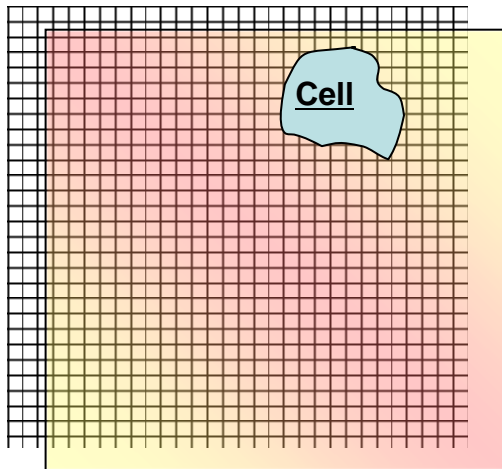


Building CompuCell3D
Simulations
Step-by-Step Tutorial
Maciej Swat

Building Your First CompuCell3D Simulation

All simulation parameters are controlled by the config file. The config file allows you to only add those features needed for your current simulation, enabling better use of system resources.

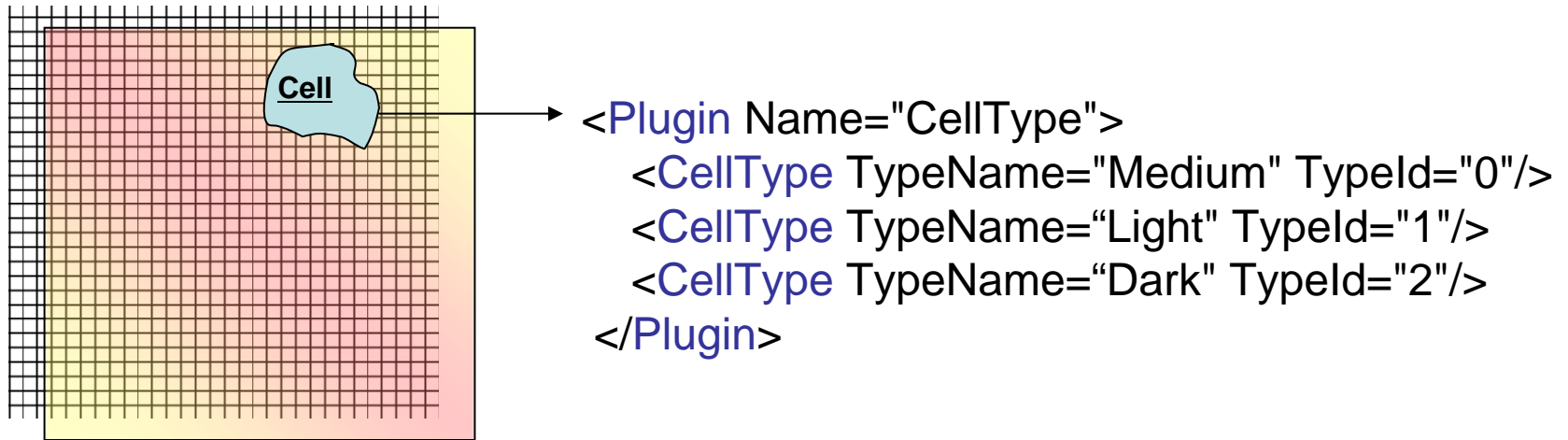
Define Lattice and Simulation Parameters



```
< CompuCell3D>  
<Potts>  
  <Dimensions x="100" y="100" z="1"/>  
  <Steps>10</Steps>  
  <Temperature>2</Temperature>  
  <Flip2DimRatio>1</Flip2DimRatio>  
</Potts>  
...  
</CompuCell3D>
```

Define Cell Types Used in the Simulation

Each CompuCell3D xml file must list all cell types that will be used in the simulation



Notice that Medium is listed with `TypeId = 0`. This is both convention and a **REQUIREMENT** in CompuCell3D. Reassigning Medium to a different `TypeId` may give undefined results. This limitation will be fixed in one of the next CompuCell3D releases

Define Energy Terms of the Hamiltonian and Their Parameters



Cell

Volume

volume
volumeEnergy(cell)

```
<Plugin Name="Volume">  
<TargetVolume>25</TargetVolume>  
<LambdaVolume>1.0</LambdaVolume>  
</Plugin>
```

Surface

area
surfaceEnergy(cell)

```
<Plugin Name="Surface">  
<TargetSurface>21</TargetSurface>  
<LambdaSurface>0.5</LambdaSurface>  
</Plugin>
```

