**Figure Instructions**

**Note**: If you do not want to repeat the simulations themselves, but instead only want to look at some example output, you can download our data from here:

<https://drive.google.com/a/kathryn.at/file/d/0B-CJMxfuuz7wai1EOTduNjdoWW8/view?usp=sharing>

The contents of the Data folder should be placed in the exampleOutput directory of your ElegansGermline repository. It will then be possible to view the data in Paraview, or plot from it using the scripts in RScripts.

Included in our data download is:

BaselineRuns – 30 repeats up to 100hph with a Baseline parameter set

NoStretching\* - Runs used to produce Fig. 5, without proximal arm stretching

FastCycle\* - For Fig 6: 8 hour cell cycle, no contact inhibition, various death

rates

SlowCycle\* - For Fig 6: 24 hour cell cycle, no contact inhibition, various

death rates

ForVideo – A dataset with more frequent snapshots, used to produce

SIVideo1

ForTracking – Run used to produce Fig. 7. Contains tracking data for all cells,

which was disabled during the other simulations to save time

**Figure 1**

Produced in Inkscape, no simulation.

Note that the adult germ line micrograph is a composite of two images, because the whole organ does not fit in view at this magnification.

**Figure 2**

Produced in Inkscape, no simulation.

**Figure 3**

To reproduce this figure you need a simulation run with baseline parameters. Either use one of the BaselineRuns provided in our data, or, from the main Chaste directory type:

*scons co=1 b=GccOpt ./projects/ElegansGermline/test/TestElegansGermline.hpp*

Then from the directory Chaste/projects/ElegansGermline/build/optimised type:

*export CHASTE\_TEST\_OUTPUT = \*Path To Your ElegansGermline*

*/exampleOutput Directory\**

*./TestElegansGermlineRunner “Baseline.txt” “MyBaselineRun” 35 82*

This runs a baseline parameters simulation, putting the output in

Chaste/projects/ElegansGermline/exampleOutput/MyBaselineRun. The simulation end time [35] is adjusted from 20 hours to 82. The simulation is very time consuming and will take multiple hours to complete.

Once the simulation finishes, you should have a results\_from\_time\_0 directory under MyBaselineRun. Using Paraview (<http://www.paraview.org/>) open the results\_..vtu files inside. Select the following Paraview options:

* Representation = 3D Glyphs
* Add a Glyph
* In the properties menu, select Scalars = Radius, Glyph Type = Sphere, Radius = 1, Scale Mode = scalar, Scale Factor = 1.0.
* Select Glyph colour to be Solid Colour yellow.
* Apply a Threshold filter . Select Scalars = CellCyclePhase, Minimum = -1.1, Maximum = 0. This selects all cells in Meiosis, which can then be Solid Coloured green.
* Using the Pipeline Browser to select groups of cells, and to keep all groups visible, apply these additional Threshold filters…

On the yellow Glyphs: threshold on isDTC between 0.1 and 1.1 to select the DTC and colour it cyan.

On the green cells: threshold on SpermFated between 0.1 and 1.1 to select spermatogenic cells and colour them pale blue.

On the green cells: threshold on OocyteFated between 0.1 and 1.1 to select oogenic cells and colour them pink.

On the spermatogenic cells: threshold on Differentiation\_Sperm between 0.1 and 1.1 to select mature sperm and colour them dark blue.

If any of these changes does not appear, click Apply in the Properties menu to refresh the screen. The germ line should now be coloured as in our manuscript. You may need to adjust the zoom . The images in Figure 3 are screenshots taken from this Paraview setup at particular time points.

**Figure 4**

This figure is also produced using a Baseline parameter simulation. However, you may wish to do multiple repeats, in order to plot the standard deviation of each property. The procedure for compiling the code is identical to Figure 3, but you will need to run:

*./TestElegansGermlineRunner “Baseline.txt” “MyBaselineRun\_N” 35 82*

for N between 0 and 29, say. Alternatively use the 30 BaselineRuns provided in our data.

Once the simulations have finished, open up the script *plotGonadData.R* from the RScripts directory in a text editor. Make any necessary changes to the user input section at the top of the file:

* Set the flag “showMicronZoneLengths” to FALSE.
* Ensure that resultsDirectory points to the directory containing all of your *MyBaselineRun* folders.
* If you ran multiple simulations, set “plotMultipleFiles” to TRUE and set “fileNumbers” the sequence of numbers identifying the runs, e.g. seq(0,29). Otherwise set “plotMultipleFiles” to FALSE.
* Set baseFileName to the name shared by all of your output directories  
  e.g. *MyBaselineRun*.
* Unless you are providing manual counts of cell rows or total number of proliferative cells, set the flags useManuallyCountedCellRows and useManuallyCountedProlifCells to FALSE.

