Slides, videos, links and more:
https://github.com/physicell-training/cecam23

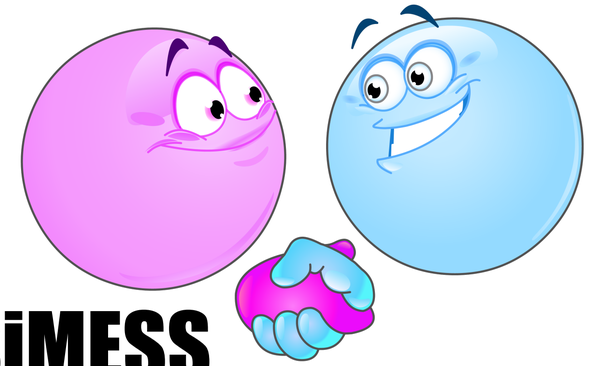
Introducing: PhysiMESS

Agent-based modelling of the ECM

Cicely Macnamara
@CicelyKrystyna

PhysiCell Hackathon
July 2021

March 7, 2023



PhysiMESS
a PhysiCell tool/add-on

LUDDY
SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

PhysiCell Project
PhysiCell.org
@PhysiCell

PhysiMESS Team

PhysiMESS (MicroEnvironment Structures Simulator)

Dr Cicely Macnamara
University of Glasgow

Dr Robyn Shuttleworth
University of Saskatchewan

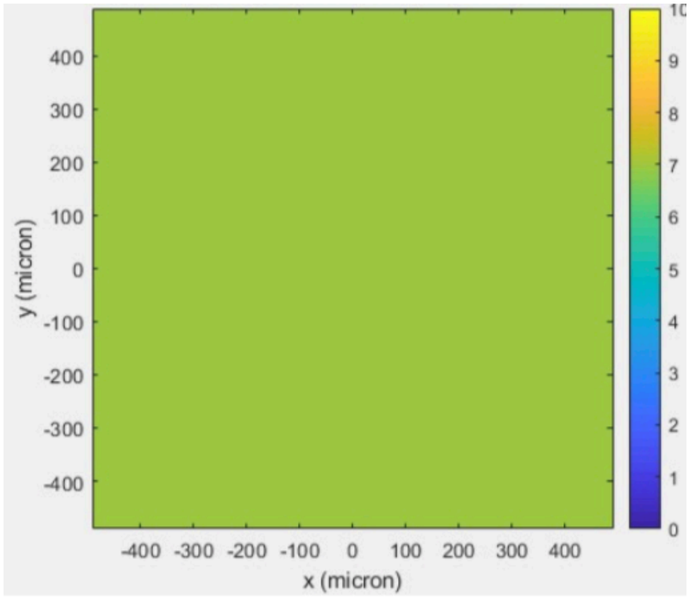
Dr Vincent Noël
Institut Curie

Marco Ruscone
Institut Curie

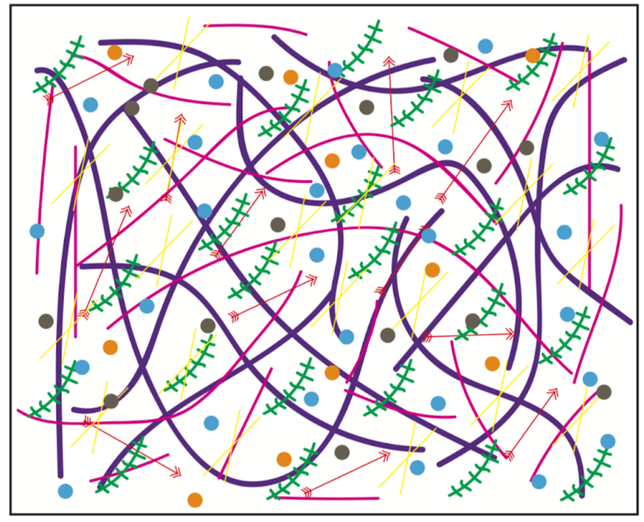
Team Members: Connah Johnson, Jamey O'Neill, Niki Tavakoli
John Metzcar, **Zoe Bell**, **Temitope Benson, Joseph Abrams**

Background: Scientific Question

Can we write a generalised framework to model realistic tissue microenvironments?



- Collagen Type I
- Collagen Type IV
- Fibronectin
- Elastin
- Laminin
- MMPs



Looking to a future where standard models of the ECM move away from a coarse grained continuum towards a fine grained agent-based approach

Design Goals: Features

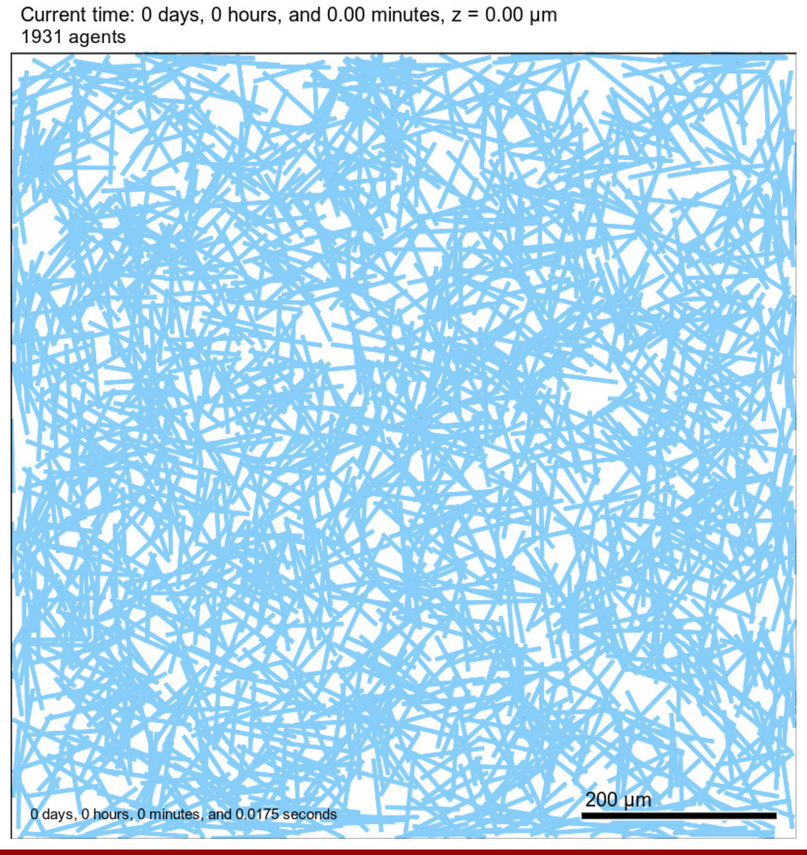
- PhysiMESS **must:**
  - be able to create and model individual structures of the microenvironment
  - allow for interactions between cells, ECM structures and underlying components
  - be fully adaptable to allow for modelling of **any** biological system
  - **be user friendly!**

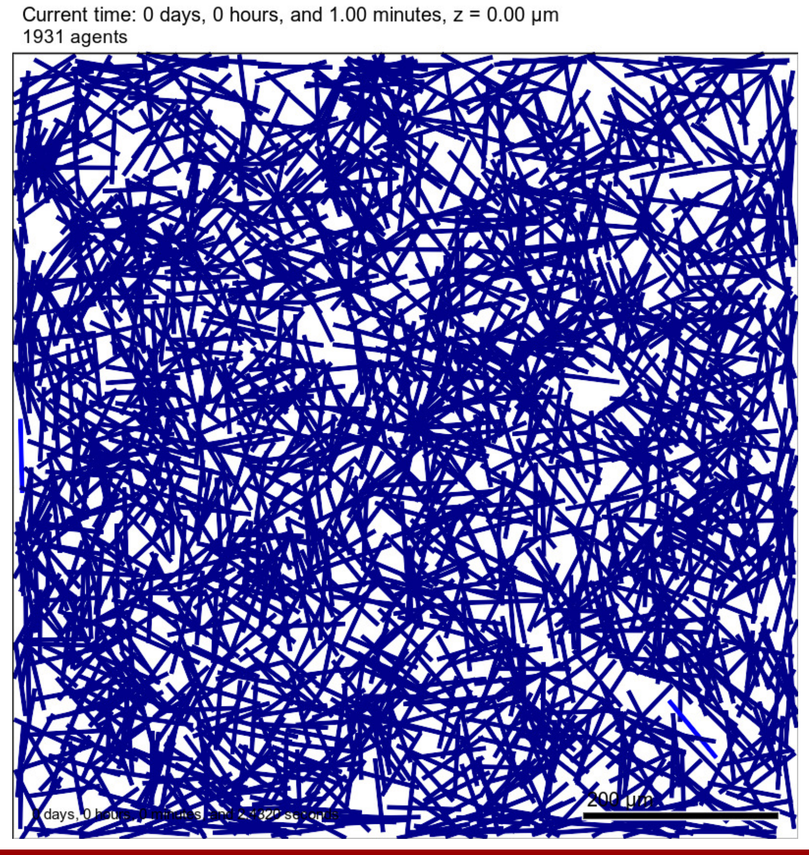
- Please note:
  - PhysiMESS was built originally built on PhysiCell v1.10.1
  - the cecam23 training repo was built on PhysiCell v1.10.4
  - some PhysiCell capabilities may not yet be supported.
  - PhysiMESS is as yet unpublished please use responsibly.

… what you did earlier

```
make clean; make
./project ./config/mymodel.xml
make jpeg
```

- This should create two files snapshot00000000.jpg & snapshot00000001.jpg





Note: if you can't use the model builder GUI you can use the pre-loaded mymodel.xml file in the ./PhysiMESS/config directory.

- You're now ready to play with PhysiMESS!!!

Fibre Initialisation

Fibres are cylindrical **agents** given by their centre, radius (default 2 microns); length (default 75 microns) and orientation
They are visualised for the time being as thin rectangles in 2D.

Initialising fibres
in list Code

Description Edit

In **custom.cpp** in function `set_up_tissue`

custom.cpp

```
void setup_tissue( void ){...}
```

Any of these agent names will signal for a cylindrical/ rod-like agent →

```
const auto agentname = std::string(pCD->name);
const auto ecm = std::string( s: "ecm");
const auto matrix = std::string( s: "matrix");
const auto fiber = std::string( s: "fiber");
const auto fibre = std::string( s: "fibre");
const auto rod = std::string( s: "rod");
```

```
(*all_cells)[i]->type_name == "fibre_vertical")
```

```
(*all_cells)[i]->type_name == "fibre_horizontal")
```

Parameters

```
<number_of_fibres type="int" units="none" description="initial number of fibres (for each fibre type)">2000</number_of_fibres>
<anisotropic_fibres type="bool" units="none" description="flag for whether we want anisotropic fibres">false</anisotropic_fibres>
<fibre_length type="double" units="microns" description="length of fibres">75.0</fibre_length>
<length_normdist_sd type="double" units="microns" description="standard deviation of fibre length">0.0</length_normdist_sd>
<fibre_radius type="double" units="microns" description="radius of fibres">2.0</fibre_radius>
<fibre_angle type="double" units="radians" description="angle of fibre orientation">0.0</fibre_angle>
<angle_normdist_sd type="double" units="radians" description="standard deviation of fibre orientation angle">0.0</angle_normdist_sd>
```

custom.cpp

```
void setup_tissue( void ){...}
```

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

// new fibre related parameters and bools
bool isFibreFromFile = false;
bool fibreanisotropy = parameters.bools("anisotropic_fibres");
double fibre_length = parameters.doubles("fibre_length");
double length_normdist_sd = parameters.doubles("length_normdist_sd");
double fibre_radius = parameters.doubles("fibre_radius");
double fibre_angle = parameters.doubles("fibre_angle");
double angle_normdist_sd = parameters.doubles("angle_normdist_sd");

for( int i=0; i < (*all_cells).size(); i++ ){

    // initialise the following parameters for all cells regardless of type
    (*all_cells)[i]->parameters.mCellVelocityMaximum = parameters.doubles("cell_velocity_max");
    (*all_cells)[i]->parameters.mVelocityAdhesion = parameters.doubles("vel_adhesion");
    (*all_cells)[i]->parameters.mVelocityContact = parameters.doubles("vel_contact");


    (*all_cells)[i]->parameters.fibreDegradationRate = parameters.doubles("fibre_deg_rate");
    (*all_cells)[i]->parameters.stuck_threshold = parameters.doubles("fibre_stuck");
    (*all_cells)[i]->parameters.fibre_degradation = parameters.bools("fibre_degradation");

    (*all_cells)[i]->parameters.fibre_pushing = parameters.bools("fibre_pushing");
    (*all_cells)[i]->parameters.fibre_rotation = parameters.bools("fibre_rotation");
    (*all_cells)[i]->parameters.mFibreStickiness = parameters.doubles("fibre_sticky");

    (*all_cells)[i]->state.crosslink_point.resize(3,0.0);
```

```
const auto agentname = std::string((*all_cells)[i]->type_name);
const auto ecm = std::string( s: "ecm");
const auto matrix = std::string( s: "matrix");
const auto fiber = std::string( s: "fiber");
const auto fibre = std::string( s: "fibre");
const auto rod = std::string( s: "rod");

if (agentname.find(ecm) != std::string::npos ||
    agentname.find(matrix) != std::string::npos ||
    agentname.find(fiber) != std::string::npos ||
    agentname.find(fibre) != std::string::npos ||
    agentname.find(rod) != std::string::npos) {
    /* fibre positions are given by csv
       assign fibre orientation and test whether out of bounds */
    isFibreFromFile = true;

    (*all_cells)[i]->parameters.mLength = NormalRandom(fibre_length, length_normdist_sd) / 2.0;
    (*all_cells)[i]->parameters.mRadius = fibre_radius;

    //assign fibre orientation as a random vector from points on unit sphere/circle
    (*all_cells)[i]->assign_orientation();
```

```
// start and end points of a fibre are calculated from fibre center
double xs = (*all_cells)[i]->position[0] -
            (*all_cells)[i]->parameters.mLength * (*all_cells)[i]->state.orientation[0];
double xe = (*all_cells)[i]->position[0] +
            (*all_cells)[i]->parameters.mLength * (*all_cells)[i]->state.orientation[0];
double ys = (*all_cells)[i]->position[1] -
            (*all_cells)[i]->parameters.mLength * (*all_cells)[i]->state.orientation[1];
double ye = (*all_cells)[i]->position[1] +
            (*all_cells)[i]->parameters.mLength * (*all_cells)[i]->state.orientation[1];
```

