotype phenotypeB.
- **Important note:** Often, A and B will have the same contact function:
 - Cell A will call `contact_function(pCellA,phA, pCellB,phB, dt);`
 - Cell B will call `contact_function(pCellB,phB, pCellA,phA, dt);`
 - Be aware of this to avoid "double counting" the interaction!
 - Also, take advantage of this for computational efficiency and thread safety:
 - ♦ Let the first call only write to cell A.
 - ♦ Let the second call only write to cell B.

A standard contact function: springs

- We wrote a standardized elastic adhesion (spring) function:

```
void standard_elastic_contact_function( Cell* pC1, Phenotype& p1, Cell* pC2, Phenotype& p2 , double dt )
```

- When Cell 1 evaluates the contact function, it adds an elastic interaction to the Cell 1 velocity:

$$\mathbf{v}_1 += E_1(\mathbf{x}_2 - \mathbf{x}_1)$$

- Here, E_i is stored in `Cell.phenotype.mechanics.attachment_elastic_constant`.

- When Cell 2 evaluates the contact function, it adds an elastic interaction to the Cell 2 velocity:

$$\mathbf{v}_2 += E_2(\mathbf{x}_1 - \mathbf{x}_2)$$

- Notice that these are only equal and opposite when $E_1 = E_2$. We should probably fix that.

Example 1: forming a "worm"

- Let's see if we can make cells grow chains ("worms")
 - All cells secrete signal S
 - No birth or death
 - Cells with 0 attachments chemotax to ∇S
 - Cells test for contact within interaction distance
 - ♦ If two cells are within interaction distance, and if they both have fewer than N_{\max} attachments, then attach them.
 - All attached cells interact with an elastic spring-like adhesion
- If a cell has 1 attachment:
 - ♦ Migration bias away from ∇S
 - ♦ Increase secretion of s
- If a cell has more than 1 attachment
 - ♦ No migration
 - ♦ Decrease secretion of s

Full modeling workflow

Suitable for creating a new PhysiCell model with custom C++ to drive dynamical phenotype changes

- Plan the model
- Populate a project
- Edit configuration Model Builder GUI
 - Edit domain
 - Edit microenvironment
 - Edit cell definitions
 - **Add custom variables**
 - **Add custom parameters**
- **Edit custom modules:**
 - **Declare functions in custom.h**
 - **Implement functions in custom.cpp**
 - **Assign functions to cell definitions**
- **Edit initial cell placement**
- **Edit cell coloring function**
- Build
- Run
- View results



LUDDY

SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

PhysiCell Project

PhysiCell.org

 **@PhysiCell**

Planning (1)

- Domain and Microenvironment
 - $[-400,400] \times [-400,400]$, 3000 minutes max time.
 - signal with $D = 100000$, $\lambda=10$
 - default parameters, boundary and initial conditions to 0
 - Enable virtual wall
- Custom cell data (known once you have planned your cell functions)
 - max_attachments (max number of spring links per cell)
- Cell definitions
 - worm

Planning (2)

- worm phenotype
 - set cycling and apoptosis to zero
 - set secretion of signal to 10
 - set motility on, chemotaxis towards signal
 - ♦ speed = 1, migration bias = 1, persistence time 1 min
 - [-400,400] x [-400,400], 3000 minutes max time.
 - signal with $D = 100000$, $\lambda=10$
 - default parameters, boundary and initial conditions to 0
 - Enable virtual wall

Planning (3)

- Custom migration rule:
 - 2 or more links: migration speed 0
 - 1 link: $-\text{grad}(s)$
- Phenotype rule:
 - none
- Custom rule:
 - Look for cells to attach (if $< N_{\text{max}}$)
 - Choose migration rules and secretion rate based on number of attachments
- Contact function:
 - Standard spring adhesion
- Coloring function:
 - no attachments: grey
 - 1 attachment: red
 - 2 attachments: blue
 - > 2 attachments: yellow
- `create_cell_types()`
 - Use the custom functions
- `main.cpp`
 - Use the custom coloring

Start modeling!

- populate and build the template project
 - `make template`
 - `make`
- Open Model Builder GUI
 - enter the `./PhysiCell/config` directory
 - `python ../../PhysiCell-model-builder/bin/gui4xml.py`



