

Supplementary Information for:

Computational model of wound healing: EGF secreted by fibroblasts promotes delayed re-epithelialization of epithelial keratinocytes

Vivi Andasari, Dongyuan Lü, Maciej Swat, ShiLiang Feng,
Fabian Spill, Li Chen, Xiangdong Luo, Muhammad Zaman, Mian Long

This supplementary information presents the XML (or CC3DML) and Python listings of the simulation implementation in CompuCell3D from the model in the main text.

Listing 1. CC3DML configuration for co-culture simulations

```
<CompuCell3D>

<Potts>
  <Dimensions x="450" y="200" z="1"/>
  <Steps>4401</Steps>
  <Temperature>80</Temperature>
  <NeighborOrder>2</NeighborOrder>
</Potts>

<Plugin Name="CellType">
  <CellType TypeId="0" TypeName="Medium"/>
  <CellType TypeId="1" TypeName="Keratinocyte"/>
  <CellType TypeId="2" TypeName="Fibroblast"/>
  <CellType Freeze="" TypeId="3" TypeName="Wall"/>
</Plugin>

<Plugin Name="Volume"/>

<Plugin Name="Surface"/>

<Plugin Name="CenterOfMass"/>

<Plugin Name="Contact">
  <Energy Type1="Medium" Type2="Wall">10.0</Energy>
  <Energy Type1="Keratinocyte" Type2="Wall">250.0</Energy>
  <Energy Type1="Fibroblast" Type2="Wall">250.0</Energy>
  <Energy Type1="Wall" Type2="Wall">10.0</Energy>
  <NeighborOrder>3</NeighborOrder>
</Plugin>
```

```

<Plugin Name="AdhesionFlex">
  <AdhesionMolecule Molecule="ECad"/>
  <AdhesionMolecule Molecule="Int"/>
  <AdhesionMolecule Molecule="Coll"/>

  <AdhesionMoleculeDensity CellType="Keratinocyte" Molecule="ECad" Density="4.0"/>
  <AdhesionMoleculeDensity CellType="Keratinocyte" Molecule="Int" Density="4.0"/>
  <AdhesionMoleculeDensity CellType="Fibroblast" Molecule="ECad" Density="4.0"/>
  <AdhesionMoleculeDensity CellType="Fibroblast" Molecule="Int" Density="4.0"/>
  <AdhesionMoleculeDensity CellType="Medium" Molecule="Coll" Density="15.0"/>

  <BindingFormula Name="Binary">
    <Formula> min(Molecule1,Molecule2)</Formula>
    <Variables>
      <AdhesionInteractionMatrix>
        <BindingParameter Molecule1="ECad" Molecule2="ECad">
          -13.0</BindingParameter>
        <BindingParameter Molecule1="Int" Molecule2="Coll">-1.5</BindingParameter>
      </AdhesionInteractionMatrix>
    </Variables>
  </BindingFormula>

  <NeighborOrder>3</NeighborOrder>
</Plugin>

<Steppable Type="DiffusionSolverFE">
  <DiffusionField Name="EGF">
    <DiffusionData>
      <FieldName>EGF</FieldName>
      <GlobalDiffusionConstant>0.0</GlobalDiffusionConstant>
      <GlobalDecayConstant>2.4e-3</GlobalDecayConstant>
    </DiffusionData>
    <SecretionData>
      <Secretion Type="Fibroblast">0.0</Secretion>
    </SecretionData>
  </DiffusionField>
</Steppable>

<Steppable Type="UniformInitializer">
  <Region>
    <BoxMin x="110" y="10" z="0"/>
    <BoxMax x="190" y="190" z="1"/>
    <Gap>0</Gap>
    <Width>3</Width>
    <Types>Keratinocyte</Types>
  </Region>
  <Region>
    <BoxMin x="325" y="10" z="0"/>
    <BoxMax x="385" y="190" z="1"/>
    <Gap>0</Gap>
    <Width>3</Width>
    <Types>Fibroblast</Types>
  </Region>
</Steppable>

</CompuCell3D>