```
// relabel so that the rest of the code works
(*all_cells)[i]->type_name = "fibre";
```

Visualising fibres using the SVG

in list Code

Description Edit

In **PhysiCell_pathology.cpp** in function `SVG_plot`

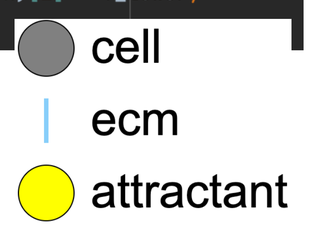
Fibres are visualised as thin rectangles rather than spheres. We plot fibres in different colours depending on the number of crosslinks.

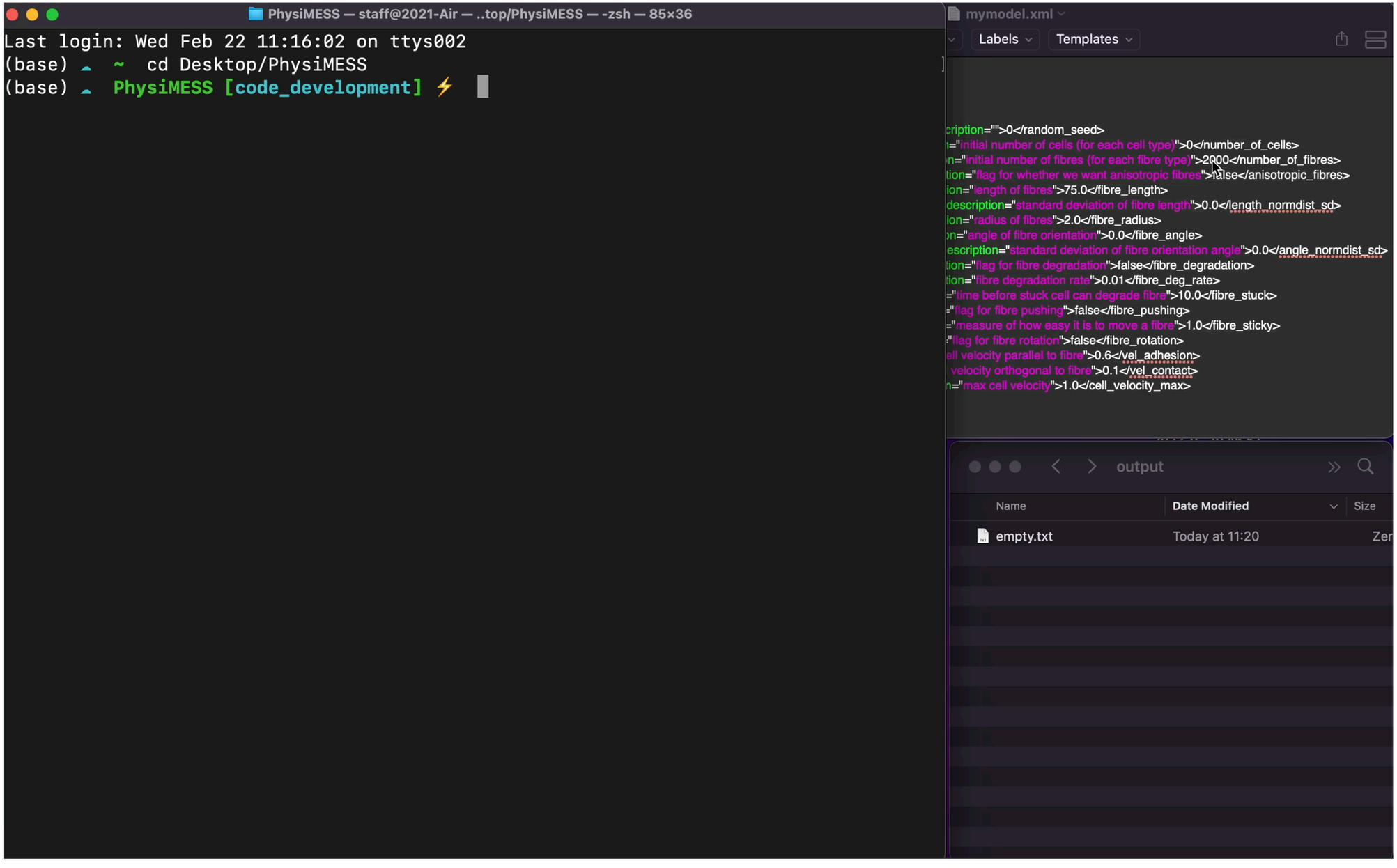
In **PhysiCell_pathology.cpp** in function `create_plot_legend`

Fibre rods are added to the plot legend.

```
// place a rod if it's a fibre (note fibre not yet renamed)
const auto agentname = std::string(C.type_name);
const auto ecm = std::string( s: "ecm");
const auto matrix = std::string( s: "matrix");
const auto fiber = std::string( s: "fiber");
const auto fibre = std::string( s: "fibre");
const auto rod = std::string( s: "rod");

if (agentname.find(ecm) != std::string::npos ||
    agentname.find(matrix) != std::string::npos ||
    agentname.find(fiber) != std::string::npos ||
    agentname.find(fibre) != std::string::npos ||
    agentname.find(rod) != std::string::npos) {
        //Write_SVG_fibre(os, cursor_x, cursor_y , 0.5*temp_cell_radius , 1.0 , colors[1] , colors[0] );
        Write_SVG_line( &: os, cursor_x, start_y: cursor_y-20.0 , cursor_x , end_y: cursor_y+20.0 , thickness: 4.0 , stroke_color: "lightskyblue" );
}
else {
    // place a big circle with cytoplasm colors
    Write_SVG_circle( &: os, cursor_x, cursor_y, temp_cell_radius, stroke_size: 1.0, stroke_color: colors[1], fill_color: colors[0]);
    // place a small circle with nuclear colors
    //Write_SVG_circle(os, cursor_x, cursor_y, 0.5 * temp_cell_radius, 1.0, colors[2], colors[3]);
}
```

```
// place a rod if it's a fibre (note fibre already renamed here)
const auto agentname = std::string(pC->type_name);
const auto fibre = std::string( s: "fibre");

if (agentname.find(fibre) != std::string::npos ){

    int crosslinks = pC->parameters.X_crosslink_count;
    if (crosslinks >= 3){
        // if fibre has cross-links different colour than if not
        Write_SVG_line( &: os, start_x: (pC->position)[0] - (pC->parameters.mLength) * (pC->state.orientation)[0] - X_lower,
                        start_y: (pC->position)[1] - (pC->parameters.mLength) * (pC->state.orientation)[1] - Y_lower,
                        end_x: (pC->position)[0] + (pC->parameters.mLength) * (pC->state.orientation)[0] - X_lower,
                        end_y: (pC->position)[1] + (pC->parameters.mLength) * (pC->state.orientation)[1] - Y_lower,
                        thickness: 4.0, stroke_color: "darkblue");
```

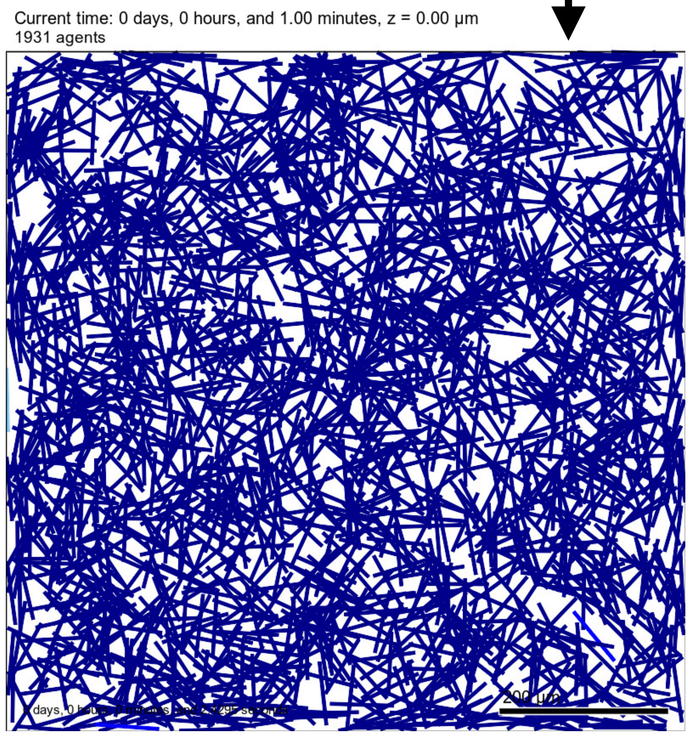


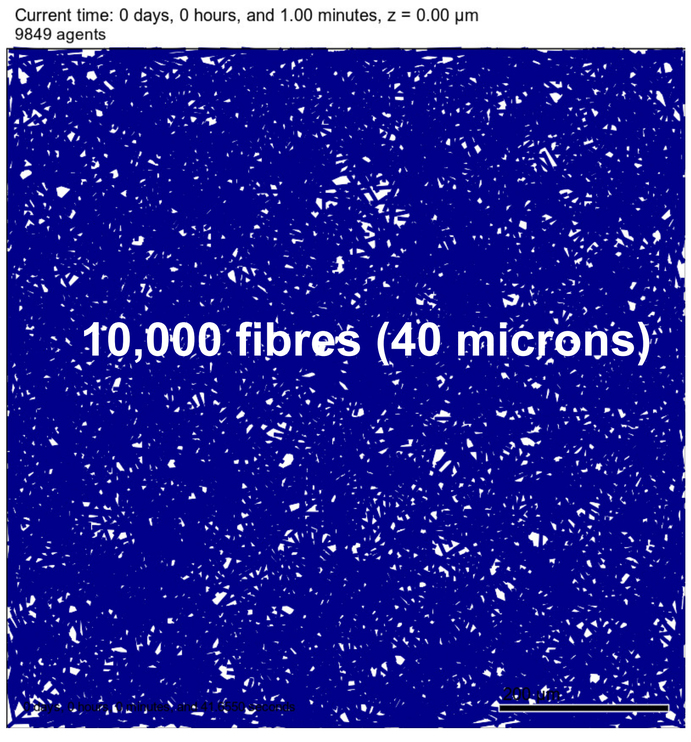

PhysiMESS — staff@2021-Air — ..top/PhysiMESS — -zsh — 85×36
Last login: Wed Feb 22 11:16:02 on ttys002
(base) ~ cd Desktop/PhysiMESS
(base) PhysiMESS [code_development]
mymodel.xml
Labels
Templates
cription="">0</random_seed>
="initial number of cells (for each cell type)">0</number_of_cells>
n="initial number of fibres (for each fibre type)">2000</number_of_fibres>
tion="flag for whether we want anisotropic fibres">false</anisotropic_fibres>
ion="length of fibres">75.0</fibre_length>
description="standard deviation of fibre length">0.0</length_normdist_sd>
ion="radius of fibres">2.0</fibre_radius>
n="angle of fibre orientation">0.0</fibre_angle>
escription="standard deviation of fibre orientation angle">0.0</angle_normdist_sd>
tion="flag for fibre degradation">false</fibre_degradation>
tion="fibre degradation rate">0.01</fibre_deg_rate>
="time before stuck cell can degrade fibre">10.0</fibre_stuck>
="flag for fibre pushing">false</fibre_pushing>
="measure of how easy it is to move a fibre">1.0</fibre_sticky>
"flag for fibre rotation">false</fibre_rotation>
ell velocity parallel to fibre">0.6</vel_adhesion>
velocity orthogonal to fibre">0.1</vel_contact>
n="max cell velocity">1.0</cell_velocity_max>
output
Name
Date Modified
Size
empty.txt
Today at 11:20
Zer

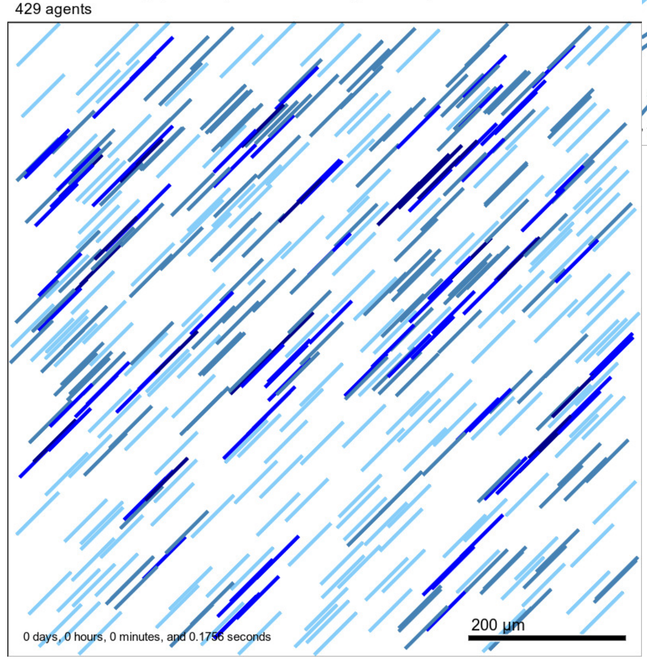
Fibre Initialisation

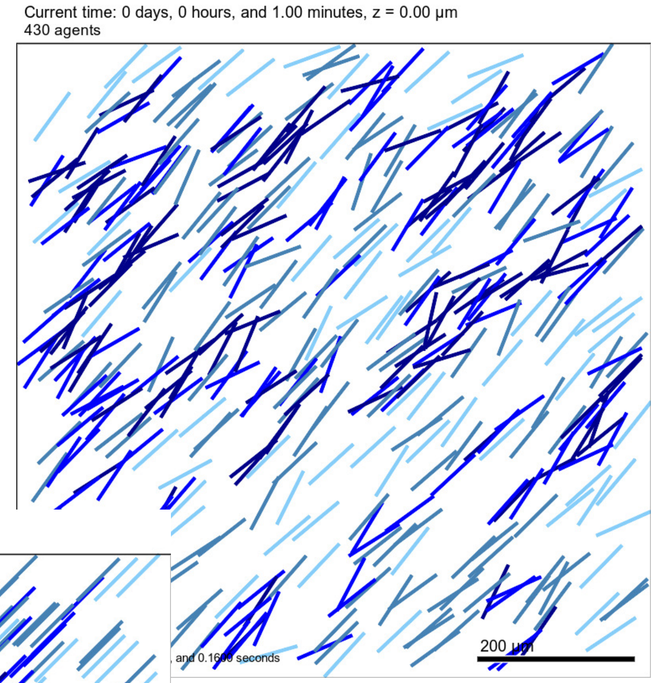
Parameters