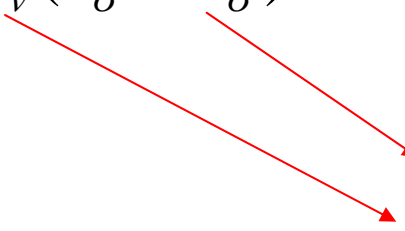
Contact

*contactEnergy(
cell1, cell2)*

```
<Plugin Name="Contact">  
<Energy Type1="Medium" Type2="Medium">0  
</Energy>  
<Energy Type1="Light" Type2="Medium">16  
</Energy>  
<Energy Type1="Dark" Type2="Medium">16  
</Energy>  
<Energy Type1="Light" Type2="Light">16.0  
</Energy>  
<Energy Type1="Dark" Type2="Dark">2.0  
</Energy>  
<Energy Type1="Light" Type2="Dark">11.0  
</Energy>  
</Plugin>
```

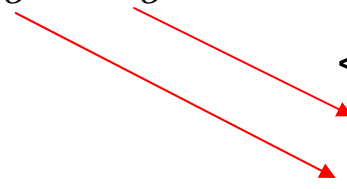
Plugin XML Syntax

$$E = \dots + \lambda_v (v_\sigma - V_\sigma)^2 + \dots$$



```
<Plugin Name="Volume">  
<TargetVolume>25</TargetVolume>  
<LambdaVolume>1.0</LambdaVolume>  
</Plugin>
```

$$E = \dots + \lambda_s (s_\sigma - S_\sigma)^2 + \dots$$



```
<Plugin Name="Surface">  
<TargetSurface>21</TargetSurface>  
<LambdaSurface>0.5</LambdaSurface>  
</Plugin>
```

Plugin XML Syntax – Contact Energy

$$E = \dots + \sum_{x,x'} J_{\tau(\sigma(x)),\tau(\sigma(x'))} (1 - \delta_{\sigma(x),\sigma(x')}) + \dots$$

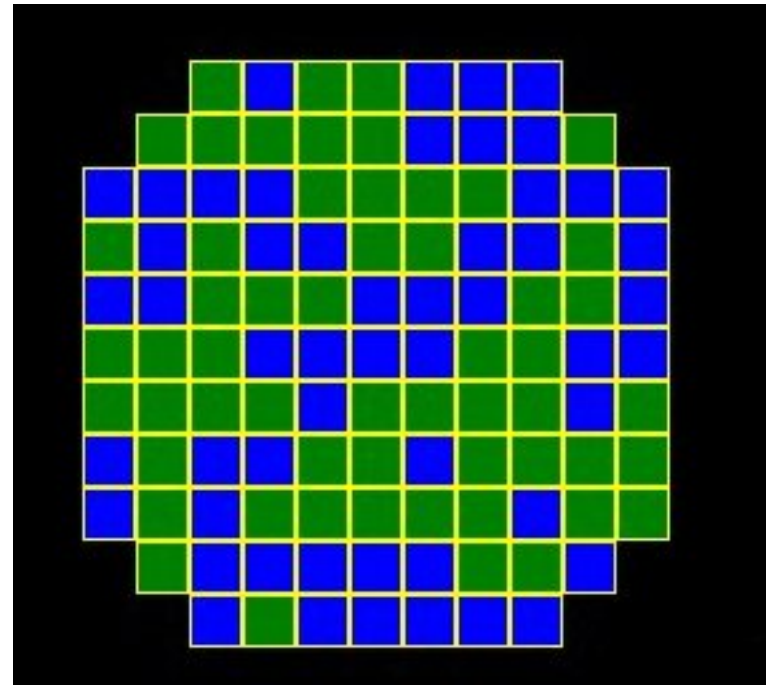
```
<Plugin Name="Contact">
  <Energy Type1="Medium" Type2="Medium">0
</Energy>
  <Energy Type1="Light" Type2="Medium">16.0
</Energy>
  <Energy Type1="Dark" Type2="Medium">16.0
</Energy>
  <Energy Type1="Light" Type2="Light">16
</Energy>
  <Energy Type1="Dark" Type2="Dark">2.0
</Energy>
  <Energy Type1="Light" Type2="Dark">11.0
</Energy>
</Plugin>
```

1- δ term ensures that pixels belonging to the same cell do not contribute to contact energy

Laying Out Cells on the Lattice

Using built-in cell field initializer:

```
<Steppable Type="BlobInitializer">  
  <Region>  
    <Radius>30</Radius>  
    <Center x="40" y="40" z="0"/>  
    <Gap>0</Gap>  
    <Width>5</Width>  
    <Types>Dark,Light</Types>  
  </Region>  
</Steppable>
```



This is just an example of cell field initializer. More general ways of cell field initialization will be discussed later.