LUDDY

SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

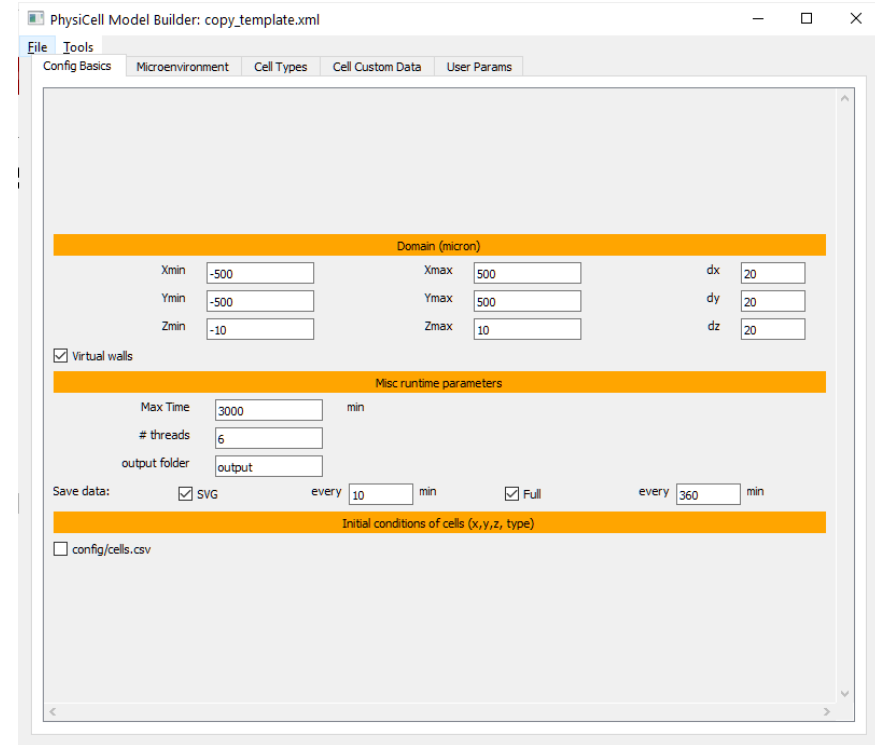
PhysiCell Project

PhysiCell.org

 [@PhysiCell](https://twitter.com/PhysiCell)

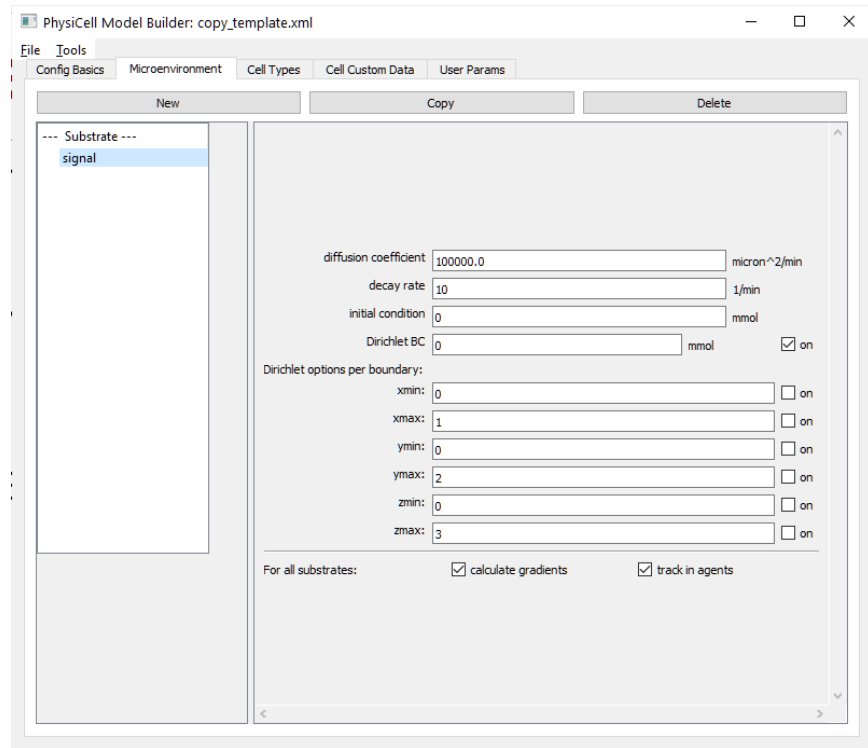
Edit the model: domain

- Go to "config basics" tab
 - max time = 3000
- full output every 360 min
- SVG every 10 min
- activate "virtual wall"
 - keep cells from leaving the domain