```
<number_of_fibres type="int" units="none" description="initial number of fibres (for each fibre type)">2000</number_of_fibres>
<anisotropic_fibres type="bool" units="none" description="flag for whether we want anisotropic fibres">false</anisotropic_fibres>
<fibre_length type="double" units="microns" description="length of fibres">75.0</fibre_length>
<length_normdist_sd type="double" units="microns" description="standard deviation of fibre length">0.0</length_normdist_sd>
<fibre_radius type="double" units="microns" description="radius of fibres">2.0</fibre_radius>
<fibre_angle type="double" units="radians" description="angle of fibre orientation">0.0</fibre_angle>
<angle_normdist_sd type="double" units="radians" description="standard deviation of fibre orientation angle">0.0</angle_normdist_sd>
```





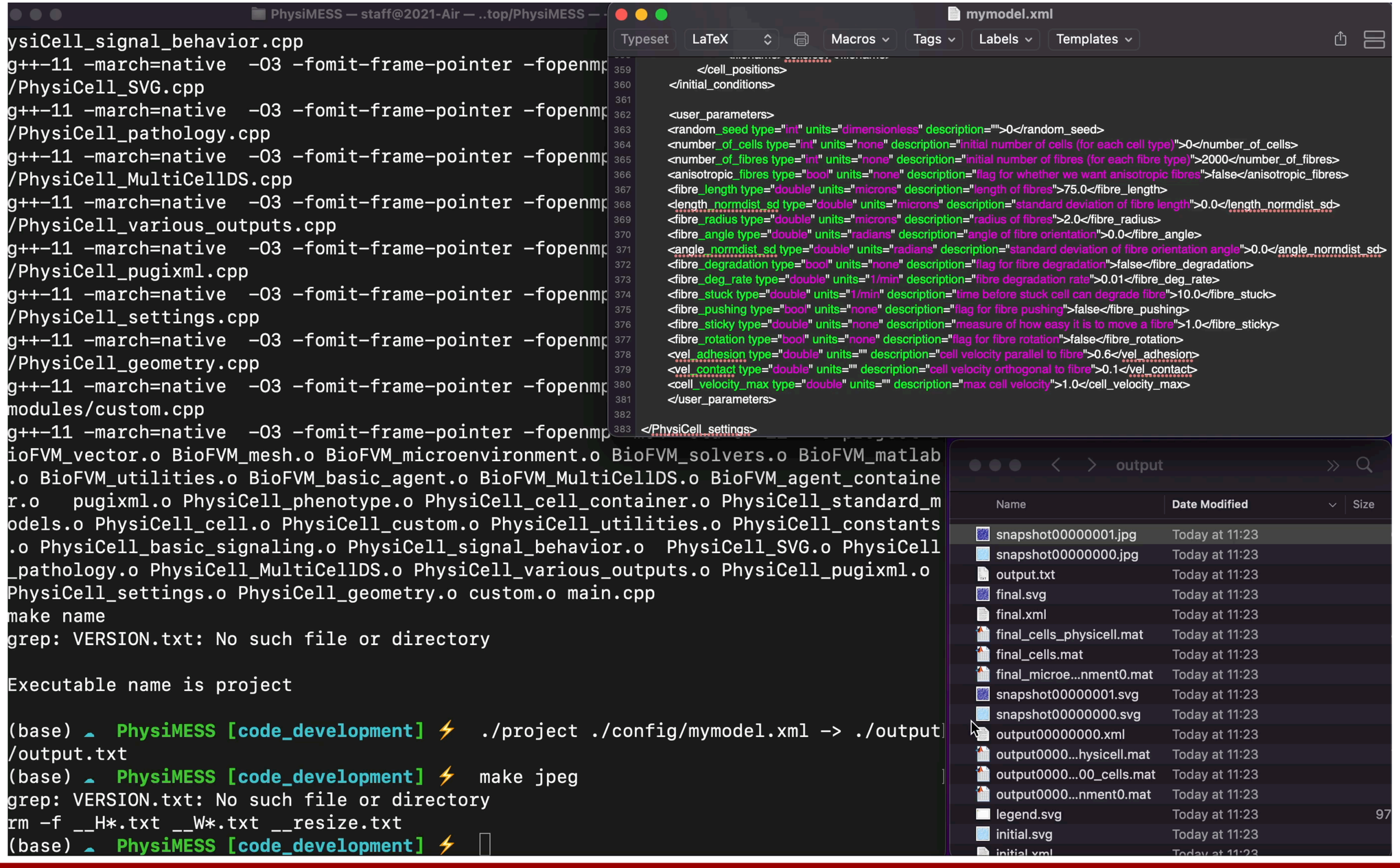




with variance of 0.2 radians

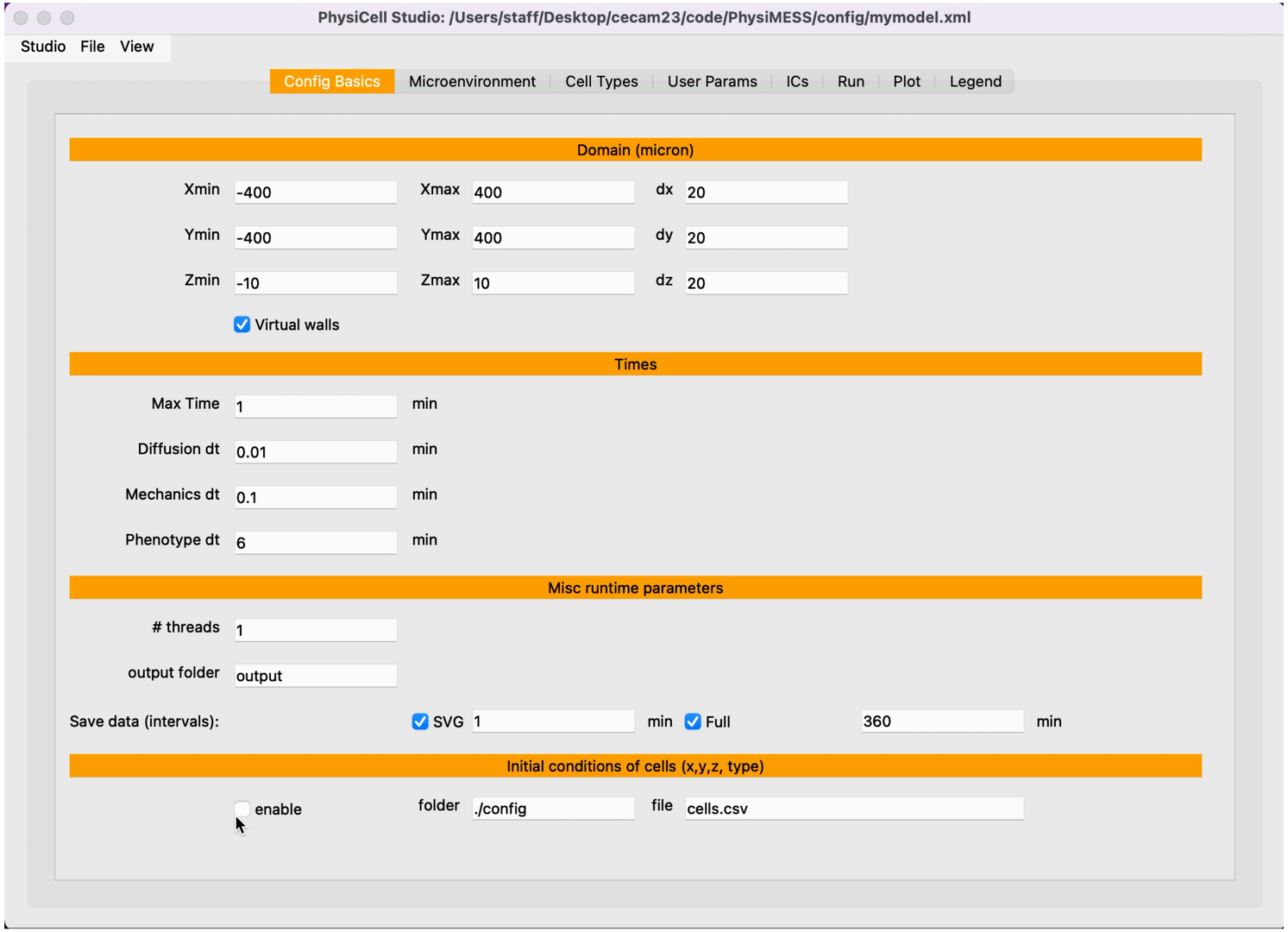
anisotropic

500 fibres (0.785 radians orientation)


PhysiMESS — staff@2021-Air — ..top/PhysiMESS —
ysiCell_signal_behavior.cpp
g++-11 -march=native -O3 -fomit-frame-pointer -fopenmp
/PhysiCell_SVG.cpp
g++-11 -march=native -O3 -fomit-frame-pointer -fopenmp
/PhysiCell_pathology.cpp
g++-11 -march=native -O3 -fomit-frame-pointer -fopenmp
/PhysiCell_MultiCellDS.cpp
g++-11 -march=native -O3 -fomit-frame-pointer -fopenmp
/PhysiCell_various_outputs.cpp
g++-11 -march=native -O3 -fomit-frame-pointer -fopenmp
/PhysiCell_pugixml.cpp
g++-11 -march=native -O3 -fomit-frame-pointer -fopenmp
/PhysiCell_settings.cpp
g++-11 -march=native -O3 -fomit-frame-pointer -fopenmp
/PhysiCell_geometry.cpp
g++-11 -march=native -O3 -fomit-frame-pointer -fopenmp
modules/custom.cpp
g++-11 -march=native -O3 -fomit-frame-pointer -fopenmp
ioFVM_vector.o BioFVM_mesh.o BioFVM_microenvironment.o BioFVM_solvers.o BioFVM_matlab
.o BioFVM_utilities.o BioFVM_basic_agent.o BioFVM_MultiCellDS.o BioFVM_agent_containe
r.o pugixml.o PhysiCell_phenotype.o PhysiCell_cell_container.o PhysiCell_standard_m
odels.o PhysiCell_cell.o PhysiCell_custom.o PhysiCell_utilities.o PhysiCell_constants
.o PhysiCell_basic_signaling.o PhysiCell_signal_behavior.o PhysiCell_SVG.o PhysiCell
_pathology.o PhysiCell_MultiCellDS.o PhysiCell_various_outputs.o PhysiCell_pugixml.o
PhysiCell_settings.o PhysiCell_geometry.o custom.o main.cpp
make name
grep: VERSION.txt: No such file or directory
Executable name is project
(base) PhysiMESS [code_development] ./project ./config/mymodel.xml -> ./output
/output.txt
(base) PhysiMESS [code_development] make jpeg
grep: VERSION.txt: No such file or directory
rm -f __H*.txt __W*.txt __resize.txt
(base) PhysiMESS [code_development]
mymodel.xml
Typeset
LaTeX
Macros
Tags
Labels
Templates
</cell_positions>
</initial_conditions>
<user_parameters>
<random_seed type="int" units="dimensionless" description="">0</random_seed>
<number_of_cells type="int" units="none" description="initial number of cells (for each cell type)">0</number_of_cells>
<number_of_fibres type="int" units="none" description="initial number of fibres (for each fibre type)">2000</number_of_fibres>
<anisotropic_fibres type="bool" units="none" description="flag for whether we want anisotropic fibres">false</anisotropic_fibres>
<fibre_length type="double" units="microns" description="length of fibres">75.0</fibre_length>
<length_normdist_sd type="double" units="microns" description="standard deviation of fibre length">0.0</length_normdist_sd>
<fibre_radius type="double" units="microns" description="radius of fibres">2.0</fibre_radius>
<fibre_angle type="double" units="radians" description="angle of fibre orientation">0.0</fibre_angle>
<angle_normdist_sd type="double" units="radians" description="standard deviation of fibre orientation angle">0.0</angle_normdist_sd>
<fibre_degradation type="bool" units="none" description="flag for fibre degradation">false</fibre_degradation>
<fibre_deg_rate type="double" units="1/min" description="fibre degradation rate">0.01</fibre_deg_rate>
<fibre_stuck type="double" units="1/min" description="time before stuck cell can degrade fibre">10.0</fibre_stuck>
<fibre_pushing type="bool" units="none" description="flag for fibre pushing">false</fibre_pushing>
<fibre_sticky type="double" units="none" description="measure of how easy it is to move a fibre">1.0</fibre_sticky>
<fibre_rotation type="bool" units="none" description="flag for fibre rotation">false</fibre_rotation>
<vel_adhesion type="double" units="" description="cell velocity parallel to fibre">0.6</vel_adhesion>
<vel_contact type="double" units="" description="cell velocity orthogonal to fibre">0.1</vel_contact>
<cell_velocity_max type="double" units="" description="max cell velocity">1.0</cell_velocity_max>
</user_parameters>
</PhysiCell_settings>
output
Name
Date Modified
Size
snapshot00000001.jpg Today at 11:23
snapshot00000000.jpg Today at 11:23
output.txt Today at 11:23
final.svg Today at 11:23
final.xml Today at 11:23
final_cells_physicell.mat Today at 11:23
final_cells.mat Today at 11:23
final_microe...nment0.mat Today at 11:23
snapshot00000001.svg Today at 11:23
snapshot00000000.svg Today at 11:23
output00000000.xml Today at 11:23
output0000...hysicell.mat Today at 11:23
output0000...00_cells.mat Today at 11:23
output0000...nment0.mat Today at 11:23
legend.svg Today at 11:23 97
initial.svg Today at 11:23

mymodel_initialisation.xml
initialfibres.csv

Fibre Initialisation

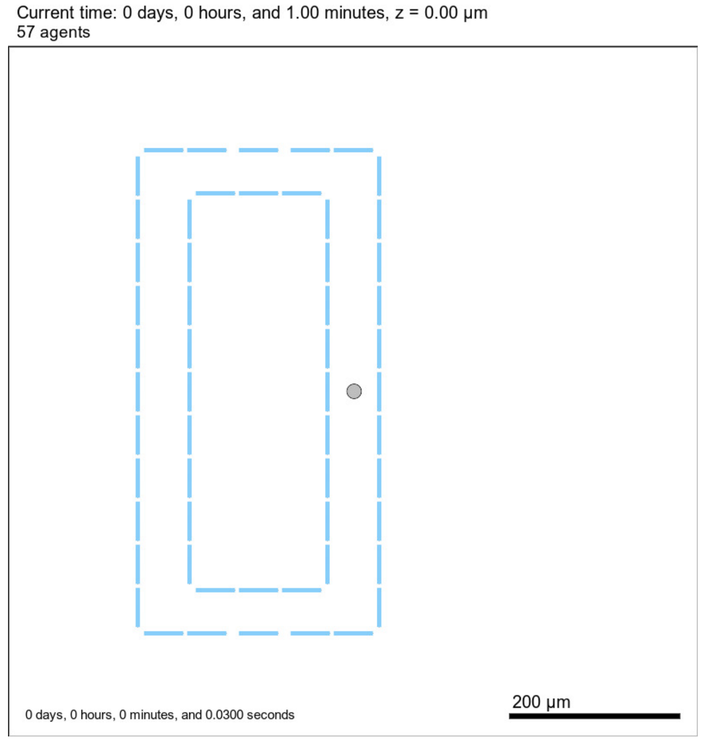


enable csv input

```
<cell_positions type="csv" enabled="true">
    <folder>./config/Fibre_Initialisation</folder>
    <filename>initialfibres.csv</filename>
```