NOTE: In actual example **Dark** cells are called **Condensing** and **Light** cells **NonCondensing**

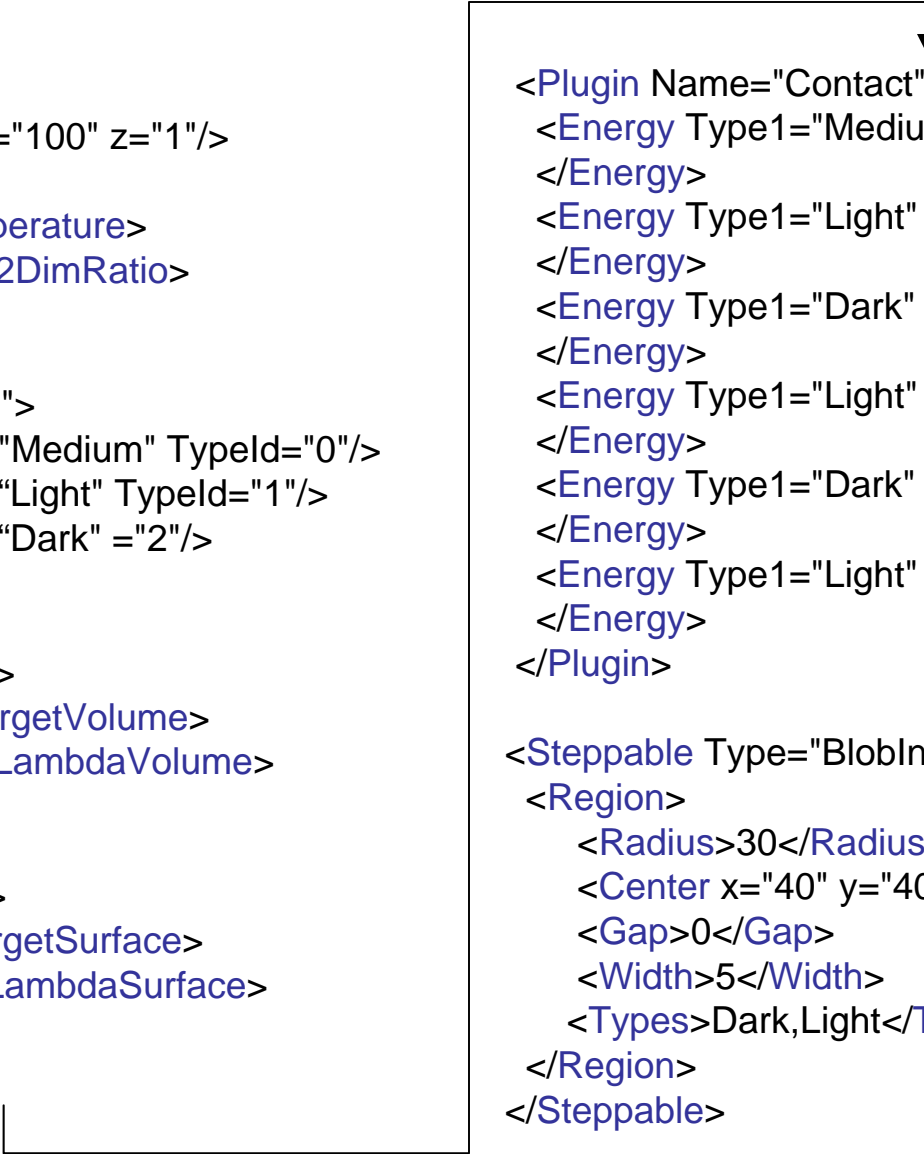
Putting It All Together - cellsort_2D.xml

```
<CompuCell3D>
  <Potts>
    <Dimensions x="100" y="100" z="1"/>
    <Steps>10</Steps>
    <Temperature>2</Temperature>
    <Flip2DimRatio>1</Flip2DimRatio>
  </Potts>

  <Plugin Name="CellType">
    <CellType TypeName="Medium" TypeId="0"/>
    <CellType TypeName="Light" TypeId="1"/>
    <CellType TypeName="Dark" TypeId="2"/>
  </Plugin>

  <Plugin Name="Volume">
    <TargetVolume>25</TargetVolume>
    <LambdaVolume>1.0</LambdaVolume>
  </Plugin>

  <Plugin Name="Surface">
    <TargetSurface>21</TargetSurface>
    <LambdaSurface>0.5</LambdaSurface>
  </Plugin>
```



```
<Plugin Name="Contact">
  <Energy Type1="Medium" Type2="Medium">0
</Energy>
  <Energy Type1="Light" Type2="Medium">16
</Energy>
  <Energy Type1="Dark" Type2="Medium">16
</Energy>
  <Energy Type1="Light" Type2="Light">16
</Energy>
  <Energy Type1="Dark" Type2="Dark">2.0
</Energy>
  <Energy Type1="Light" Type2="Dark">11
</Energy>
</Plugin>

<Steppable Type="BlobInitializer">
  <Region>
    <Radius>30</Radius>
    <Center x="40" y="40" z="0"/>
    <Gap>0</Gap>
    <Width>5</Width>
    <Types>Dark,Light</Types>
  </Region>
</Steppable>

</CompuCell3D>
```

Coding the same simulation in C/C++/Java/Fortran would take you at least 1000 lines of code...