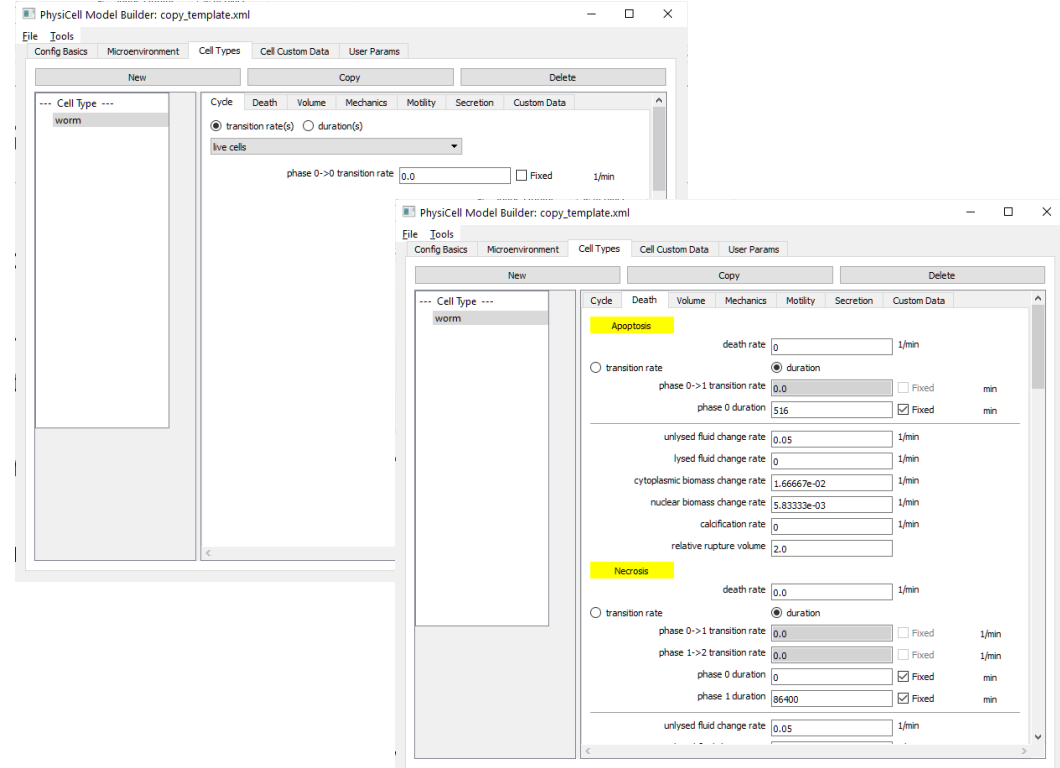
Edit the model: microenvironment

- Go to "microenvironment" tab
- double-click "substrate"
 - rename it signal
 - set Dirichlet BC to 0
 - enable the Dirichlet BC
 - set initial value to 0



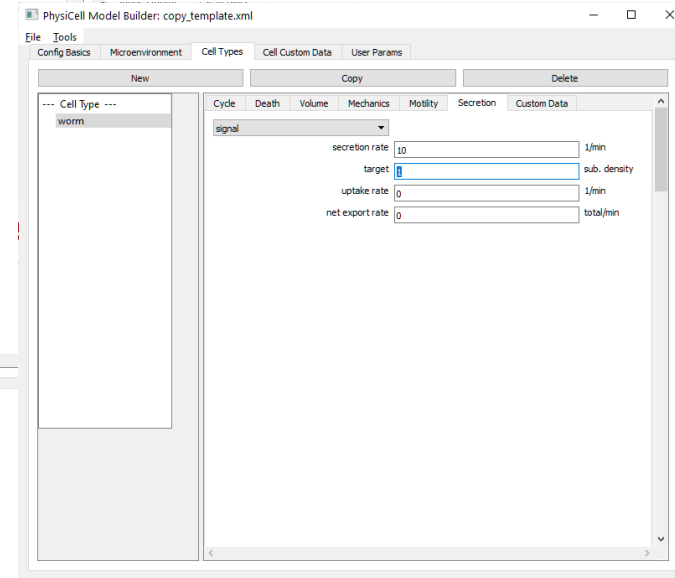
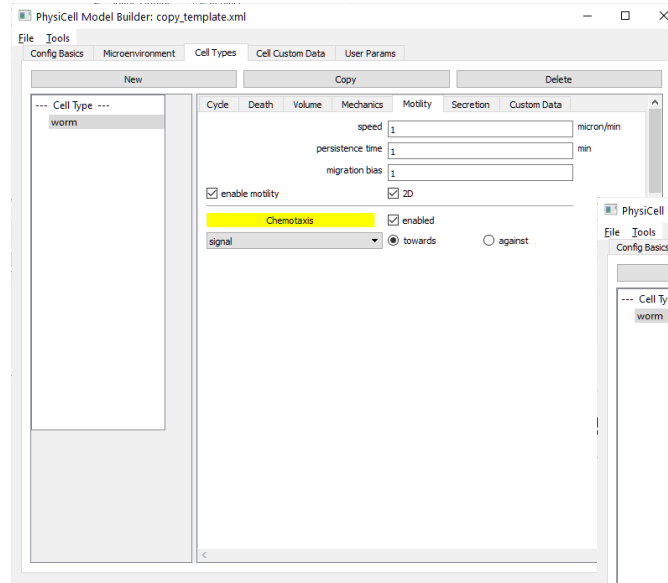
Edit the model: cell definitions (1)