Finally, run the script in R: source(“plotGonadData.R”)

**Note** that two of the measurements we plot here involve counts of cell rows. Counting rows is not an activity that the computer performs very successfully; the numbers given by our simulation are approximate and tend to be a systematic underestimate. For this reason, our published figure is based on manual counts of cell rows carried out in Paraview. Manual count data can be provided by creating a file called “ZoneLengths.txt” under each output directory, filling it in with counts, then plotting with useManuallyCountedCellRows set to TRUE. See our BaselineRuns data for an example.

**Figure 5**

The images in Figure 5C simply come from a Baseline parameters run; they can be reproduced by following the instructions in Figure 3 for visualisation in Paraview.

The remaining images in Figures 5A and B come from simulations with proximal arm stretching disabled. 5B includes DTC halting, while 5A does not. These properties of a simulation can be configured using command line arguments.

So, Figure 5A corresponds to our dataset NoStretchingNoHalting, which can be reproduced by running:

*./TestElegansGermlineRunner "Baseline.txt" “NoStretchNoHalt” 1 0 5 46 37 0*

This sets the proximal arm stretching rate [1] to 0, the DTC migration rate in late L4 [5] to 46, and the DTC halting flag [37] to 0.

Meanwhile Figure 5B corresponds to our dataset NoStretchingButHalting, which can be reproduced by running:

*./TestElegansGermlineRunner "Baseline.txt" “NoStretchButHalt” 1 0 5 46*

i.e. the same simulation, but with the DTC halting flag left at its default value of 1.0.

Finally, the results\_..vtu output from both simulations can be visualized in Paraview, by following the same instructions as for Figure 3.

**Figure 6**

This figure is based on 6 sets of runs, all of them carried out without contact inhibition. Switching contact inhibition off requires recompilation. Go into the statechart model file src/statechart/**FateUncoupledFromCycle.cpp**, and set the flags contactInhibitionInG2 and contactInhibitionInG1 to false.  
  
From the main Chaste directory, recompile:

*scons co=1 b=GccOpt ./projects/ElegansGermline/test/TestElegansGermline.hpp*

Then from the build/optimised directory carry out the following runs (ideally with repeats)

For part (A):

1) *./TestElegansGermlineRunner "Baseline.txt" “FastCycleD0.025” 35 50*

No CI, end time = 50.

2) *./TestElegansGermlineRunner "Baseline.txt" “FastCycleD0.2” 35 50 21 0.2*

No CI, end time = 50, death rate = 0.2.

3) *./TestElegansGermlineRunner "Baseline.txt" “FastCycleD0.5” 35 50 21 0.5*

No CI, end time = 50, death rate = 0.5.

For part (B):

1) *./TestElegansGermlineRunner "Baseline.txt" “SlowCycleD0.025” 35 90 15*

*24*

No CI, end time = 90, 24 hour cell cycle

2) *./TestElegansGermlineRunner "Baseline.txt" “SlowCycleD0.1” 35 90 15 24*

*21 0.1*

No CI, end time = 90, death rate = 0.1, 24 hour cell cycle

3) *./TestElegansGermlineRunner "Baseline.txt" “FastCycleD0.2” 35 90 15 24*

*21 0.2*

No CI, end time = 90, death rate = 0.2, 24 hour cell cycle.

The results of these 6 sets of simulations can be plotted using the R Script: source(“plotDeathRateVariation.R”).

**Figure 7**

This figure also uses a Baseline parameters run, but with some additional data output. Cell positions must be tracked, and division events must also be recorded (see our example dataset ForTracking).   
  
Cell tracking is enabled by default. Check that the lines:  
  
*MAKE\_PTR\_ARGS(CellTrackingOutput<3>, positionRecording, (parameters  
->GetParameter(36), 1));*

*simulator.AddSimulationModifier(positionRecording);*

are present somewhere in **TestElegansGermline.hpp** (around line 276).   
  
Division event recording is *not* enabled by default, as the output would be annoying in many circumstances. To enable division event recording, go to the file **Cell.cpp** in Chaste/cell\_based/src/population/cell. Scroll down to the method *Divide*.