Putting It All Together - Avoiding Common Errors in XML code

1. First specify Potts section, then list all the plugins and finally list all the steppables. This is the correct order and if you mix e.g. plugins with steppables you will get an error. Remember the correct order is

- Potts
- Plugins
- Steppables

2. Remember to match every xml tag with a closing tag

<Plugin>

...

</Plugin>

3. Watch for typos – if there is an error in the XML syntax CC3D will give you an error pointing to the location of an offending line
4. Modify/reuse available examples rather than starting from scratch – saves a lot of time

Adding Python Scripting

Begin with template code (the file will be called **cellsort_2D.py**)

```
#import useful modules
```

```
import sys
```

```
from os import environ
```

```
from os import getcwd
```

```
import string
```

```
#setup search paths
```

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

```
import CompuCellSetup
```

```
CompuCellSetup.setSimulationXMLFileName("Demos/cellsort_2D/cellsort_2D.xml")
```

```
sim,simthread = CompuCellSetup.getCoreSimulationObjects()
```

```
#Create extra player fields here or add attributes
```

```
CompuCellSetup.initializeSimulationObjects(sim,simthread)
```

```
#Add Python steppables here
```

```
steppableRegistry=CompuCellSetup.getSteppableRegistry()
```

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

Full Main Script ([examples_PythonTutorial/cellsort_2D_info_printer/cellsort_2D_info_printer.py](#)):

```
#import useful modules
```

```
import sys
```

```
from os import environ
```

```
from os import getcwd
```

```
import string
```

```
#setup search paths
```

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

```
sys.path.append(getcwd()+"/examples_PythonTutorial") #add search path
```

```
import CompuCellSetup
```

```
# tell CC3D that you will use CC3DML file together with current Python script
```

```
CompuCellSetup.setSimulationXMLFileName\
```

```
("examples_PythonTutorial/cellsort_2D_info_printer/cellsort_2D.xml")
```

```
sim,simthread = CompuCellSetup.getCoreSimulationObjects()
```

```
#Create extra player fields here or add attributes
```

```
CompuCellSetup.initializeSimulationObjects(sim,simthread)
```

```
#Add Python steppables here
```

```
steppableRegistry=CompuCellSetup.getSteppableRegistry()
```

```
from cellsort_2D_steppables import InfoPrinterSteppable
```

```
infoPrinterSteppable=InfoPrinterSteppable(_simulator=sim,_frequency=10)
```

```
steppableRegistry.registerSteppable(infoPrinterSteppable)
```

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

Implementing simple Python module for CC3D simulation:

```
class InfoPrinterSteppable(SteppableBasePy):  
    def __init__(self, _simulator, _frequency=10):  
        SteppableBasePy.__init__(self, _frequency)  
    def start(self):  
        print "This function is called once before simulation"  
  
    def step(self, mcs):  
        print "This function is called every 10 MCS"  
        for cell in self.cellList:  
            print "CELL ID=", cell.id, " CELL TYPE=", cell.type, " volume=", cell.volume
```

Notice that we have used as a base class **SteppableBasePy** instead of **SteppablePy**.

SteppableBasePy already contains members and initializations for:

self.cellList

self.simulator

self.potts

self.cellField

self.dim

self.inventory

Controlling volume of individual cells:

```
class TargetVolumeSteppable(SteppableBasePy):
    def __init__(self, _simulator, _frequency=10):
        SteppableBasePy.__init__(self, _simulator, _frequency)
    def start(self):
        for cell in self.cellList:
            cell.targetVolume=25
            cell.lambdaVolume=2
```

Cell Growth?

```
def step(self, mcs):
    for cell in self.cellList:
        cell.targetVolume+=0.3
```

Exercise – how to implement shrinkage or cell death?

Mitosis in CompuCell3D simulations

Supporting cell division (mitosis) in CompuCell3D simulations is a prerequisite for building faithful biomedical simulations.

You can use mitosis module (Mitosis Plugin) directly from XML however, its use will be very limited because of the following fact:

After cell division you end up with two cells. **What parameters should those two cells have (type, target volume etc.)? How do you modify the parameters?**

The best solution is to manage mitosis from Python and the example below will explain you how to do it.

There are two ways to implement mitosis – as a plugin or as a steppable. On older versions we have used plugin-based approach as this was the only option. However steppable based approach is much simpler to implement and we will focus on it first.

```
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()
```

```
#add additional attributes
```

```
pyAttributeAdder,listAdder=CompuCellSetup.attachListToCells(sim)
```

```
CompuCellSetup.initializeSimulationObjects(sim,simthread)
```

```
import CompuCell #notice importing CompuCell to main script has to be done after call to  
getCoreSimulationObjects()
```

```
#Add Python steppables here
```

```
steppableRegistry=CompuCellSetup.getSteppableRegistry()
```

```
from cellsort_2D_field_modules import VolumeConstraintSteppable
```

```
volumeConstraint=VolumeConstraintSteppable(sim)
```

```
steppableRegistry.registerSteppable(volumeConstraint)
```