```
<cell_definition name="fibre_horizontal" ID="2">
```

Anisotropic 1.57 radians 40 microns



Cell-Fibre Mechanics

The potential function

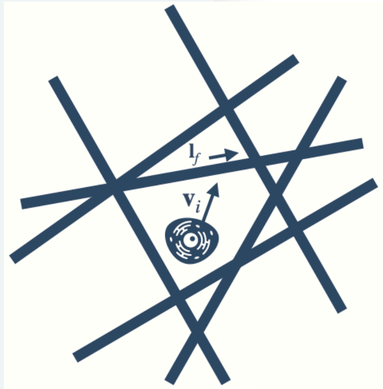
in list Code

Description Edit

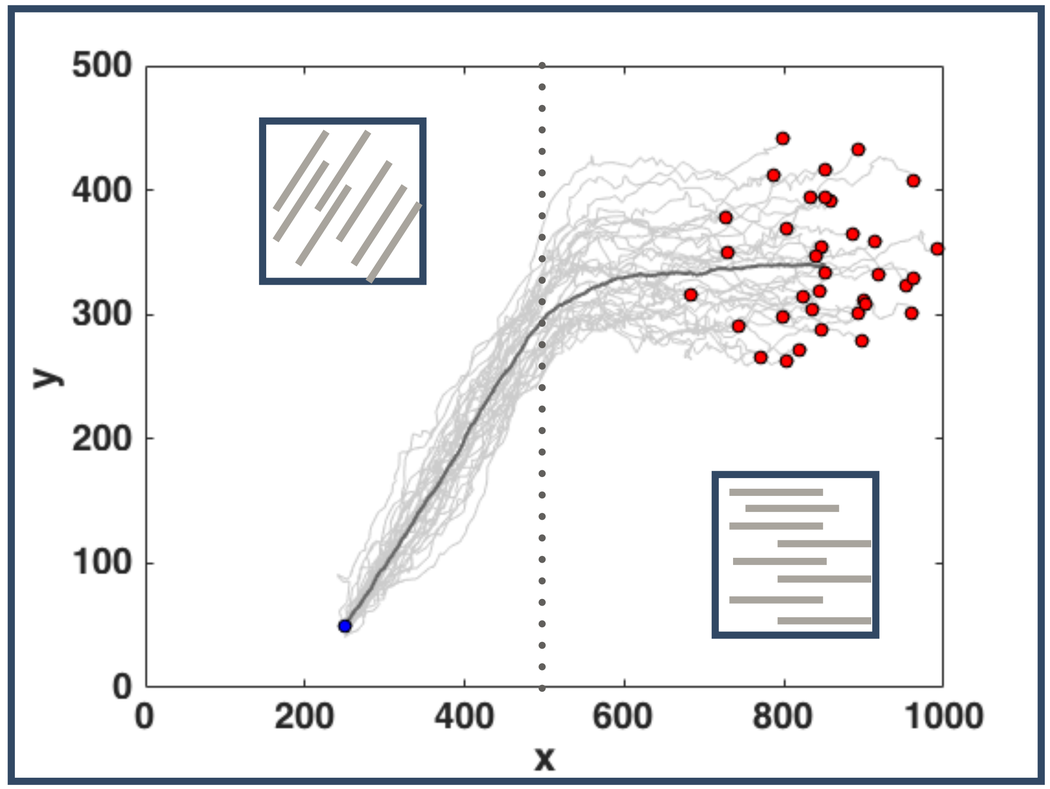
In **PhysiCell_cell.cpp** in function `add_potential`

There will be four different types of interaction:

> ***cell-cell*** as per PhysiCell see (1) Cell-Cell Interactions
> ***cell-fibre*** as per **@cicelykrystyna** code see (2) Cell-Fibre Interactions
> ***fibre-cell*** will be a similar repulsion to cell-cell in which a cell pushes a fibre out of its way see (3) Fibre-Cell Interactions
> ***fibre-fibre*** currently NULL see (4) Fibre-Fibre Interactions



$$\mathbf{F}_{if} = \alpha_{\text{fibre}} \left(1 - \frac{\|\mathbf{v}_i\|}{v_{\max}}\right) \left(\frac{|\mathbf{v}_i \cdot \mathbf{l}_f|}{\|\mathbf{v}_i\|}\right) \mathbf{l}_f - \beta_{\text{fibre}} \left(1 - \frac{|\mathbf{v}_i \cdot \mathbf{l}_f|^2}{\|\mathbf{v}_i\|^2}\right) \mathbf{v}_i$$




```
void Cell::add_potentials(Cell* other_agent) {...}
```

```
// two non-fibre agents e.g. cell-cell interacting - as per PhysiCell
if (this->type_name != "fibre" && (*other_agent).type_name != "fibre") {...}
// cell-type agent interacting with a fibre-type agent
else if (this->type_name != "fibre" && (*other_agent).type_name == "fibre") {...}
// fibre-type agent interacting with a cell-type agent
else if (this->type_name == "fibre" && (*other_agent).type_name != "fibre") {...}
// fibre interacting with a fibre
else if (this->type_name == "fibre" && (*other_agent).type_name == "fibre") {...}
```

```
<vel_adhesion type="double" units="" description="cell velocity parallel to fibre">0.6</vel_adhesion>
<vel_contact type="double" units="" description="cell velocity orthogonal to fibre">0.1</vel_contact>
<cell_velocity_max type="double" units="" description="max cell velocity">1.0</cell_velocity_max>
```

```
double fibre_adhesion = 0;
double fibre_repulsion = 0;
if (distance < max_interactive_distance) {
    double cell_velocity_dot_fibre_direction = 0.;
    for (unsigned int j = 0; j < 3; j++) {
        cell_velocity_dot_fibre_direction += (*other_agent).state.orientation[j] * previous_velocity[j];
    }
    double cell_velocity = 0;
    for (unsigned int j = 0; j < velocity.size(); j++) {
        cell_velocity += previous_velocity[j] * previous_velocity[j];
    }
    cell_velocity = std::max(sqrt(cell_velocity), 1e-8);

    double p_exponent = 1.0;
    double q_exponent = 1.0;
    double xi = fabs(cell_velocity_dot_fibre_direction) / (cell_velocity);
    double xip = pow(xi, p_exponent);
    double xiq = pow((1 - xi * xi), q_exponent);

    fibre_adhesion = (*other_agent).parameters.mVelocityAdhesion * xip *
                     (1 - cell_velocity / this->parameters.mCellVelocityMaximum);

    fibre_repulsion = (*other_agent).parameters.mVelocityContact * xiq;

    axpy(&velocity, &fibre_adhesion, &(*other_agent).state.orientation);
    naxpy(&velocity, &fibre_repulsion, &previous_velocity);
```

PhysiCell_cell.cpp

```
void Cell::update_position( double dt ) {...}
```

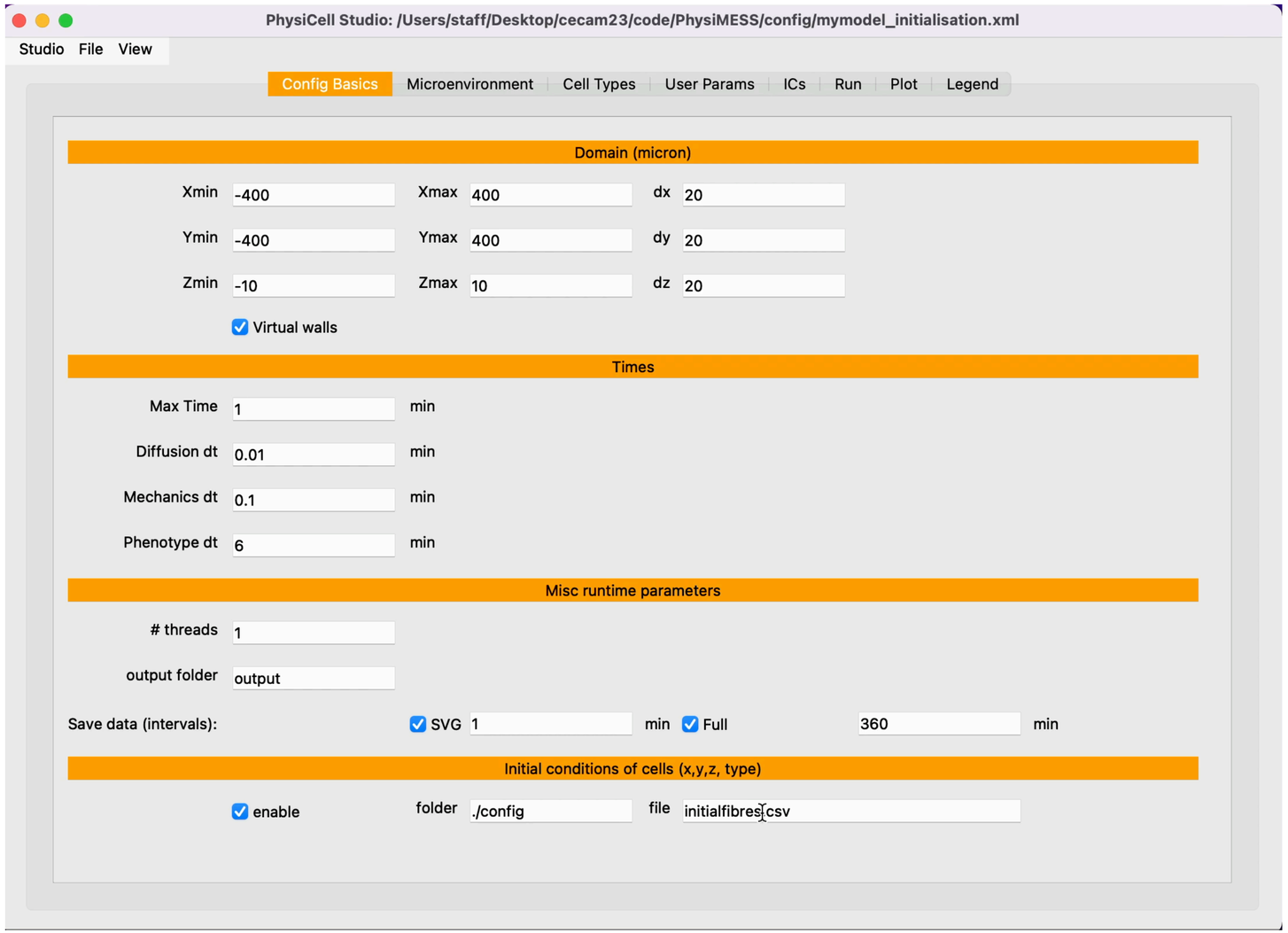
```
double movement_threshold = 0.05;
if (this->type_name != "fibre" && phenotype.motility.is_motile) {
    if (dist( p1: old_position, p2: position) < movement_threshold) {
        this->parameters.stuck_counter++;
    } else {
        this->parameters.stuck_counter = 0;
    }
}
```

PhysiCell_standard_models.cpp

```
void standard_update_cell_velocity(
if (1 <= pCell->parameters.unstuck_counter && pCell->parameters.unstuck_counter < unstuck_threshold+1) {
    pCell->parameters.unstuck_counter++;
    pCell->force_update_motility_vector(dt);
    pCell->velocity += phenotype.motility.motility_vector;
}
else {
    pCell->update_motility_vector(dt);
    pCell->velocity += phenotype.motility.motility_vector;
}
```

Cell-Fibre Mechanics

mymodel_fibremaze.xml
fibre_maze.csv



increase run time

```
<max_time units="min">1000</max_time>
```

enable csv input

```
<cell_positions type="csv" enabled="true">
    <folder>./config</folder>
    <filename>fibre_maze.csv</filename>
</cell_positions>
```

add attractant: secretion rate 10

ECM: Anisotropic 1.57 radians 60 microns

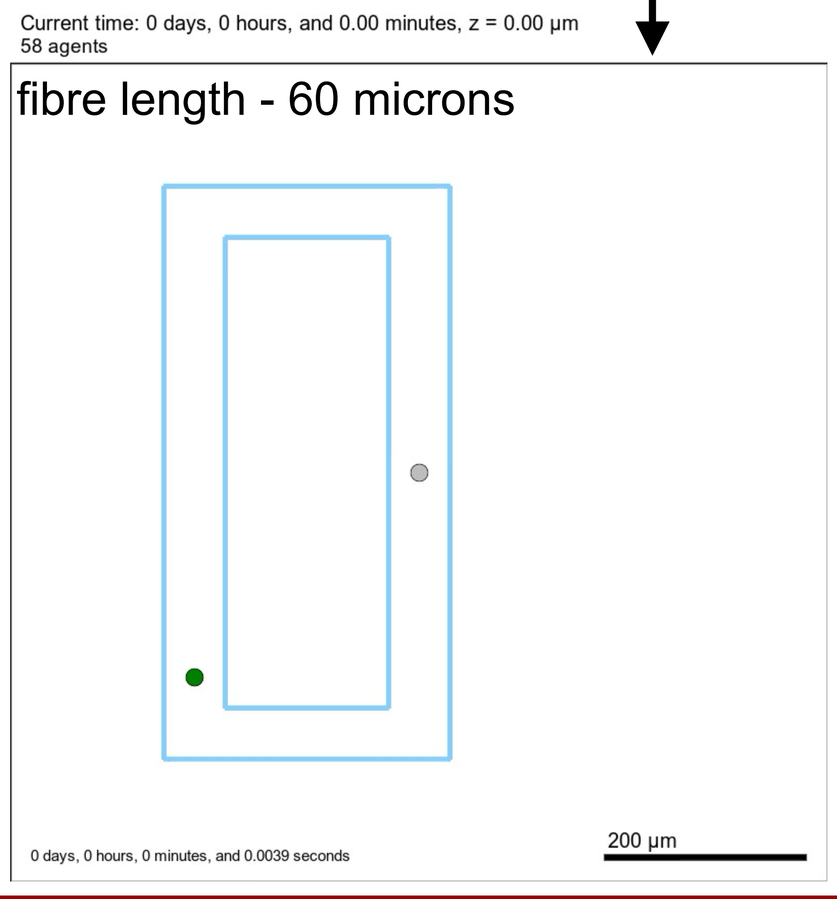
Cell:

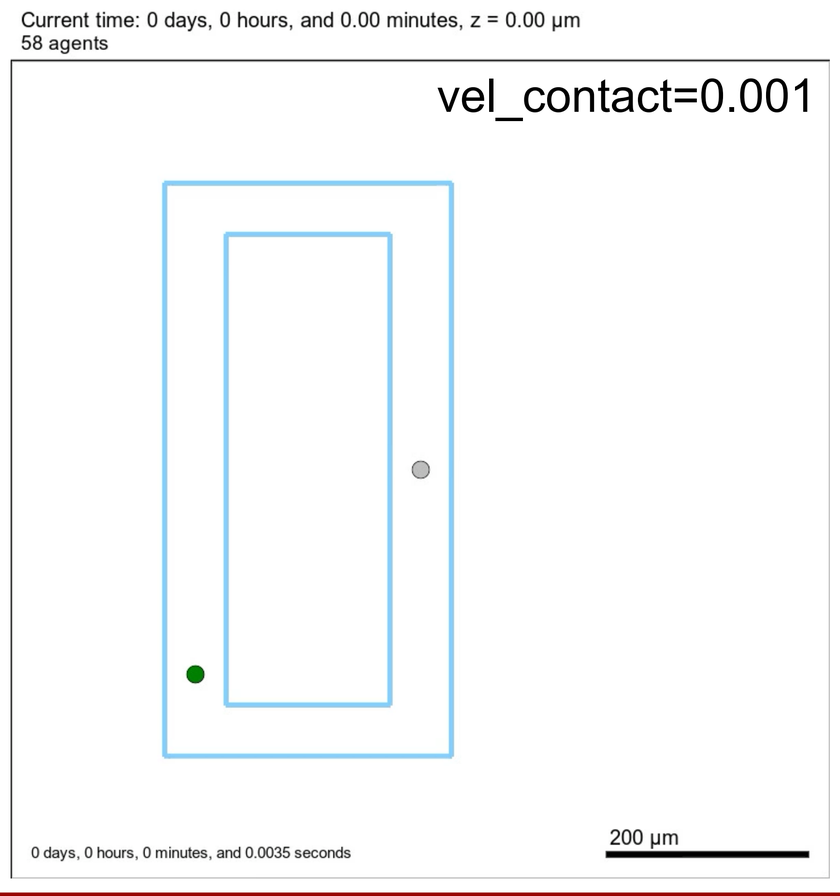
```
<speed>2</speed>
<persistence_time>1</persistence_time>
<migration_bias>.5</migration_bias>
```

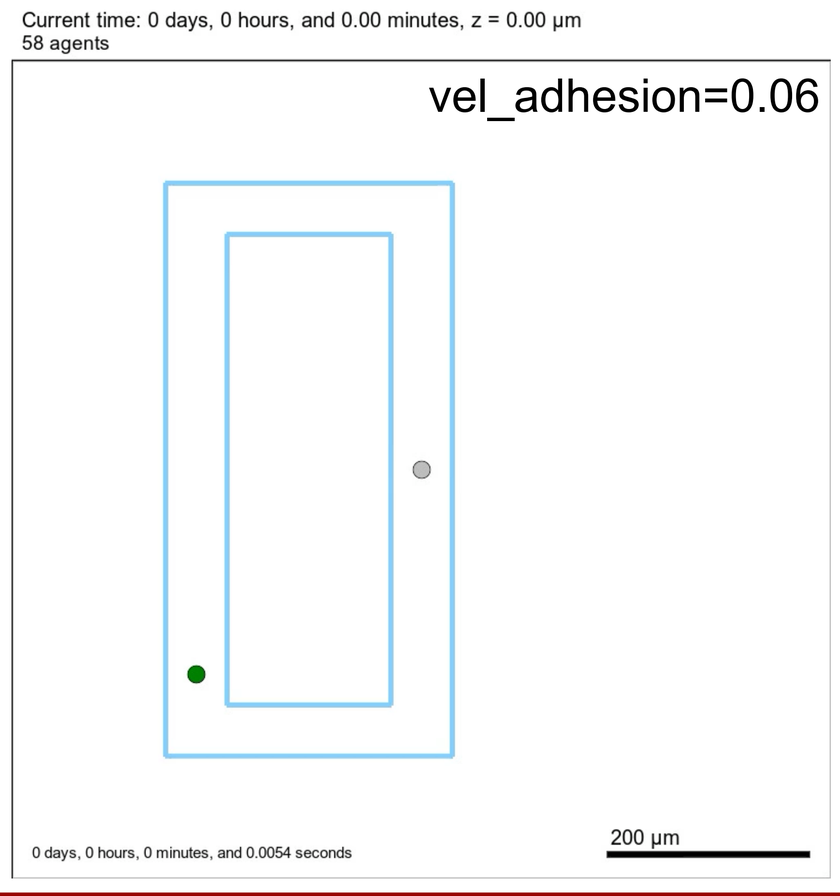
Cell-Fibre Mechanics

mymodel_fibremaze.xml
fibre_maze.csv