- Go to "cell types" tab
- double-click "default"
 - rename it "worm"
 - edit its phenotype:
 - ◆ click "cycle" subtab
 - » choose live cycle model
 - » select "transition rate(s)"
 - » set 0→0 transition to 0
 - ◆ click "death" subtab
 - » set apoptosis to 0



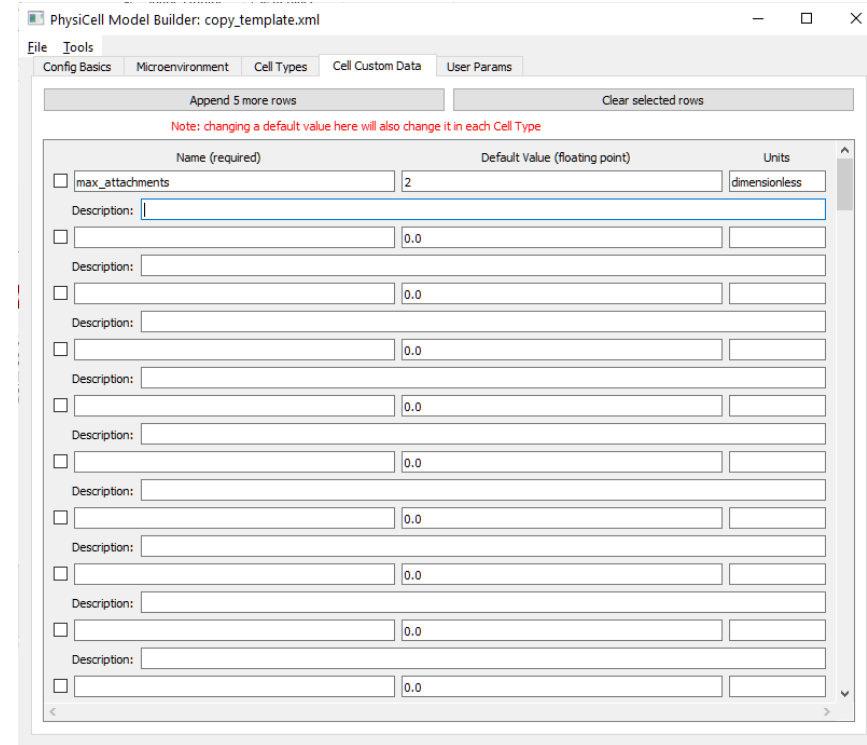
Edit the model: cell definitions (2)

- Go to "cell types" tab
- double-click "worm"
 - edit its phenotype:
 - ♦ click "motility" tab
 - » speed = 1
 - » persistence time = 1
 - » migration bias = 1
 - » check "enabled"
 - » chemotaxis "enabled"
 - towards signal
 - ♦ click "secretion" tab
 - » choose "signal"



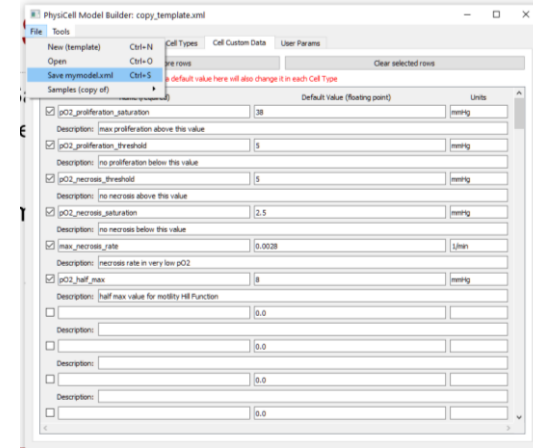
Edit the model: cell definitions (3)

- Go to "custom cell data" tab
- rename to "max_attachments"
 - set value to 2



Save to the project

- Go to "File", then "Save mymodel.xml"
 - This saves to wherever we ran PhysiCell Model Builder
- If needed, copy mymodel.xml to `./PhysiCell/config/`
- Since we ran inside the config directory, it's already there!



Unzip [Session10_checkpoint1.zip](#)
in `./PhysiCell` to get this code.