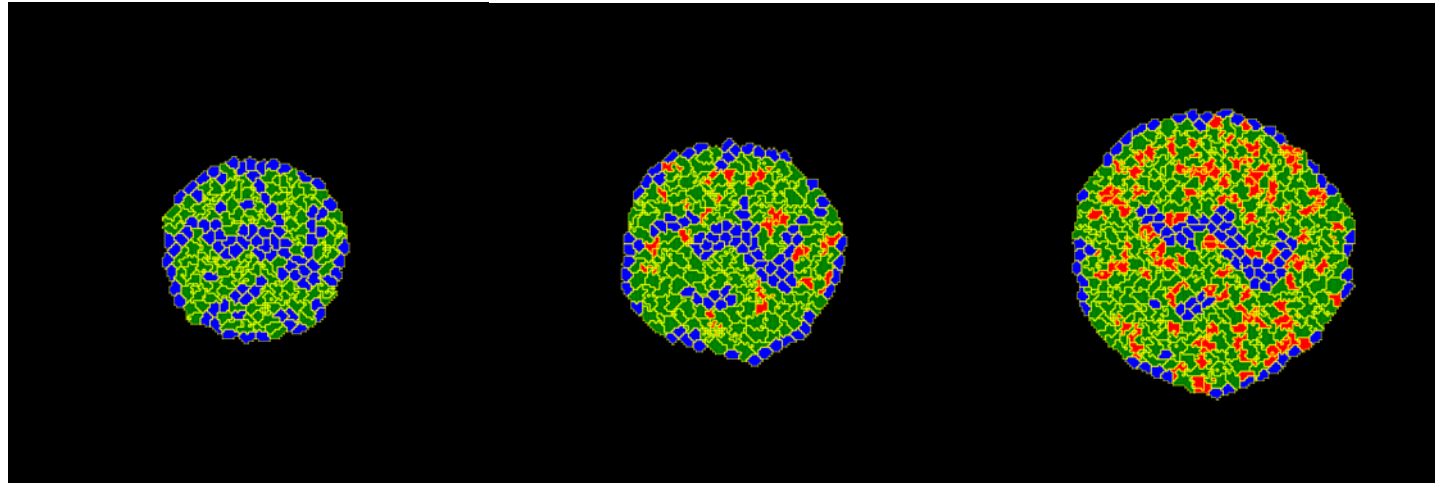
```
from cellsort_2D_field_modules import MitosisSteppable
```

```
mitosisSteppable=MitosisSteppable(sim,1)
```

```
steppableRegistry.registerSteppable(mitosisSteppable)
```

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

Mitosis example results



t=200 MCS

t=600 MCS

t=1000 MCS

“Green” cells grow in response to diffusing FGF. Once they reach doubling volume they divide. They have 50% probability of differentiating into “red” cells. After 1500 MCS we gradually decrease target volume of each cell, effectively killing them.

Mitosis is implemented as a steppable class `MitosisSteppable` which inherits from `MitosisSteppableBase`

```
class MitosisSteppable(MitosisSteppableBase):
    def __init__(self, _simulator, _frequency=1):
        MitosisSteppableBase.__init__(self, _simulator, _frequency)

    def step(self, mcs):
        cells_to_divide=[]
        for cell in self.cellList:
            if cell.volume>50:
                cells_to_divide.append(cell)

        for cell in cells_to_divide:
            self.divideCellRandomOrientation(cell)

    def updateAttributes(self):
        parentCell=self.mitosisSteppable.parentCell
        childCell=self.mitosisSteppable.childCell

        parentCell.targetVolume/=2.0
        childCell.targetVolume=parentCell.targetVolume
        childCell.lambdaVolume=parentCell.lambdaVolume
        if (random())<0.5:
            childCell.type=parentCell.type
        else:
            childCell.type=3
```

Keeping track of mitotic history of cells – you may skip this during first reading

In CompuCell3D simulations each cell by default will have several attributes such as volume, surface, centroids, target volume, cell id etc.

One can write a plugin that **attaches additional attributes to a cell during run time**. Doing so avoids recompilation of entire CompuCell3D but requires to write and compile the C++ plugin.

It is by far the easiest to attach additional cell attribute in Python. Not only there is no need to recompile anything, but the actual task takes one line of code:

```
pyAttributeAdder,listAdder=CompuCellSetup.attachListToCells(sim)
```

Above we told CompuCell3D to attach a Python list to each cell that will be produced by the CompuCell3D kernel.

We can access this list very easily from Python level. Python list is dynamic data structure that can grow or shrink and can hold arbitrary Python objects. Therefore by attaching a list to each cell we effectively came up with a way to attach any cell attribute.

We may also attach dictionary instead of the list:

```
pyAttributeAdder,dictAdder=CompuCellSetup.attachDictionaryToCells(sim)
```

And everything takes place during run time...