```
<vel_adhesion type="double" units="" description="cell velocity parallel to fibre">0.6</vel_adhesion>
<vel_contact type="double" units="" description="cell velocity orthogonal to fibre">0.1</vel_contact>
<cell_velocity_max type="double" units="" description="max cell velocity">1.0</cell_velocity_max>
```







Fibre Degradation

```
void Cell::add_potentials(Cell* other_agent) {...}
```

```
else if (this->type_name != "fibre" && (*other_agent).type_name == "fibre") {...}
```

```
// Fibre degradation by cell - switched on by flag fibre_degradation
int stuck_threshold = this->parameters.stuck_threshold;
if (this->parameters.fibre_degradation && this->parameters.stuck_counter >= stuck_threshold) {
    displacement *= -1.0/distance;
    double dot_product = DotProduct( vector_A: displacement,  vector_B: phenotype.motility.motility_vector);
    if (dot_product >= 0) {
        double rand_degradation = UniformRandom();
        double prob_degradation = this->parameters.fibreDegradationRate;
        if (rand_degradation <= prob_degradation) {
            (*other_agent).parameters.degradation_flag = true;
            this->parameters.stuck_counter = 0;
        }
    }
}
```

PhysiCell_cell.cpp

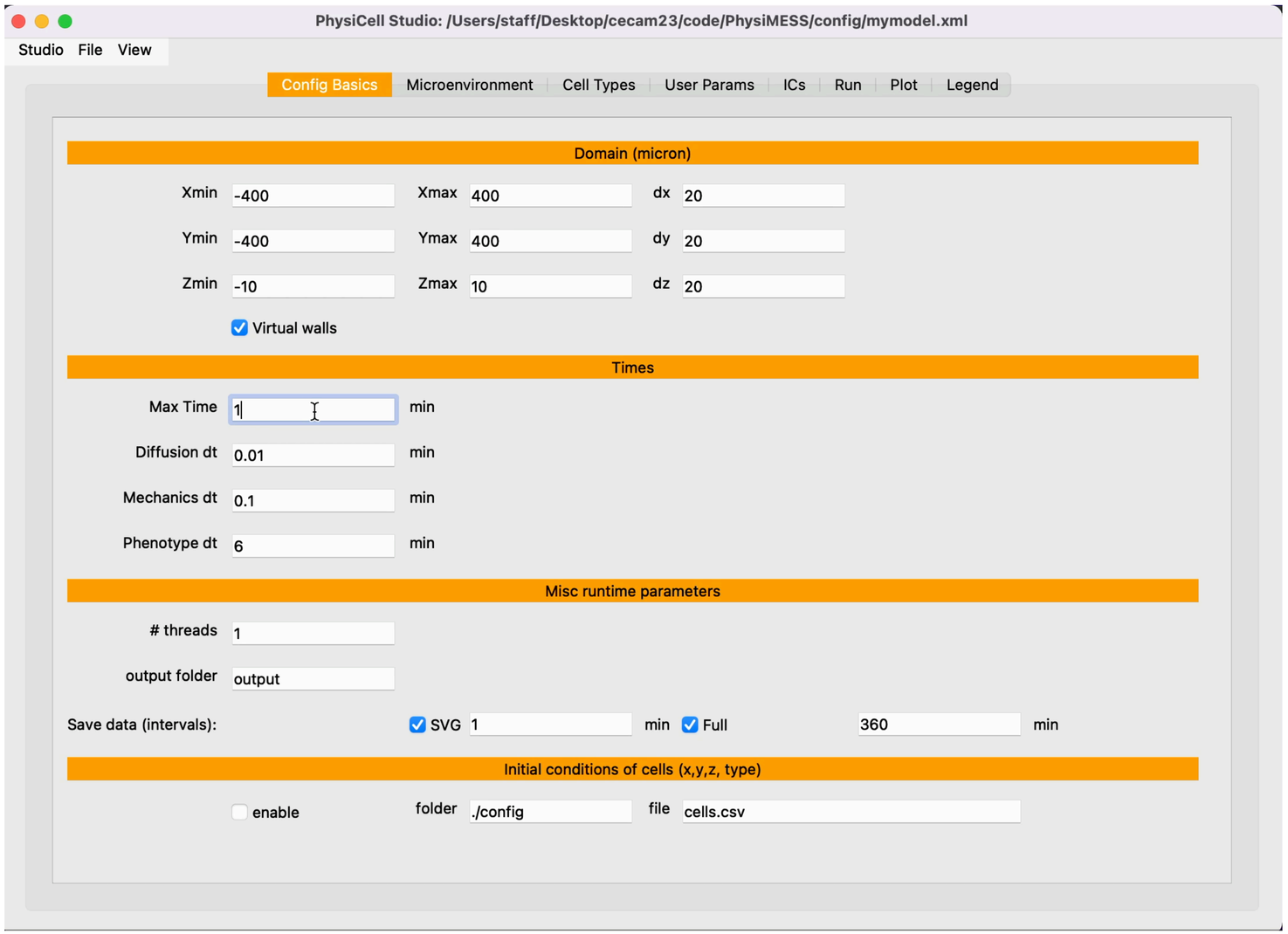
```
// degrade any flagged fibres
#pragma omp parallel for
for( int i=0; i < (*all_cells).size(); i++ )
{
    Cell* pC = (*all_cells)[i];
    if (pC->parameters.degradation_flag){
        pC->degrade_fibre(pC);
    }
}
```

PhysiCell_cell_container.cpp

```
void Cell::degrade_fibre(Cell *fibre_to_degrade) {

    // don't degrade anything that is not a fibre
    if ((*fibre_to_degrade).type_name != "fibre") { return; }
    // de-allocate (delete) the cell;
    fibre_to_degrade->flag_for_removal();
    return;
}
```

Fibre Degradation

mymodel_fibre_degradation.xml
cells_and_fibres_attractant.csv



increase run time

```
<max_time units="min">1000</max_time>
```

enable csv input

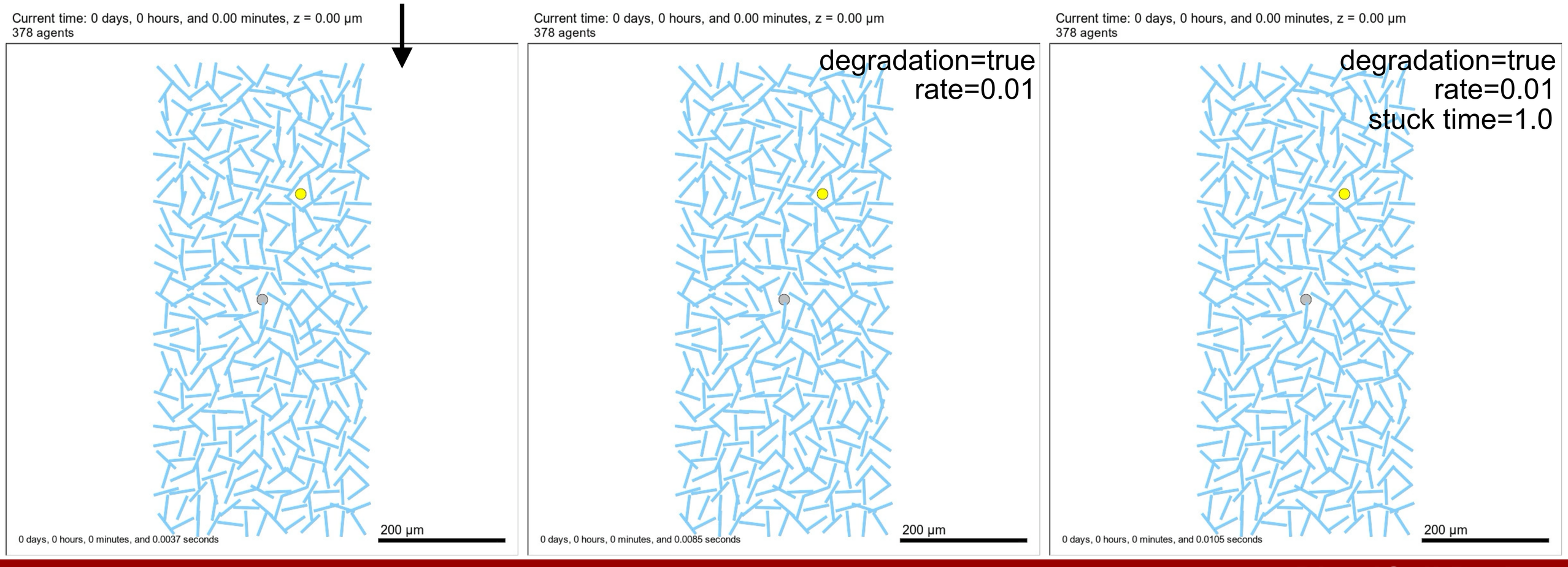
```
<cell_positions type="csv" enabled="true">
      <folder>./config/Fibre_Degradation</folder>
      <filename>cells_and_fibres_attractant.csv</filename>
</cell_positions>
```

ECM: 40 microns

Fibre Degradation

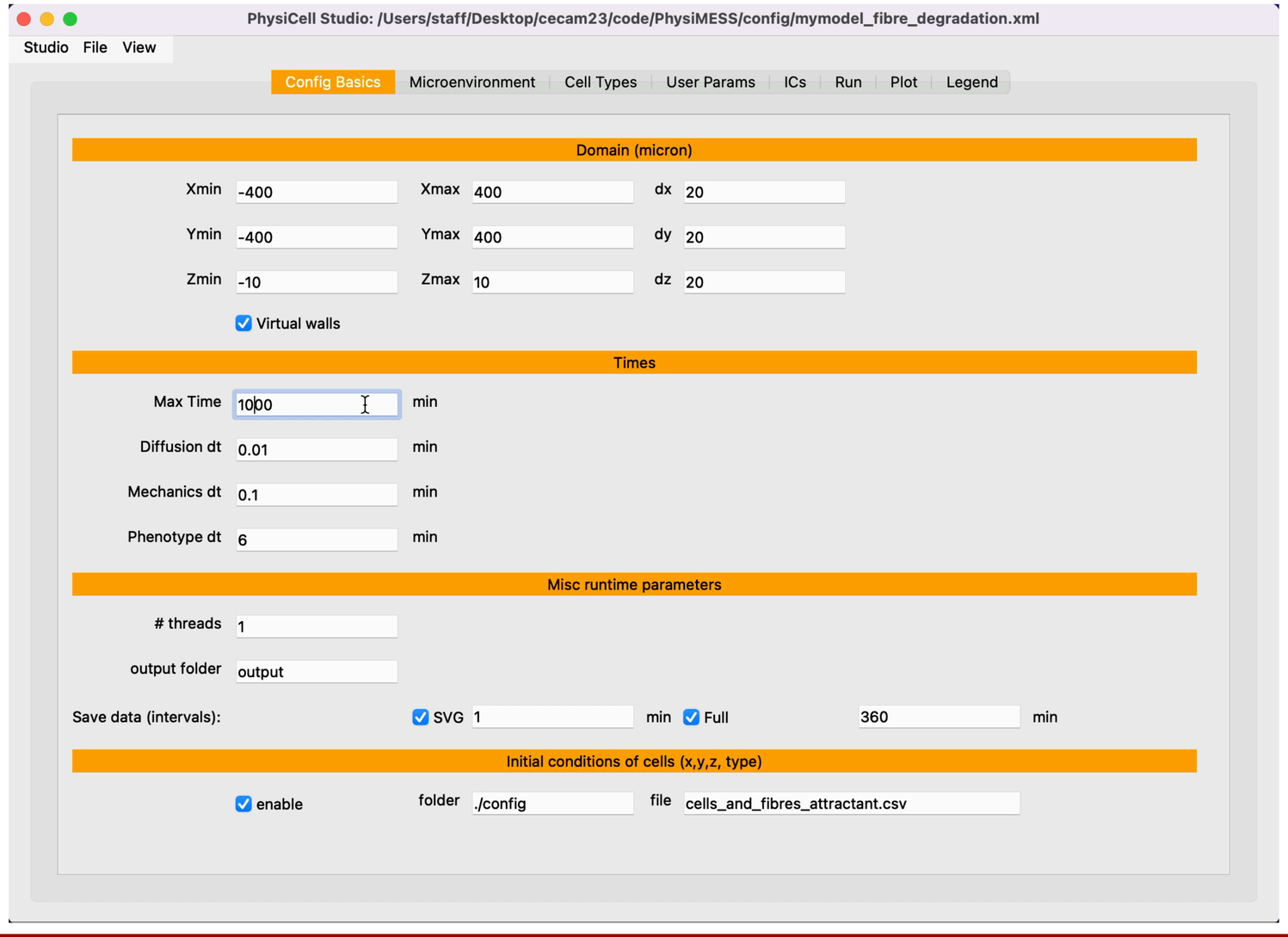
mymodel_fibre_degradation.xml
cells_and_fibres_attractant.csv

```
<fibre_degradation type="bool" units="none" description="flag for fibre degradation">false</fibre_degradation>
<fibre_deg_rate type="double" units="none" description="rate of fibre degradation">0.001</fibre_deg_rate>
<fibre_stuck type="double" units="none" description="how long to wait before considered stuck">10</fibre_stuck>
```



Fibre Degradation

mymodel_matrix_degradation.xml
cells_and_fibres.csv



increase run time