Declare custom functions

- In `./custom_modules/custom.h`, declare:

```
void stretch_migration_direction(  
    Cell* pCell, Phenotype& phenotype, double dt );  
  
void middle_migration_direction(  
    Cell* pCell, Phenotype& phenotype, double dt );  
  
void custom_worm_function( Cell*, Phenotype&, double );  
  
void worm_contact_function( Cell* pMe, Phenotype& phenoMe,  
    Cell* pThem, Phenotype &phenoThem, double dt );  
  
std::vector<std::string> worm_coloring_function( Cell* pCell );
```

Migration rules

```
// These functions are used by PhysiCell to choose a migration direction whenever  
// a cell makes a turn.
```

```
void stretch_migration_direction( Cell* pCell, Phenotype& phenotype, double dt )  
{  
    phenotype.motility.chemotaxis_direction = -1;  
  
    phenotype.motility.migration_speed = 0.2;  
    phenotype.motility.migration_bias = 0.5;  
    phenotype.motility.persistence_time = 10;  
  
    return chemotaxis_function( pCell,phenotype,dt);  
}  
  
void middle_migration_direction( Cell* pCell, Phenotype& phenotype, double dt )  
{  
    phenotype.motility.migration_speed = 0; // 0.1;  
    return;  
}
```

Custom rule (1)

```
void custom_worm_function( Cell* pCell, Phenotype& phenotype , double dt )
{
    // look for cells to form attachments
    int number_of_attachments = pCell->state.number_of_attached_cells();
    std::vector<Cell*> nearby = pCell->nearby_interacting_cells();

    int n = 0;
    while( number_of_attachments < (int) pCell->custom_data["max_attachments"] &&
           n < nearby.size() )
    {
        if( nearby[n] != pCell &&
            nearby[n]->state.number_of_attached_cells() <
            nearby[n]->custom_data["max_attachments"] )
        {
            attach_cells( nearby[n] , pCell );
            number_of_attachments++;
        }
        n++;
    }
}
```

Custom rule (2)

```
// if no attachments, use chemotaxis
if( number_of_attachments == 0 )
{ pCell->functions.update_migration_bias = chemotaxis_function; }

// if 1 attachment, use stretch
if( number_of_attachments == 1 )
{
    pCell->functions.update_migration_bias = stretch_migration_direction;
    phenotype.secretion.secretion_rates[0] = 100;
}

// if 2 or more attachments, use middle
if( number_of_attachments > 1 )
{
    pCell->functions.update_migration_bias = middle_migration_direction;
    phenotype.secretion.secretion_rates[0] = 1;
}

return;
}
```


Contact function

```
// PhysiCell has a built-in contact function for elastic spring-like attachments  
  
void worm_contact_function( Cell* pMe, Phenotype& phenoMe,  
    Cell* pOther, Phenotype& phenoOther, double dt )  
{ return standard_elastic_contact_function(pMe,phenoMe,pOther,phenoOther,dt); }
```



LUDDY

SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

PhysiCell Project

PhysiCell.org

 **@PhysiCell**

Set the cell to use these

```
void create_cell_types( void )
{
    // set the random seed
    SeedRandom( parameters.ints("random_seed") );

    // etc etc etc etc etc

    /*
       This parses the cell definitions in the XML config file.
    */

    initialize_cell_definitions_from_pugixml();

    /*
       Put any modifications to individual cell definitions here.

       This is a good place to set custom functions.
    */

    Cell_Definition* pCD = find_cell_definition("worm");

    pCD->functions.update_phenotype = NULL;
    pCD->functions.custom_cell_rule = custom_worm_function;
    pCD->functions.contact_function = worm_contact_function;

    pCD->phenotype.mechanics.attachment_elastic_constant = 0.03;
```

Coloring function

```
std::vector<std::string> worm_coloring_function( Cell* pCell )
{
    if( pCell->state.number_of_attached_cells() == 0 )
    { return { "grey", "black", "grey", "grey"}; }

    if( pCell->state.number_of_attached_cells() == 1 )
    { return { "red", "black", "red", "red"}; }

    if( pCell->state.number_of_attached_cells() == 2 )
    { return { "blue", "black", "blue", "blue"}; }

    return { "yellow", "black", "yellow", "yellow" };
}
```

Use the coloring function in main.cpp

```
// ...  
  
// for simplicity, set a pathology coloring function  
  
std::vector<std::string> (*cell_coloring_function) (Cell*) =  
    worm_coloring_function; // my_coloring_function;  
  
// ...
```

Unzip [Session10_checkpoint2.zip](#)
in ./PhysiCell to get this code.