This method seems to be the only place in the Chaste code where we can access both the cell ID of the mother and the cell ID of the daughter during a division. After the block:

*// Create daughter cell with modified cell property collection*

*CellPtr p\_new\_cell(new Cell(GetMutationState(), mpCellCycleModel  
->CreateCellCycleModel(), false, daughter\_property\_collection));*

at lines 444-445, add in the following line of code:

*std::cout << SimulationTime::Instance()->GetTime() << "\t" << this->GetCellId() << "\t" << p\_new\_cell->GetCellId() << "\t" << std::endl;*

Now, whenever a division occurs during a simulation, the time, mother ID and daughter ID will be output to command line. To produce the figure we’re aiming for, we need to save this output to a file called DivisionData.txt. We recommend doing this by redirecting all the simulation’s command line output to a file, then deleting irrelevant lines at the end of the run. After recompiling from the main Chaste directory:  
  
 *scons co=1 b=GccOpt ./projects/ElegansGermline/test/TestElegansGermline.hpp*

Go to Chaste/projects/ElegansGermline/build/optimised and run:  
  
*./TestElegansGermlineRunner “Baseline.txt”* “MyTrackingRun” 35 82 | DivisionData.txt

Place DivisionData.txt in the same location as the other output text files. Check that TrackingData.txt was produced. Lastly, open up DivisionData.txt and delete any lines not related to cell divisions.   
  
Now we’re ready to set the user input paths in, and then run, the R script *getTreePaths.R*: *source(“GetTreePaths.R”)*

This script defines a function *getTreeOfPaths* that can read our data files and generate a branching series of cell tracks for visualisation in Paraview. Once the script has run, type in R:  
  
*getTreeOfPaths(1, 0, 82)*

*getTreeOfPaths(8, 0, 82)*

*getTreeOfPaths(16, 0, 82)*to generate path data for cells 1, 8 and 16 along with their descendants. The resulting files are called “branchingPathsDataTime0to82Cell<X>.vtu”.

Visualising the cell tracking data in Paraview is a little time consuming, and we only give a summary here…  
First, load up the results\_..vtu files as normal, and follow the Figure 3 instructions to represent each cell as a spherical glyph. Colour the glyphs grey and reduce their Opacity in the Properties menu. Using threshold filters, select the cells with ancestor labels 1, 8 and 16 and colour them red, green and blue.  
Next, open your path files in Paraview, using a LegacyVTK reader. Apply a Tube filter, and colour the resulting tree by the Time property. To clip the cell paths to a certain time range, edit the colour scale of the tube filter, making it transparent outside the range of interest. To add arrows, add a 3D Arrow glyph to each point in the path, and direct it according to the vector property directions.

**Figure 8**

This figure also uses a single Baseline parameter run (although only up to 20 hours, so you can dispense with the ‘35 82’ command line arguments and run a shorter simulation if you wish). Other than that, follow exactly the instructions for Figure 3, up to the point where the results are open in Paraview and you are choosing a colour for the 3D glyphs. At that point, rather than selecting Solid Colour yellow, colour by Ancestors instead. Edit the colour scale, selecting a spectrum of colours and rescaling it to the range of the data (0-16). You can now take screenshots of the pattern of ancestor labels at each time point.

**SIFigure1**

Run the script from the RScripts directory:

*source(“plotGrowthRates.R”)*

The mean and standard deviation measurements from Table 2 are coded into this script.

**SIFigure2**

Follow the instructions for Figure 4. Once your simulations are complete, in the RScripts directory, edit the script *PlotGonadData.R*. Make all the same changes to the user input section at the top of the file as for Figure 4, EXCEPT:

* Change the flag “showMicronZoneLengths” to TRUE, rather than FALSE.

Run the script using source(“PlotGonadData.R”)

**SIFigure3**

Produced in Inkscape, no simulation.

**SIFigure4**

Produced in Inkscape, no simulation.

To reproduce **SIVideo1**, select one of the baseline runs used to make Figure 3 OR the ForVideo run from our data. Look in the results\_from\_time\_0 directory, and open the results\_...vtu collection of files in Paraview. Display and colour code the cells as described in the Figure 3 instructions. Go to File, and select SaveAnimation. From there, either save the frames as PNG images and make them into a movie using ffmpeg, or save a .avi movie directly.