```
<max_time units="min">1500</max_time>
```

save SVG every 5 mins

```
<SVG>
    <interval units="min">5</interval>
    <enable>true</enable>
</SVG>
```

enable csv input

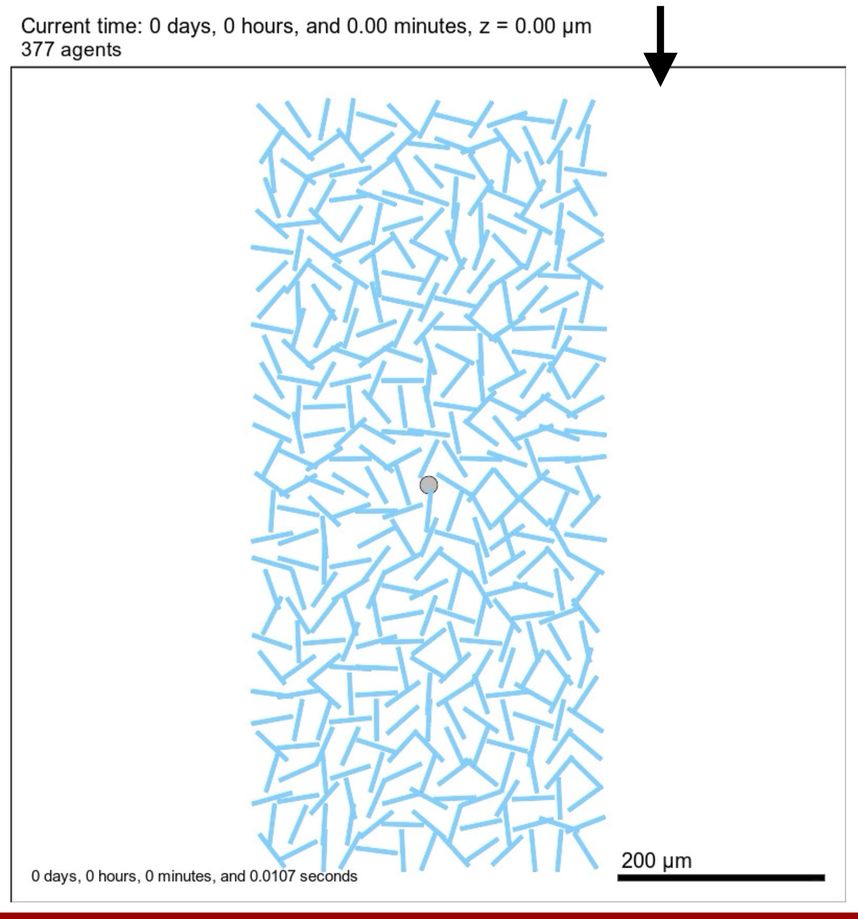
```
<cell_positions type="csv" enabled="true">
    <folder>./config/Fibre_Degradation</folder>
    <filename>cells_and_fibres_attractant.csv</filename>
</cell_positions>
```

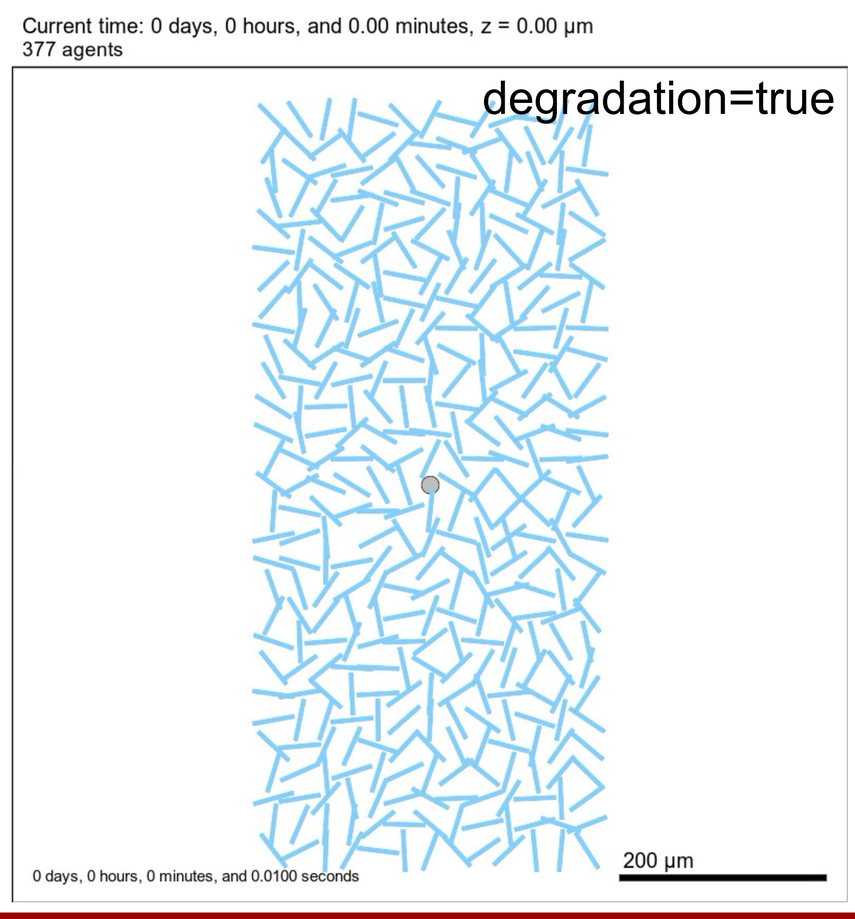
Cell: birth rate 0.0072 speed 0.0

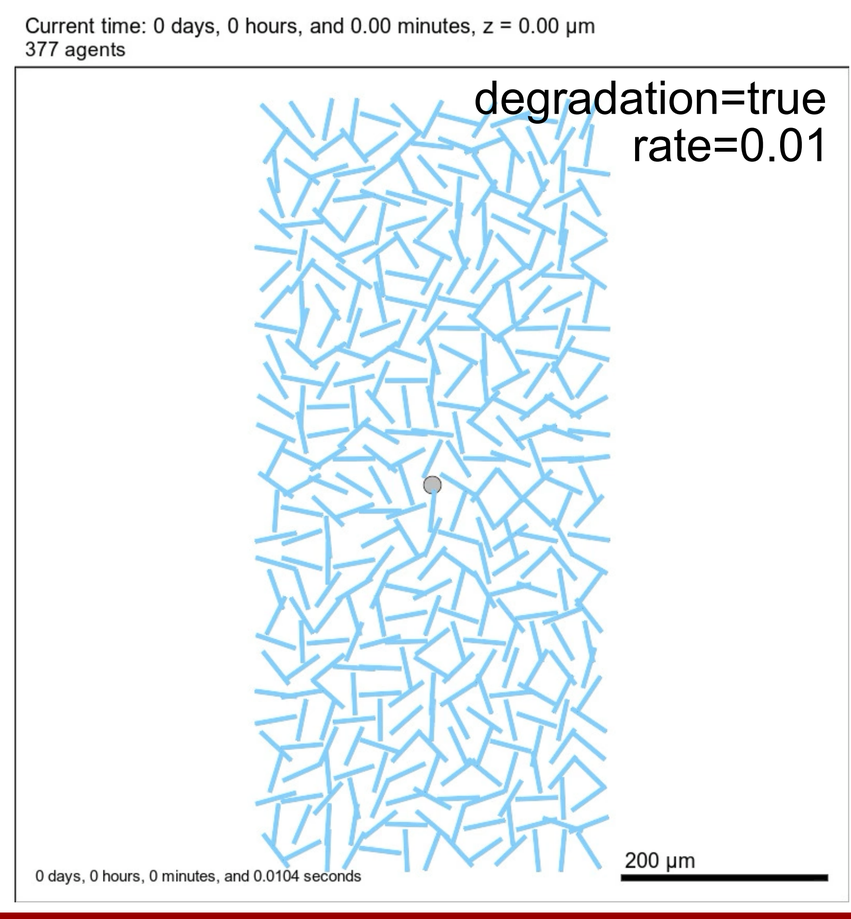
Fibre Degradation

mymodel_matrix_degradation.xml
cells_and_fibres.csv

```
<fibre_degradation type="bool" units="none" description="flag for fibre degradation">false</fibre_degradation>
<fibre_deg_rate type="double" units="none" description="rate of fibre degradation">0.001</fibre_deg_rate>
<fibre_stuck type="double" units="none" description="how long to wait before considered stuck">10</fibre_stuck>
```



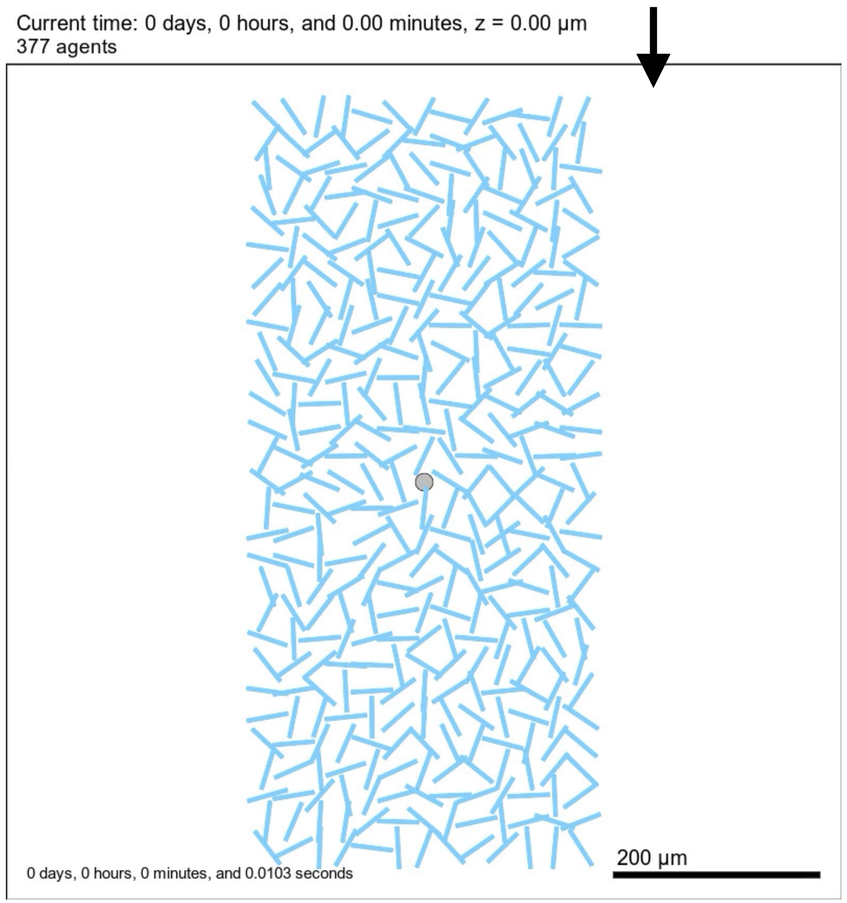


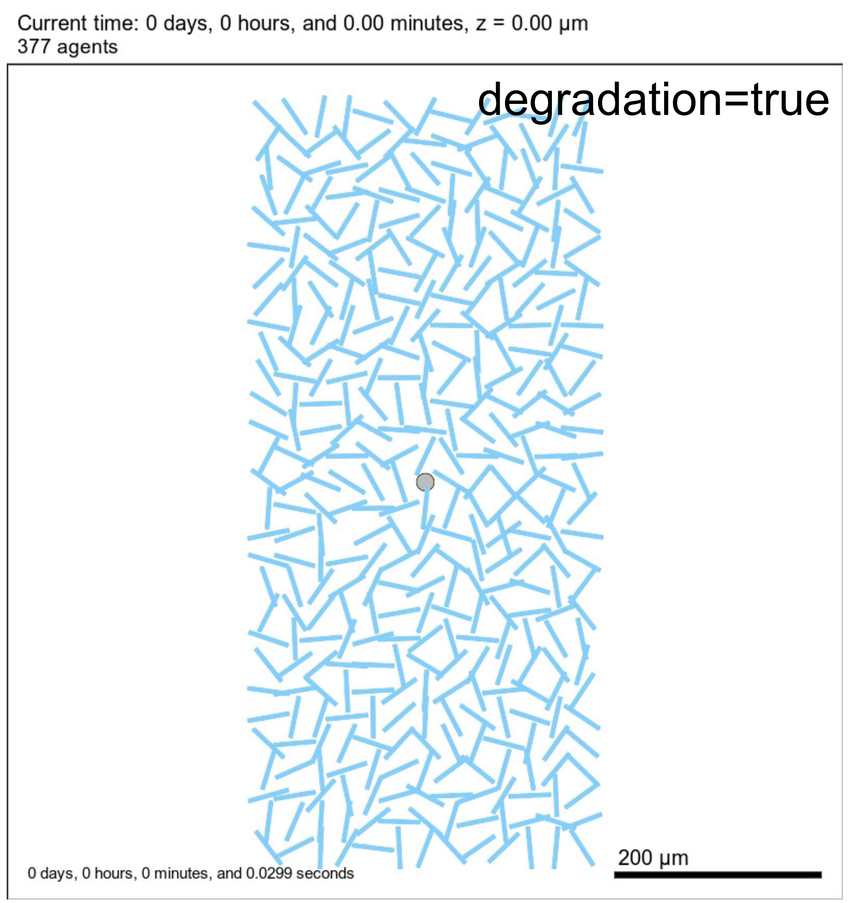


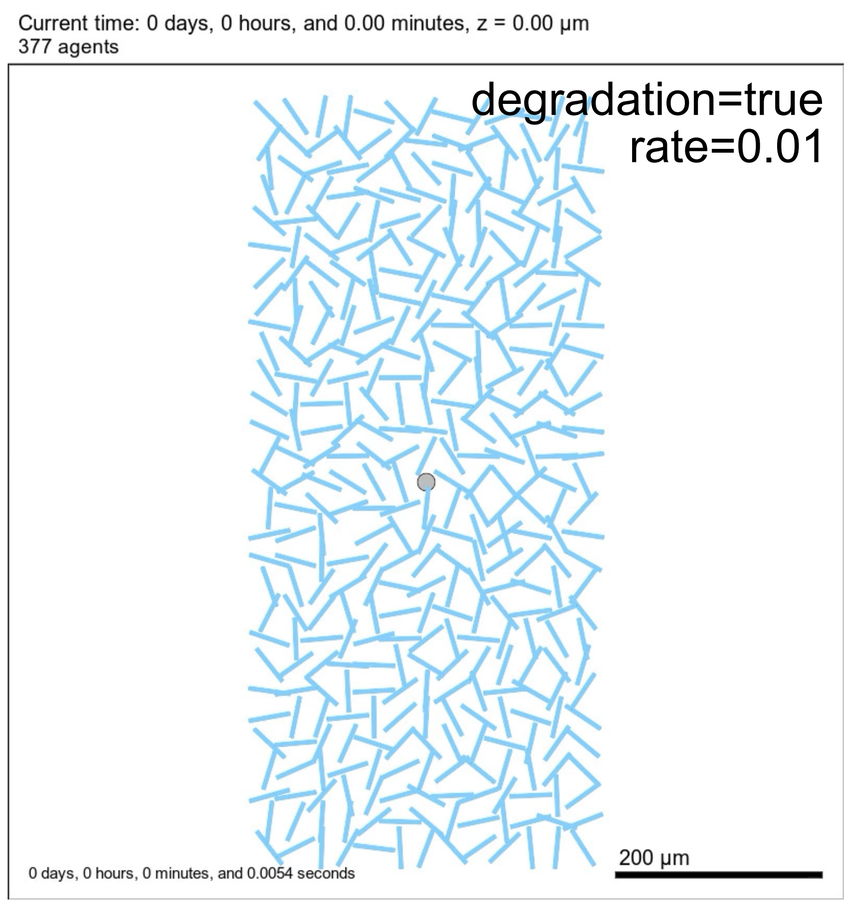
Fibre Degradation

mymodel_matrix_degradation.xml
cells_and_fibres.csv

```
<fibre_degradation type="bool" units="none" description="flag for fibre degradation">false</fibre_degradation>
<fibre_deg_rate type="double" units="none" description="rate of fibre degradation">0.001</fibre_deg_rate>
<fibre_stuck type="double" units="none" description="how long to wait before considered stuck">10</fibre_stuck>
```







Fibre Pushing: free fibres