LUDDY

SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

PhysiCell Project

PhysiCell.org

 **@PhysiCell**

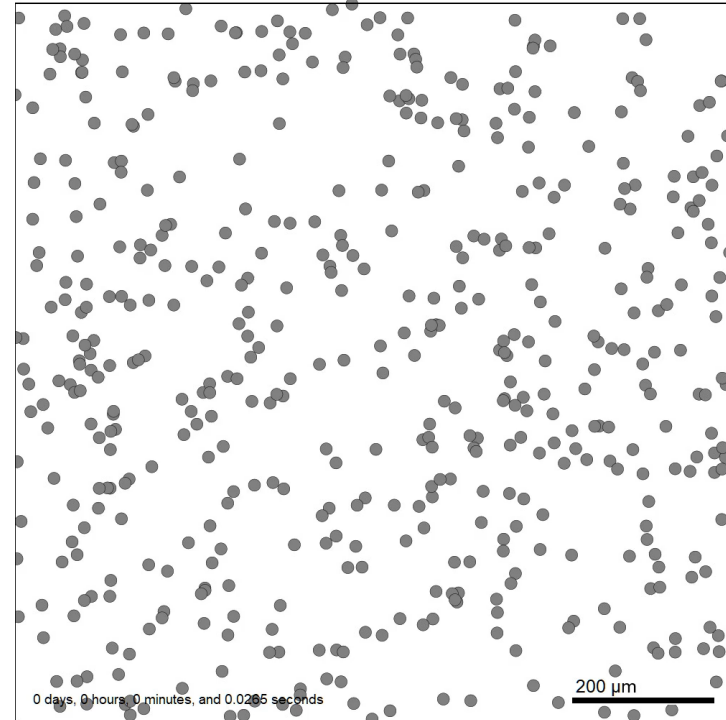
Rebuild and run the project

- Open the XML config file and set:
 - number of cells to 500
 - max time to 720 min
- rebuild:
 - **make**
- run:
 - `./project ./config/mymodel.xml` (linux, MacOS)
 - `project ./config/mymodel.xml` (Windows)

View results!

- make movie
 - **make jpeg && make movie**
- Expected behavior:
 - grays (0 attachments) move towards others
 - many cells get attached
 - ♦ red: one attachment
 - ♦ blue: two attachments
 - "worms" get stretched
 - ♦ red cells migrate away from others to stretch worms

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



Example 2: give the worm a head

- We'll use diffusion of a signaling factor along the worms help them work out heads and tails:
 - Each agent has stochastic level of "Head" protein
 - If a cell has > 1 attached cell, then diffuse Head across connections:

$$\frac{dH_i}{dt} = \sum_j r_{\text{transfer}}(H_j - H_i)$$

$$(\text{and } \frac{dH_j}{dt} = -r_{\text{transfer}}(H_j - H_i))$$

- If the cell has one attachment, don't overwrite Head value. Use initial value.
- If a cell has 1 attachment, it is at the "head" if $H_{\text{self}} > H_{\text{attached}}$

Cell migration

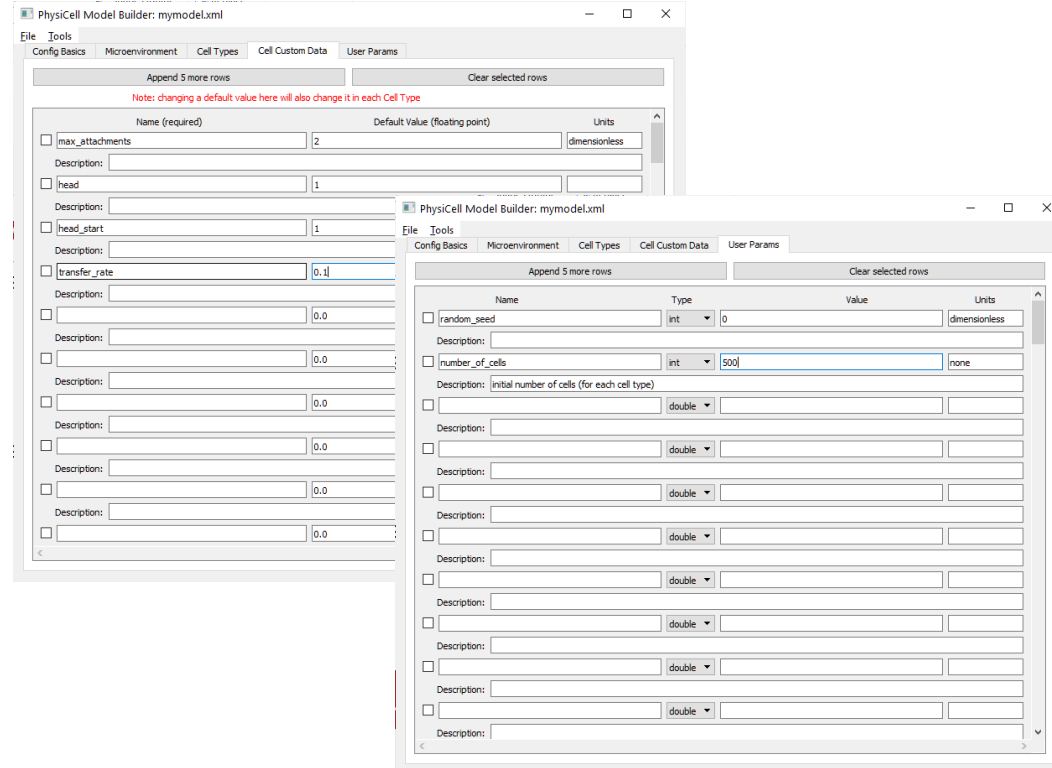
- head cells:
 - strong avoidance chemotaxis
 - They guide the worm
- tail cells:
 - they don't actively migrate. they are dragged.
- middle cells:
 - they sample "head" signal to determine gradient
 - they follow the upstream cell