```

Listing 2. Python Steppables for co-culture simulations

```
from PySteppables import *
import CompuCell
import sys
from random import uniform
from math import *

class CocultureStretchSteppable(SteppableBasePy):

    def __init__(self, _simulator, _frequency=1):
        SteppableBasePy.__init__(self, _simulator, _frequency)
        self.adhesionFlexPlugin=CompuCell.getAdhesionFlexPlugin()

    def start(self):

        # Live plot for cell position
        self.pw1 = self.addNewPlotWindow(\
            _title = 'Location',\
            _xAxisTitle = 'MCS', _yAxisTitle = 'Cell position', \
            _xScaleType = 'linear', _yScaleType = 'linear' \
        )

        self.pw1.addPlot('Cell_id_811', _style='Dots', _color='red', _size=2)
        self.pw1.addPlot('Cell_id_810', _style='Steps', _color='blue', _size=1)

        # Live plot for integrin expression
        self.pw2 = self.addNewPlotWindow(\
            _title = 'Integrin',\
            _xAxisTitle = 'MCS', _yAxisTitle = 'Integrin density', \
            _xScaleType = 'linear', _yScaleType = 'linear' \
        )

        self.pw2.addPlot('Cell_id_810', _style='Dots', _color='green', _size=1)

        # Live plot for EGF concentration
        self.pw3 = self.addNewPlotWindow(\
            _title = 'EGF',\
            _xAxisTitle = 'MCS', _yAxisTitle = 'EGF concentration', \
            _xScaleType = 'linear', _yScaleType = 'linear' \
        )

        self.pw3.addPlot('Cell_id_810', _style='Dots', _color='black', _size=1)

        # Build a wall of frozen cells on the boundaries
        self.buildWall(self.WALL)

        # Cells' properties
        for cell in self.cellList:
            cell.targetVolume = 20
            cell.lambdaVolume = 25
            cell.targetSurface = 20
            cell.lambdaSurface = 15
            if cell.type == self.FIBROBLAST:
                cell.fluctAmpl = 7
```

```

def step(self, mcs):

    field = self.getConcentrationField('EGF')

    # Loop over all cell types
    for cell in self.cellList:
        conc = field[int(cell.xCOM), int(cell.yCOM), 0]
        mint = self.adhesionFlexPlugin.getAdhesionMoleculeDensity(cell, "Int")

        # Change fibroblast's membrane fluctuation to 7
        if cell.type == self.FIBROBLAST:
            cell.fluctAmpl = 7.0

    if mcs >= 200:
        if cell.type == self.KERATINOCYTE:

            alpha = 0.15
            nu = 0.01
            K1 = 1.0
            phi = 1.0
            n = 1.0

            # Implement Equation (5)
            mint = (phi*mint + alpha*(conc**n))/(K1 + nu*(conc**n))
            self.adhesionFlexPlugin.setAdhesionMoleculeDensity(cell, \
                "Int", mint)

    # Track the properties of a keratinocyte moving toward fibroblasts
    if cell.id == 810:
        print "Integrin level cellID 810 (right) : ", mint
        if cell.volume > 0:
            currentCellPosition = cell.xCM/float(cell.volume)
            print "Current cellID 810 (right) position = ", \
                currentCellPosition

            # Write cell position output
            self.pW1.addDataPoint("Cell_id_810", mcs, currentCellPosition)
            fileName810pos = "CellID810Position_"+str(mcs)+".txt"
            self.pW1.savePlotAsData(fileName810pos)

            # Write integrin expression output
            self.pW2.addDataPoint("Cell_id_810", mcs, mint)
            fileName810int = "CellID810Integrin_"+str(mcs)+".txt"
            self.pW2.savePlotAsData(fileName810int)

            # Write EGF concentration output
            self.pW3.addDataPoint("Cell_id_810", mcs, conc)
            fileName810egf = "CellID810EGF_"+str(mcs)+".txt"
            self.pW3.savePlotAsData(fileName810egf)

    # Track a keratinocyte moving to the left (away from fibroblasts)
    if cell.id == 811:
        print "Integrin level cellID 811 (left) : ", mint
        if cell.volume > 0:
            currentCellPosition = cell.xCM/float(cell.volume)
            print "Current cellID 811 (left) position = ", \
                currentCellPosition
            self.pW1.addDataPoint("Cell_id_811", mcs, currentCellPosition)

```

```

def finish(self):
    # Finish Function gets called after the last MCS
    pass

# Steppable to implement EGF diffusion and secretion after 200 MCS

from XMLUtils import dictionaryToMapStrStr as d2mss
from XMLUtils import CC3DXMLListPy

class DiffusionSolverSteering(SteppablePy):

    def __init__(self, _simulator, _frequency=1):
        SteppablePy.__init__(self, _frequency)
        self.simulator=_simulator

    def step(self, mcs):

        if mcs>=200:
            fiexDiffXMLData=self.simulator.getCC3DModuleData("Steppable",\
                "DiffusionSolverFE")

            if fiexDiffXMLData:
                diffusionFieldsElementVec=CC3DXMLListPy(\
                    fiexDiffXMLData.getElements("DiffusionField"))

                for diffusionFieldElement in diffusionFieldsElementVec:
                    if diffusionFieldElement.getFirstElement("DiffusionData"\
                        ).getFirstElement("FieldName").getText()=="EGF":
                        diffConstElement=diffusionFieldElement.getFirstElement(\
                            "DiffusionData").getFirstElement("GlobalDiffusionConstant")
                        secretionElement=diffusionFieldElement.getFirstElement(\
                            "SecretionData").getFirstElement("Secretion",\
                                d2mss({"Type":"Fibroblast"}))

                        # Convert string value of <DiffusionConstant> element to
                        # float
                        diffConst=float(diffConstElement.getText())
                        secretionConst=float(secretionElement.getText())
                        print "Previous diffusion constant: ", diffConst
                        print "Previous secretion constant: ", secretionConst

                        # Set diffusion & secretion constants
                        diffConst = 0.25
                        secretionConst = 0.2

                        # Update the value of the <DiffusionConstant> element -
                        # convert float to string
                        diffConstElement.updateElementValue(str(diffConst))
                        secretionElement.updateElementValue(str(secretionConst))
                        print "Current diffusion constant : ", diffConst
                        print "Current secretion constant : ", secretionConst

                self.simulator.updateCC3DModule(fiexDiffXMLData)

```

Listing 3. Python main file for co-culture simulations

```
import sys
from os import environ
from os import getcwd
import string

sys.path.append(environ[ "PYTHON_MODULE_PATH" ])

import CompuCellSetup

sim,simthread = CompuCellSetup.getCoreSimulationObjects()

CompuCellSetup.initializeSimulationObjects(sim,simthread)

steppableRegistry=CompuCellSetup.getSteppableRegistry()

#Register steppables
from CocultureStretchSteppables import CocultureStretchSteppable
cocultureStretchSteppable = CocultureStretchSteppable(sim,_frequency=1)
steppableRegistry.registerSteppable(cocultureStretchSteppable)

from CocultureStretchSteppables import DiffusionSolverSteering
instanceOfDiffusionSolverSteering = DiffusionSolverSteering(sim,_frequency=200)
steppableRegistry.registerSteppable(instanceOfDiffusionSolverSteering)

CompuCellSetup.mainLoop(sim,simthread,steppableRegistry)
```