```
else if (this->type_name == "fibre" && (*other_agent).type_name != "fibre") {...}
```

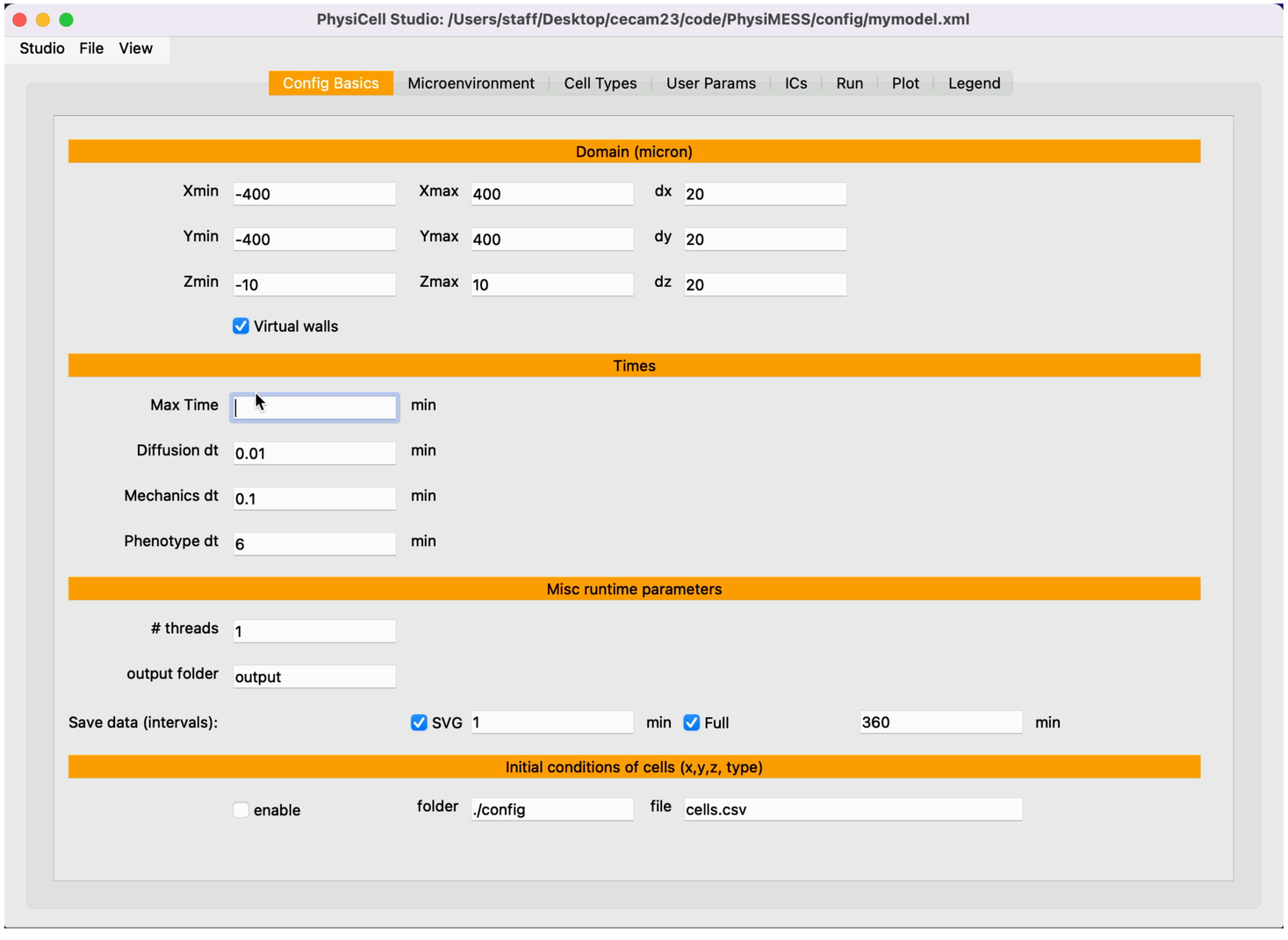
PhysiCell_cell.cpp

```
double R = phenotype.geometry.radius + this->parameters.mRadius;
if (distance <= R) {
    std::vector<double> point_of_impact( n: 3, x: 0.0);
    for (int index = 0; index < 3; index++) {
        point_of_impact[index] = (*other_agent).position[index] - displacement[index];
    }
    // cell-fibre pushing only if fibre no crosslinks
    if (this->parameters.X_crosslink_count == 0) {
        //fibre pushing turned on
        if ((*other_agent).parameters.fibre_pushing) {
            // as per PhysiCell
            static double simple_pressure_scale = 0.027288820670331;
            // temp_r = 1 - distance/R;
            double temp_r = 0;
            temp_r = -distance;
            temp_r /= R;
            temp_r += 1.0;
            temp_r *= temp_r;
            // add the relative pressure contribution NOT SURE IF NEEDED
            state.simple_pressure += (temp_r / simple_pressure_scale);

            double effective_repulsion = sqrt(phenotype.mechanics.cell_cell_repulsion_strength *
                                              (*other_agent).phenotype.mechanics.cell_cell_repulsion_strength);
            temp_r *= effective_repulsion;

            if (fabs(temp_r) < 1e-16) { return; }
            temp_r /= distance;
            naxpy(&velocity, &: temp_r, &: displacement);
        }
    }
```

```
// fibres only get pushed or rotated by motile cells
if (!other_agent->phenotype.motility.is_motile || this->parameters.X_crosslink_count >= 2) {
    return;
}

double distance = 0.0;
nearest_point_on_fibre( point: (*other_agent).position, fibre_agent: this, &: displacement);
for (int index = 0; index < 3; index++) {
    distance += displacement[index] * displacement[index];
}
distance = std::max(sqrt(distance), 0.00001);
```

Fibre Pushing: free fibres

mymodel_pushing.xml
snowplough.csv



increase run time

```
<max_time units="min">500</max_time>
```

enable csv input

```
<cell_positions type="csv" enabled="true">
    <folder>./config</folder>
    <filename>snowplough.csv</filename>
</cell_positions>
```

Cell:

```
<speed>2</speed>
<persistence_time>1</persistence_time>
<migration_bias>.5</migration_bias>
```

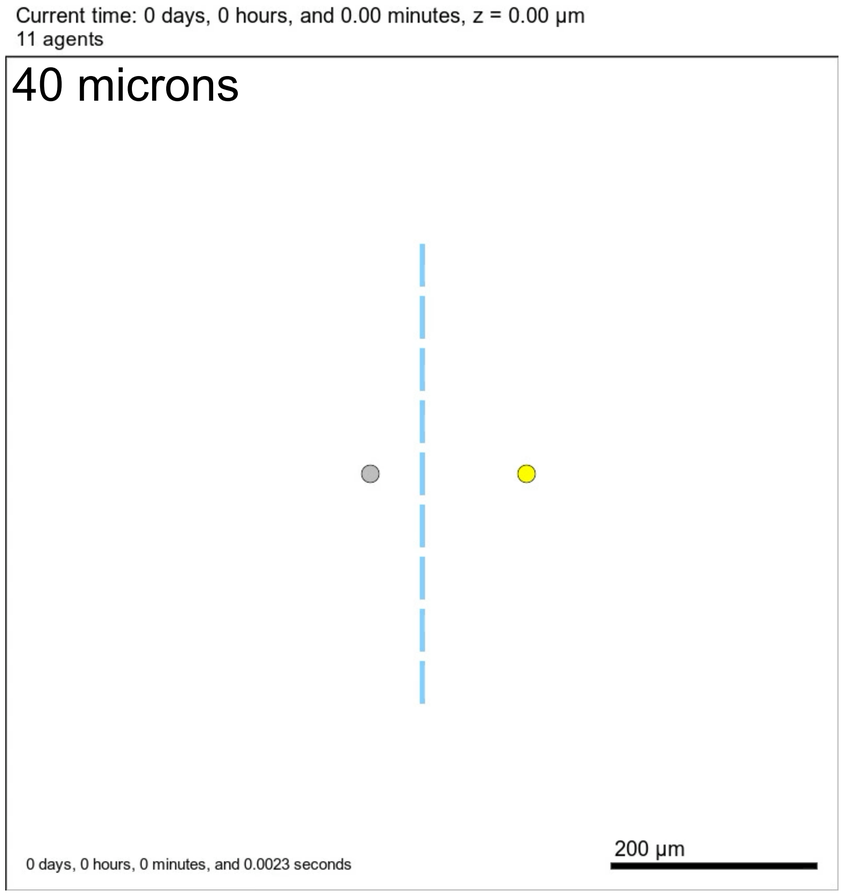
ECM: Anisotropic 1.57 radians 40 microns

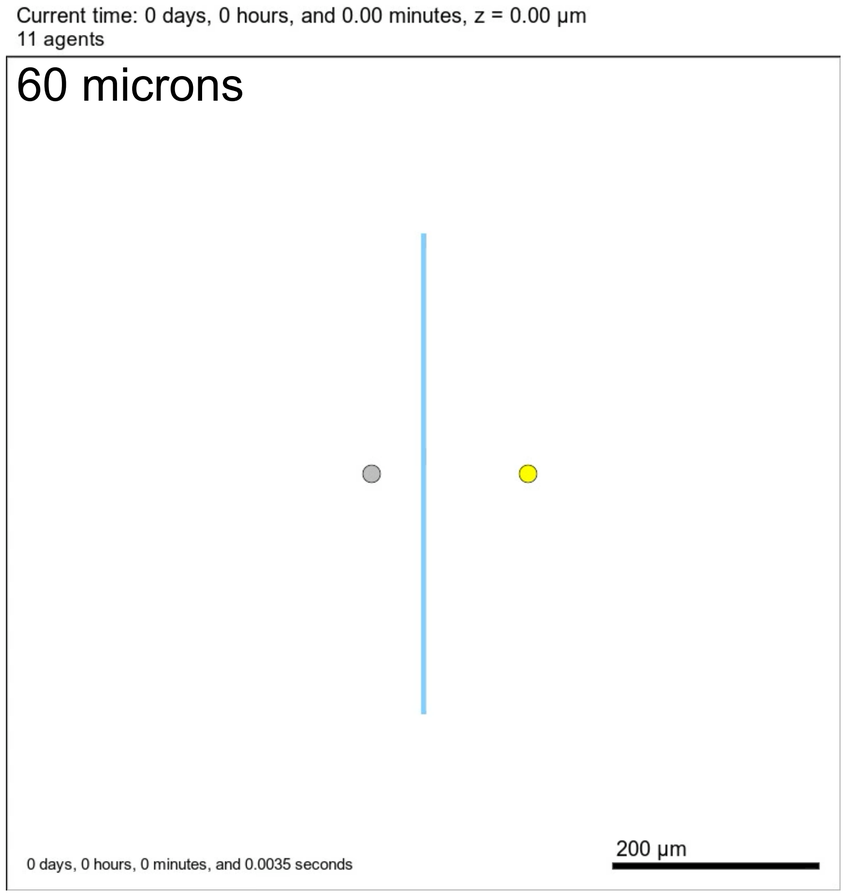
Fibre pushing on

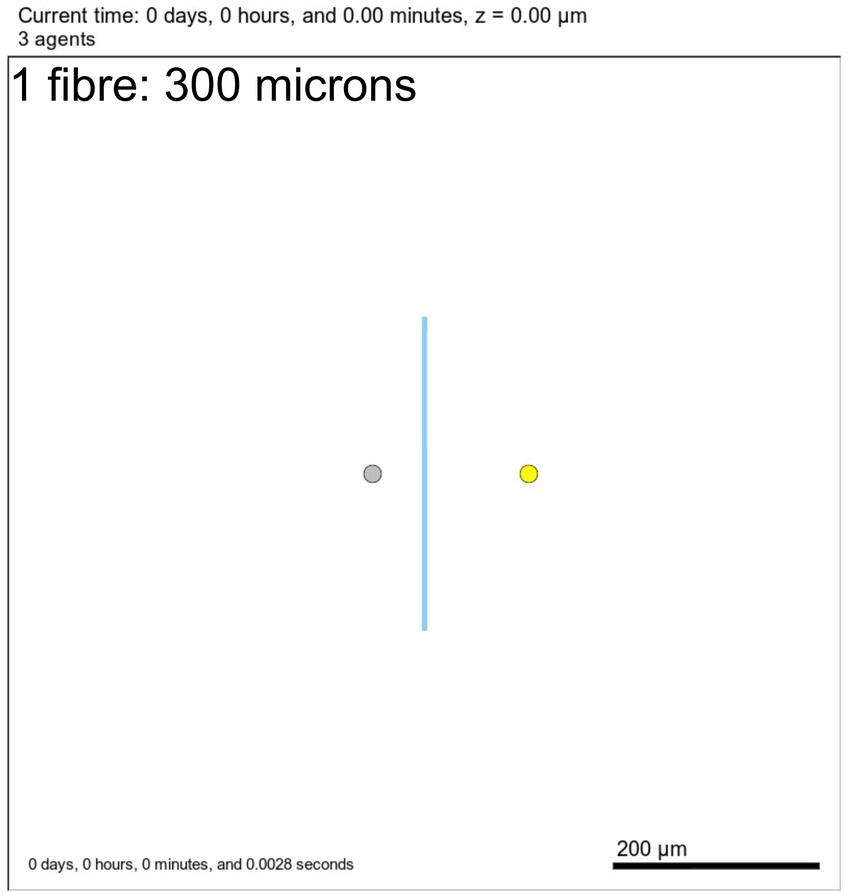
Fibre Pushing: free fibres

mymodel_pushing.xml
snowplough.csv

```
<fibre_pushing type="bool" units="none" description="flag for fibre pushing">true</fibre_pushing>
```







Fibre Rotation: free fibres