Approach

- Update setup_tissue:
 - Each cell gets random value of Head
- Update the contact function:
 - If more than one attachment:
 - ♦ check each attachment
 - » If $\text{Head_me} > \text{Head_other}$, diffuse some head protein into neighbor
 - » If $\text{Head_me} \leq \text{Head_other}$, don't (This avoids double-counting the diffusion!)
- Update the custom rule:
 - Set motility function accordingly based on sensed position

Edit the model

- Go to "custom cell data" tab
 - Add
 - ♦ head (initial value 1)
 - ♦ head_start (initial value 1)
 - ♦ transfer_rate (initial value 0.1)
- Go to "user parameters" tab
 - Set initial cell count to 500



Edit setup_tissue()

```
// create some of each type of cell

Cell* pC;

for( int k=0; k < cell_definitions_by_index.size() ; k++ )
{
    Cell_Definition* pCD = cell_definitions_by_index[k];
    std::cout << "Placing cells of type " << pCD->name << " ... " << std::endl;
    for( int n = 0 ; n < parameters.ints("number_of_cells") ; n++ )
    {
        std::vector<double> position = {0,0,0};
        position[0] = Xmin + UniformRandom()*Xrange;
        position[1] = Ymin + UniformRandom()*Yrange;
        position[2] = Zmin + UniformRandom()*Zrange;

        pC = create_cell( *pCD );
        pC->assign_position( position );

        pC->custom_data["head"] = UniformRandom();
        pC->custom_data["head_start"] = pC->custom_data["head"];
    }
    std::cout << std::endl;

    // load cells from your CSV file (if enabled)
    load_cells_from_pugixml();

    return;
}
```

Update function declarations in custom.h

```
void custom_worm_function( Cell*, Phenotype&, double );  
void worm_contact_function( Cell* pMe, Phenotype& phenoMe,  
    Cell* pThem, Phenotype &phenoThem, double dt );
```

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

```
void head_migration_direction( Cell* pCell, Phenotype& phenotype, double dt );  
void tail_migration_direction( Cell* pCell, Phenotype& phenotype, double dt );  
void middle_migration_direction( Cell* pCell, Phenotype& phenotype , double dt );
```

edit contact function

```
void worm_contact_function( Cell* pMe, Phenotype& phenoMe,
    Cell* pOther, Phenotype& phenoOther, double dt )
{
    standard_elastic_contact_function(pMe,phenoMe,pOther,phenoOther,dt);

    if( pMe->state.number_of_attached_cells() > 1 )
    {
        double head_me = pMe->custom_data["head"];
        double head_other = pOther->custom_data["head"];

        // make the trnasfer
        if( head_me > head_other )
        {
            double amount_to_transfer = dt * pMe->custom_data["transfer_rate"]
                * (head_me - head_other );
            pMe->custom_data["head"] -= amount_to_transfer;
            #pragma omp critical
            { pOther->custom_data["head"] += amount_to_transfer; }
        }
    }
}
```



LUDDY

SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

PhysiCell Project

PhysiCell.org

@PhysiCell

edit custom function (1)

```
void custom_worm_function( Cell* pCell, Phenotype& phenotype , double dt )
{
    // bookkeeping

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

    // look for cells to form attachments, if 0 attachments
    int number_of_attachments = pCell->state.number_of_attached_cells();
    std::vector<Cell*> nearby = pCell->nearby_interacting_cells();

    if( number_of_attachments == 0 )
    {
        int n = 0;
        while( number_of_attachments < (int) pCell->custom_data["max_attachments"] && n < nearby.size() )
        {
            if( nearby[n]->state.number_of_attached_cells() < nearby[n]->custom_data["max_attachments"] )
            {
                attach_cells( nearby[n] , pCell );
                number_of_attachments++;
            }
            n++;
        }
    }

    // if no attachments, use chemotaxis
    if( number_of_attachments == 0 )
    { pCell->functions.update_migration_bias = chemotaxis_function; }
```