Full listing of simulation where each cell gets extra attribute – a list:

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

sys.path.append(environ["PYTHON_MODULE_PATH"])
sys.path.append(getcwd()+"/examples_PythonTutorial")

import CompuCellSetup

CompuCellSetup.setSimulationXMLFileName("examples_PythonTutorial/cellsort_2D_extra_attrib/cellsort_2D.xml")

sim,simthread = CompuCellSetup.getCoreSimulationObjects()

#Create extra player fields here or add attributes
pyAttributeAdder,listAdder=CompuCellSetup.attachDictionaryToCells(sim)

CompuCellSetup.initializeSimulationObjects(sim,simthread)

#Add Python steppables here
steppableRegistry=CompuCellSetup.getSteppableRegistry()

#here we will add ExtraAttributeCellsort steppable
from cellsort_2D_steppables import ExtraAttributeCellsort
extraAttributeCellsort=ExtraAttributeCellsort(_simulator=sim,_frequency=10)
steppableRegistry.registerSteppable(extraAttributeCellsort)

from cellsort_2D_steppables import TypeSwitcherSteppable
typeSwitcherSteppable=TypeSwitcherSteppable(sim,100)
steppableRegistry.registerSteppable(typeSwitcherSteppable)

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

ExtraAttributeCellsort

```
class ExtraAttributeCellsort(SteppablePy):
    def __init__(self, _simulator, _frequency=10):
        SteppablePy.__init__(self, _frequency)
        self.simulator=_simulator
        self.inventory=self.simulator.getPotts().getCellInventory()
        self.cellList=CellList(self.inventory)

    def step(self, mcs):
        for cell in self.cellList:
            pyAttrib=CompuCell.getPyAttrib(cell)
            pyAttrib["CellData"]=[cell.id*mcs , cell.id*(mcs-1)]
            print "CELL ID modified=", pyAttrib["CellData"]
```

Manipulating dictionary attached to the cell

Notice, you may also attach a **list to a cell instead of a dictionary**. See Python Scripting Tutorials for more information. Dictionaries are more useful than lists though in the CompuCell3D context so make sure you understand them and know how to attach them to cells.

Adding mitosis history to parent and child cells:

**#Mitosis data has to have base class "object" otherwise if cell will be deleted CC3D
#may crash due to improper garbage collection**

```
class MitosisData(object):
    def __init__(self, _MCS=-1, _parentId=-1, _parentType=-1,\
                 _offspringId=-1, _offspringType=-1):
        self.MCS=_MCS
        self.parentId=_parentId
        self.parentType=_parentType
        self.offspringId=_offspringId
        self.offspringType=_offspringType
    def __str__(self):
        return "Mitosis time="+str(self.MCS)+" parentId="\
              +str(self.parentId)+" offspringId="+str(self.offspringId)
```

```
def updateAttributes(self):
    parentCell=self.mitosisSteppable.parentCell
    childCell=self.mitosisSteppable.childCell

    .....
```

#get a reference to lists storing Mitosis data

```
parentCellList=CompuCell.getPyAttrib(parentCell)
childCellList=CompuCell.getPyAttrib(childCell)
##will record mitosis data in parent and offspring cells
mcs=self.simulator.getStep()
mitData=MitosisData(mcs,parentCell.id,parentCell.type,childCell.id,childCell.type)
parentCellList.append(mitData)
childCellList.append(mitData)
```

Cell-attributes revisited - **VERY IMPORTANT:**

- In the previous example – *examples_PythonTutorial/cellsort_2D_extra_attrib* - we were attaching integers as additional cell attributes.
- If we define our own class in the most straightforward way and try to append objects of this class to the list of attributes of a cell, it may happen that CompuCell3D will crash when such cell gets deleted (e.g. in a simulations where some of the cells die).
- It turns out that with exception of Python “core” objects such as integers or floating point numbers adding user defined class which DOES NOT inherit from Python “object” leads to improper garbage collection, and this causes overall CC3D crash
- The solution: Always inherit from “object” when you want to add objects of your custom class as cell attribute:

```
class CellPosition(object):
```

```
    def __init__(self, _x=0, _y=0, _z=0):
```

```
        self.x=_x
```

```
        self.y=_y
```

```
        self.z=_z
```

Directional Mitosis

So far we have divided cells using randomly chosen division axis/plane:

```
class MitosisSteppable(MitosisSteppableBase):
```

```
.....
```

```
def step(self, mcs):
```

```
    cells_to_divide=[]
```

```
    for cell in self.cellList:
```

```
        if cell.volume>50:
```

```
            cells_to_divide.append(cell)
```

```
    for cell in cells_to_divide:
```

```
        self.divideCellRandomOrientation(cell)
```

However MitosisSteppableBase, and consequently any class which inherits from MitosisSteppableBase, allows additional modes of division:

- Along major axis/plane of a cell
- Along minor axis/plane of a cell
- Along user specified axis/plane of the cell

By changing one function name in the Mitosis Steppable we can cause cells to divide along e.g. Major axis:

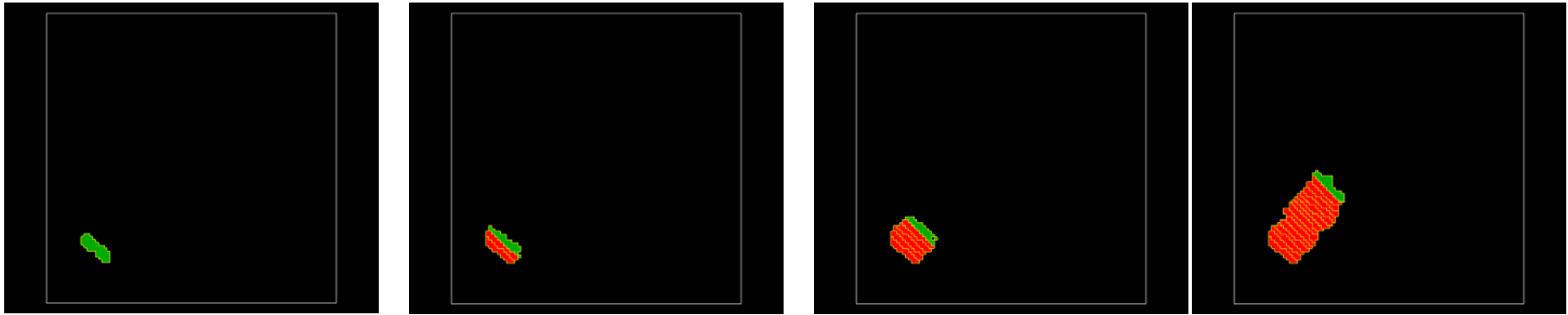
```
class MitosisSteppable(MitosisSteppableBase):
    def __init__(self, _simulator, _frequency=1):
        MitosisSteppableBase.__init__(self, _simulator, _frequency)

    def step(self, mcs):
        cells_to_divide=[]
        for cell in self.cellList:
            if cell.volume>50:
                cells_to_divide.append(cell)

        for cell in cells_to_divide:
            self.divideCellAlongMajorAxis(cell)
            # self.divideCellOrientationVectorBased(cell,1,1,0)
            # self.divideCellAlongMajorAxis(cell)
```

Commented lines show addition options in choosing orientation of division axis. Notice that when specifying user-defined orientation we simply specify vector along which the division will take place. The vector does not have to be normalized

Results of dividing cells along major axis



Try running `examples_PythonTutorial/steppableBasedMitosis` example and change division axis to major axis. Do you get similar results as the one shown above?

What do you have to modify to achieve similar picture?

Hint: look at

`examples_PythonTutorial/growingcells_fast/growingcells_fast_directional.xml`

Flexible Diffusion Solver

```
<Steppable Type="FlexibleDiffusionSolverFE">
  <DiffusionField>
    <DiffusionData>
      <FieldName>FGF</FieldName>
      <DiffusionConstant>0.010</DiffusionConstant>
      <DecayConstant>0.000</DecayConstant>
      <ConcentrationFileName>diffusion_2D.pulse.txt</ConcentrationFileName>
    </DiffusionData>
  </DiffusionField>
</Steppable>
```

Define diffusion field

Define diffusion parameters

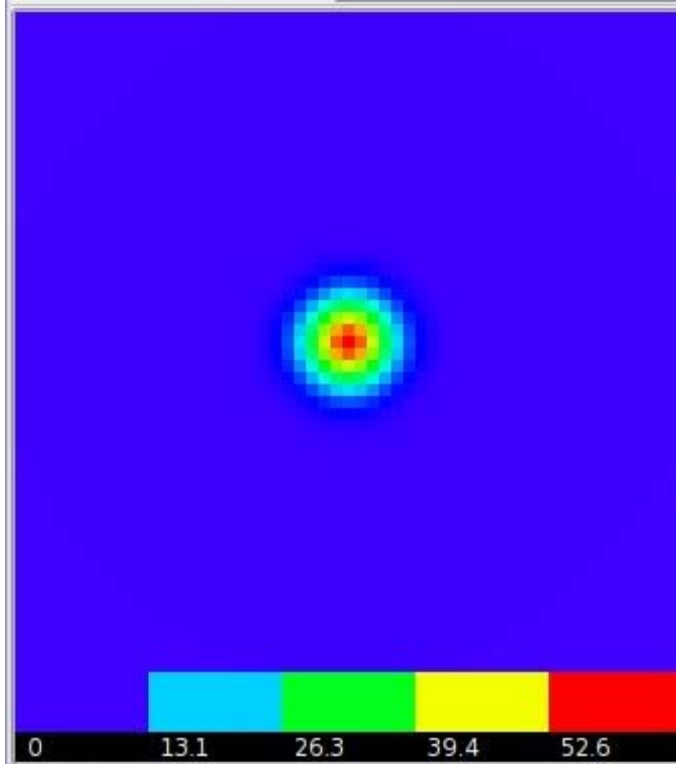
Read-in initial condition

Initial Condition File Format:

x y z concentration

Example:

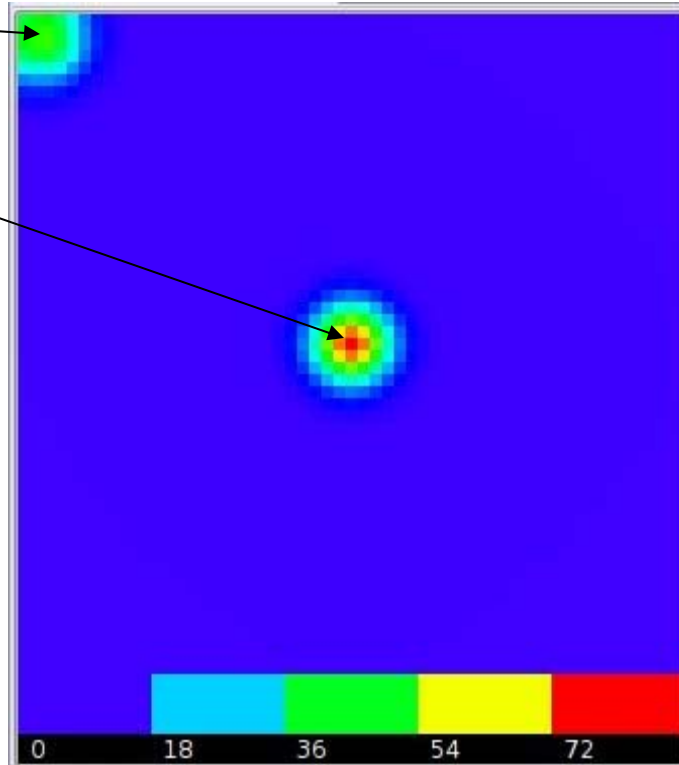
```
27 27 0 2000.0
45 45 0 0.0 ...
```



Two-pulse initial condition

Initial condition (`diffusion_2D.pulse.txt`):

5 5 0 1000.0
27 27 0 2000.0



Accessing concentrations from Python

```
def step(self,mcs):  
    field=CompuCell.getConcentrationField(self.simulator,"FGF")  
    pt=CompuCell.Point3D()  
    for cell in self.cellList:  
        pt.x=int(cell.xCM/cell.volume)  
        pt.y=int(cell.yCM/cell.volume)  
  
    print "concentration at COM for cell with ID=",cell.id," is ",field.get(pt)
```

Putting things together

- By combining cell growth, concentration fields, mitosis, cell type differentiation you can start building biologically interesting simulations.
- Adding more “ingredients” to simulations does not require any more knowledge. All you have to know is how to copy, paste, and modify code snippets.

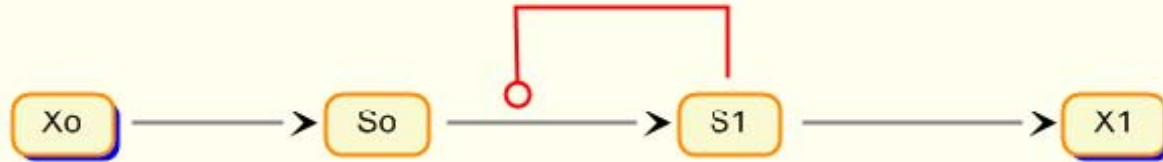
Linking SBML models to describe cell properties

- Specifying cell properties individually is quite straight forward but better approach is to use underlying molecular model to control cell behavior.
- Use soslib Python wrapper developed by Ryan Roper (UW)
- Final release is pending

Molecular Circuits Inside Each Cell

- Define molecular pathway model in Jarnac (or if you prefer any other tool which exports SBML)
- Save pathway model in SBML format
- Load SBML model to each CompuCell3D cell
- Solve SBML model
- Make cell properties of cell depend on the current state of the SBML model

Bistable Oscillator System



Pathway description in Jarnac (get SBW from sys-bio.org)

```
p = defn bistable_oscillator
```

```
$X0 -> S0; k0*X0;
```

```
S0 -> S1; k1*S0 + Vmax*S0*S1^n/(15 + S1^n);
```

```
S1 -> $X1; k2*S1;
```

```
end;
```

Initial Conditions

```
p.X0 = 1;
```

```
p.X1 = 0;
```

```
p.S1 = 1;
```

```
p.n = 4;
```

```
p.Vmax = 12;
```

```
p.k0 = 0.044;
```

```
p.k1 = 0.01;
```

```
p.k2 = 0.1;
```


Controlling cell-cell adhesion using molecular toy circuit

- Inside each cell we run (i.e. solve) reaction kinetics model – specified in SBML format
- We extract concentration of molecular species to control cell-cell adhesion
- We may conveniently plot time series of species concentration in CC3D