```
else if (this->type_name == "fibre" && (*other_agent).type_name != "fibre") {...}
```

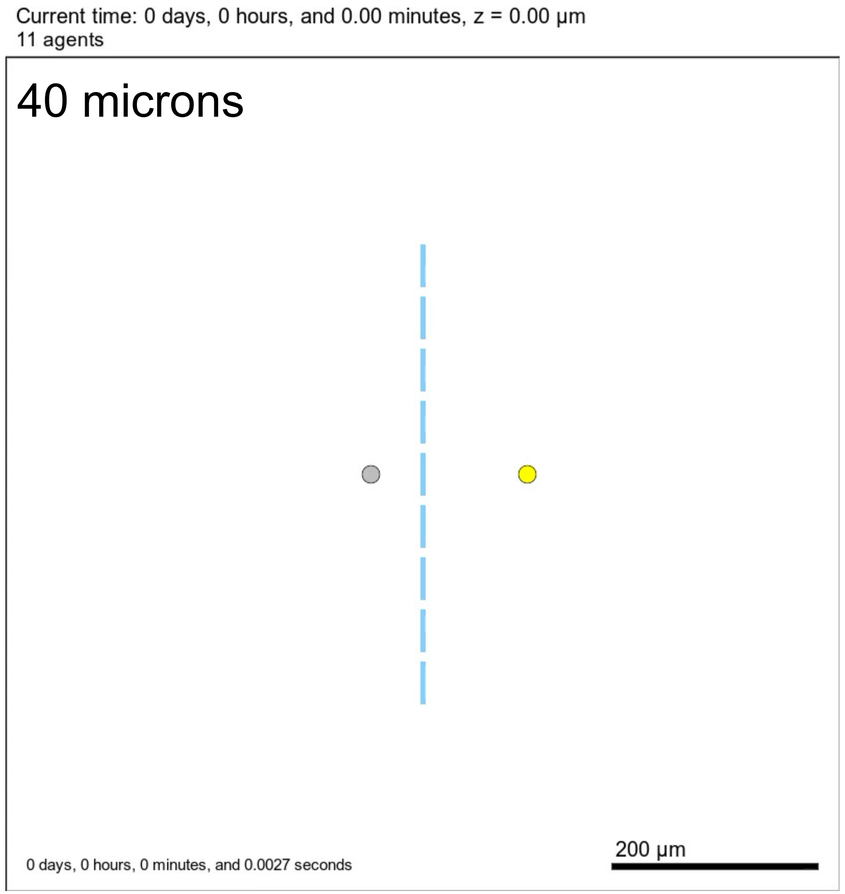
```
// fibre rotation turned on (2D)
if ((*other_agent).parameters.fibre_rotation) {
    std::vector<double> old_orientation( n: 3,  x: 0.0);
    for (int i = 0; i < 2; i++) {
        old_orientation[i] = this->state.orientation[i];
    }

    double moment_arm_magnitude = sqrt(
            point_of_impact[0] * point_of_impact[0] + point_of_impact[1] * point_of_impact[1]);
    double impulse = this->parameters.mFibreStickiness*(*other_agent).phenotype.motility.migration_speed * moment_arm_magnitude;
    double fibre_length = 2 * this->parameters.mLength;
    double angular_velocity = impulse / (0.5 * fibre_length * fibre_length);
    double angle = angular_velocity;
    this->state.orientation[0] = old_orientation[0] * cos(angle) - old_orientation[1] * sin(angle);
    this->state.orientation[1] = old_orientation[0] * sin(angle) + old_orientation[1] * cos(angle);
    normalize(&this->state.orientation);
}
```

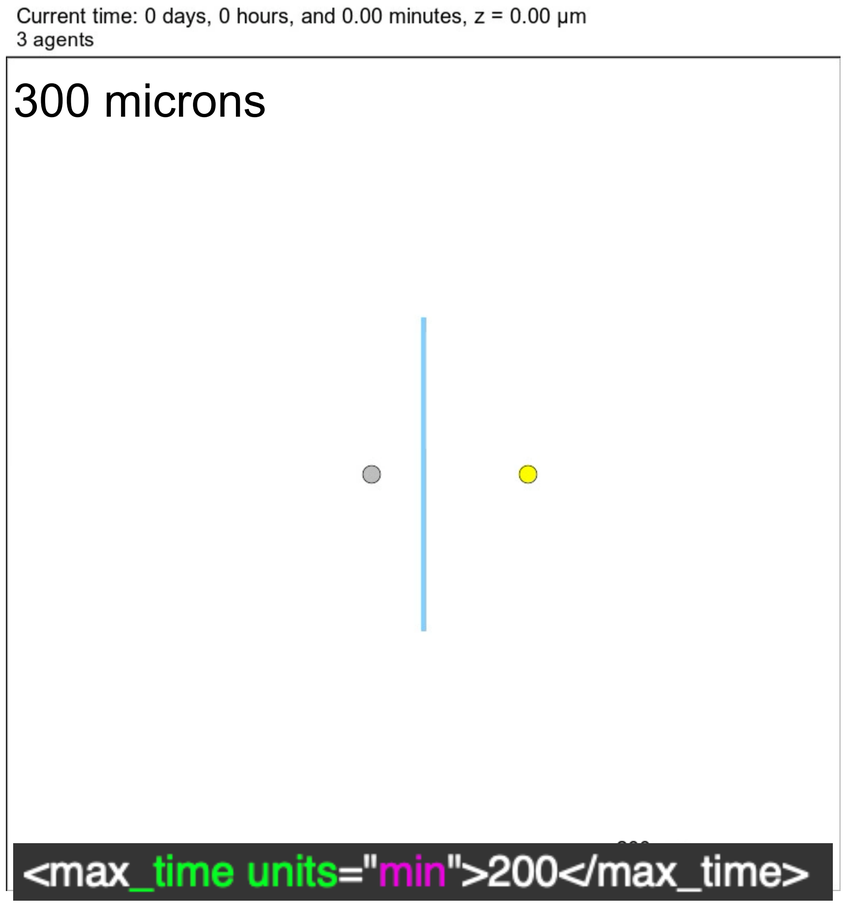
Marco Ruscone
Institut Curie

Fibre Rotation: free fibres

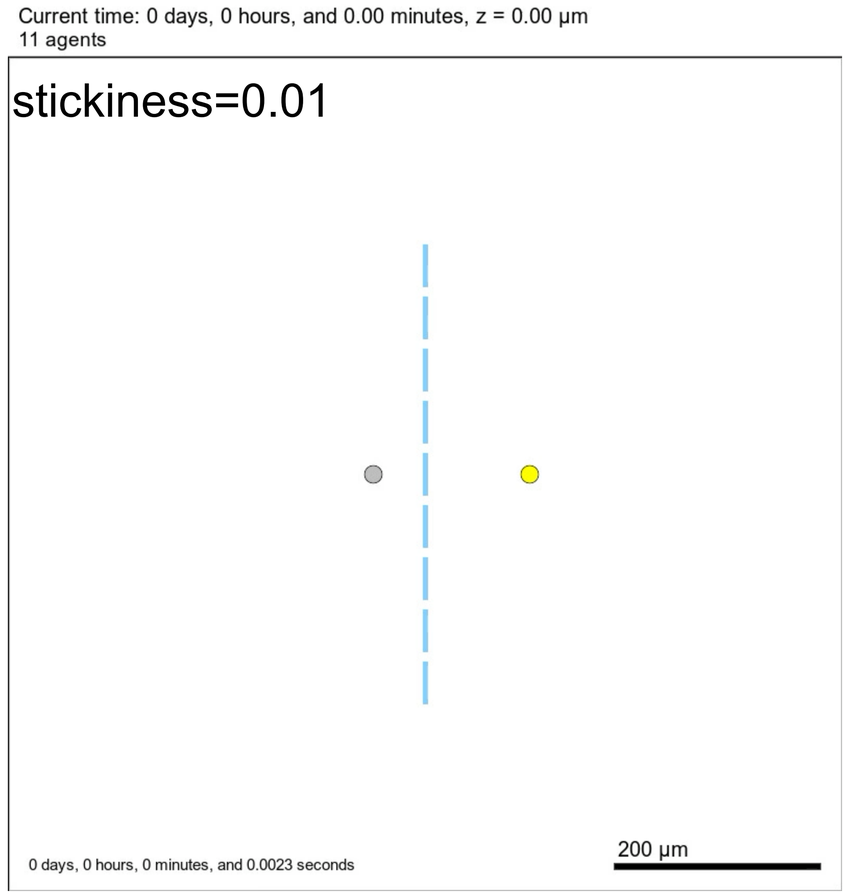
mymodel_rotating.xml
snowplough.csv

```
<fibre_pushing type="bool" units="none" description="flag for fibre pushing">false</fibre_pushing>
<fibre_sticky type="double" units="none" description="measure of how easy it is to move a fibre">1.0</fibre_sticky>
<fibre_rotation type="bool" units="none" description="flag for fibre rotation">true</fibre_rotation>
```






```
<max_time units="min">200</max_time>
```

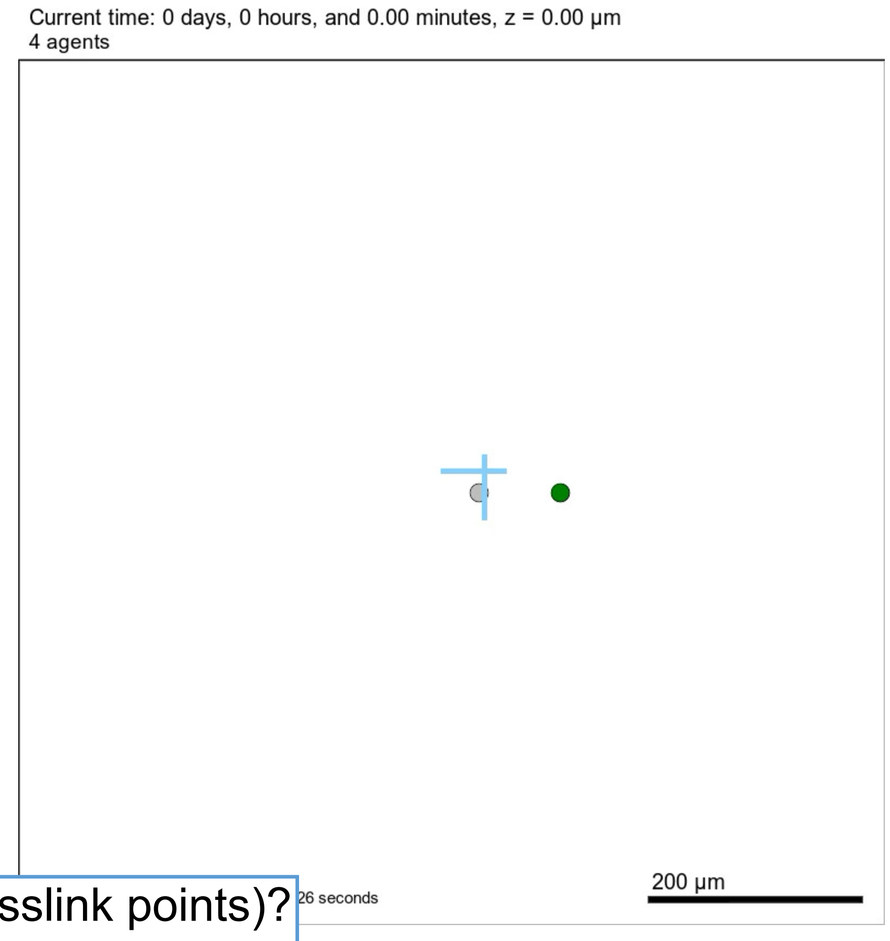


Fibre Rotation: cross linked fibres

mymodel_hinge.xml
hinge.csv

```
else if (this->type_name == "fibre" && (*other_agent).type_name != "fibre") {...}
```

```
// fibre rotation around other fibre (2D only and fibres intersect at a single point)
if ((*other_agent).parameters.fibre_rotation && this->parameters.X_crosslink_count == 1) {
    double distance_fibre_centre_to_crosslink = 0.0;
    std::vector<double> fibre_centre_to_crosslink( n: 3,  x: 0.0);
    for (int i = 0; i < 2; i++) {
        fibre_centre_to_crosslink[i] = this->state.crosslink_point[i]-this->position[i];
        distance_fibre_centre_to_crosslink += fibre_centre_to_crosslink[i]*fibre_centre_to_crosslink[i];
    }
    distance_fibre_centre_to_crosslink = sqrt(distance_fibre_centre_to_crosslink);

    std::vector<double> old_orientation( n: 3,  x: 0.0);
    for (int i = 0; i < 2; i++) {
        old_orientation[i] = this->state.orientation[i];
    }
    double moment_arm_magnitude = sqrt(
            point_of_impact[0] * point_of_impact[0] + point_of_impact[1] * point_of_impact[1]);
    double impulse = this->parameters.mFibreStickiness*(*other_agent).phenotype.motility.migration_speed * moment_arm_magnitude;
    double fibre_length = 2 * this->parameters.mLength;
    double angular_velocity = impulse / (0.5 * fibre_length * fibre_length);
    double angle = angular_velocity;
    this->state.orientation[0] = old_orientation[0] * cos(angle) - old_orientation[1] * sin(angle);
    this->state.orientation[1] = old_orientation[0] * sin(angle) + old_orientation[1] * cos(angle);
    normalize(&this->state.orientation);
    this->position[0] = this->state.crosslink_point[0]-distance_fibre_centre_to_crosslink*state.orientation[0];
    this->position[1] = this->state.crosslink_point[1]-distance_fibre_centre_to_crosslink*state.orientation[1];
}
```



What if two fibres begin to overlap (multiple crosslink points)?
Why does the fibre pair keep rotating?

PhysiMESS Modifications

The following files have been modified for PhysiMESS:

Code read between lines:

```
// !!! PHYSIMESS CODE BLOCK START !!! //
// !!! PHYSIMESS CODE BLOCK END !!! //
```

PhysiMESS/modules/PhysiCell_pathology.cpp

- Code to visualise fibres as thin rectangles rather than spheres added to `SVG_plot`
- Similarly fibres visualised as rectangles in the plot legend via `create_plot_legend`

PhysiMESS/custom-modules/custom.cpp

- Changes made in `set_up_tissue` to allow us to initialise fibres either manually or from a csv file

PhysiMESS/core/PhysiCell_cell.h

```
#include <list> required
```

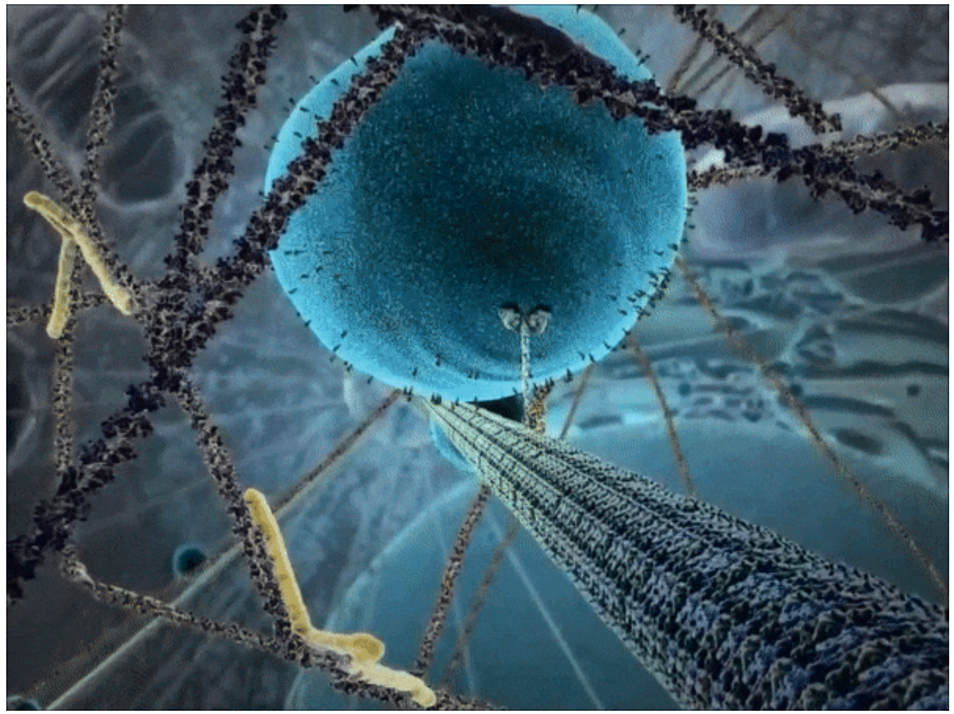
The following additional functions:

- `std::vector<Cell*> crosslinkers;`
- `std::vector<double> crosslink_point;`
- `void force_update_motility_vector(double dt_);`

Next Steps

- Release PhysiMESS
- Incorporate further additional standard/basic physics for cell-ECM interactions
- Keep pushing the capabilities of PhysiMESS/PhysiCell

This is where you come in!!!!



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, 1818187)

ECM Workshop:

Lorentz Center

Training Materials:

- Administrative supplement to NCI U01CA232137 (Year 2)

Other Funding:

- NCI / DOE / Frederick National Lab for Cancer Research (21X126F)
- DOD / Defense Threat Reduction Agency (HDTRA12110015)
- NIH Common Fund (3OT2OD026671-01S4)