edit custom function (2)

```
// if 1 attachment, do some logic
if( number_of_attachments == 1 )
{
    // constant expression in end cells
    pCell->custom_data["head"] = pCell->custom_data["head_start"];

    // am I the head?
    bool head = false;
    if( pCell->custom_data["head"] > pCell->state.attached_cells[0]->custom_data["head"] )
    { head = true; }

    if( head )
    { pCell->functions.update_migration_bias = head_migration_direction; }
    else
    { pCell->functions.update_migration_bias = tail_migration_direction; }
    phenotype.secretion.secretion_rates[nSignal] = 100;
}

// if 2 or more attachments, use middle
if( number_of_attachments > 1 )
{
    pCell->functions.update_migration_bias = middle_migration_direction;
    phenotype.secretion.secretion_rates[nSignal] = 1;
}

return;
}
```

migration bias functions

```
void head_migration_direction( Cell* pCell, Phenotype& phenotype, double dt )
{
    phenotype.motility.chemotaxis_direction = -1;
    phenotype.motility.migration_speed = 0.75;
    phenotype.motility.migration_bias = 0.5;
    phenotype.motility.persistence_time = 60;

    return chemotaxis_function( pCell,phenotype,dt);
}

void tail_migration_direction( Cell* pCell, Phenotype& phenotype, double dt )
{
    phenotype.motility.chemotaxis_direction = -1;
    phenotype.motility.migration_speed = 0;
    phenotype.motility.migration_bias = 0.5;
    phenotype.motility.persistence_time = 100;

    return chemotaxis_function( pCell,phenotype,dt);
}

void middle_migration_direction( Cell* pCell, Phenotype& phenotype , double dt )
{
    // get velocity from "Upstream"
    Cell* pUpstream = pCell->state.attached_cells[0];

    if( pCell->state.attached_cells[1]->custom_data["head"] >
        pCell->state.attached_cells[0]->custom_data["head"] )
    { pUpstream = pCell->state.attached_cells[1]; }
}
```


edit coloring

```
// only head cell is red
```

```
std::vector<std::string> worm_coloring_function( Cell* pCell )
{
    if( pCell->state.number_of_attached_cells() == 0 )
    { return { "grey", "black", "grey", "grey"}; }

    if( pCell->state.number_of_attached_cells() == 1 &&
        pCell->custom_data["head"] > pCell->state.attached_cells[0]->custom_data["head"] )
    { return { "red", "black", "red", "red"}; }

    if( pCell->state.number_of_attached_cells() == 2 )
    { return { "blue", "black", "blue", "blue"}; }

    return { "yellow", "black", "yellow", "yellow" };
}
```

Rebuild and run the project

- Open the XML config file and set:

- number of cells to 500
- max time to 1440 min

- rebuild:

- **make**

Unzip [Session10_checkpoint3.zip](#)
in ./PhysiCell to get this code.

- run:

- `./project ./config/mymodel.xml` (linux, MacOS)
- `project ./config/mymodel.xml` (Windows)

View results!

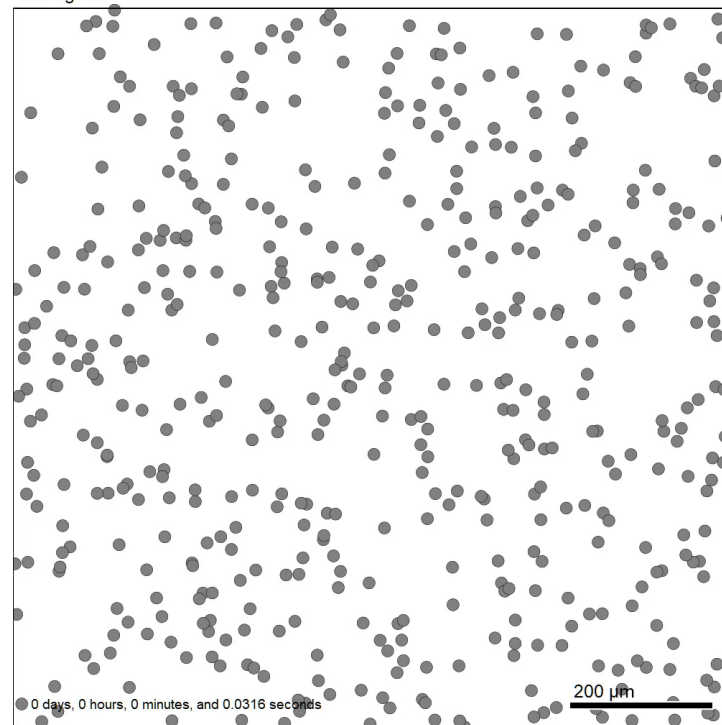
- make movie

- make jpeg && make movie

- Expected behavior:

- grays (0 attachments) move towards others
 - many cells get attached
 - ♦ blue: one attachment
 - ♦ red: two attachments, head
 - ♦ yellow: two attachments, tail
 - "worms" get stretched
 - ♦ red cells migrate away from others to stretch worms
 - ♦ red cells lead the worm

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



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)