Introduction to Microphysiological Simulations Using MCell

Markus Dittrich, Ph.D.
dittrich@psc.edu

National Resource for Biomedical Supercomputing, PSC, CMU
Dept. of Computational and Systems Biology, Univ. of Pittsburgh

May 7, 2012
1. Use The Proper Tool to Study Your Problem

2. MCell, a Powerful Tool for Computational Microphysiology

3. Software Pipeline For Building a Microphysiological Model

4. Geometry Creation, Mesh Generation, Annotation

5. Non-spatial Model Parameters

6. Simulate Your Model

7. Visualize and Analyse Results

8. MCell - Basic Definitions and Units

9. Your First Model
The MCell Team

- Tom Bartol (Salk Institute)
- Joel Stiles (in Memoriam)
- Jacob Czech (NRBSC)
- Markus Dittrich (NRBSC)
- Jim Faeder (Univ. of Pittsburgh)
- Boris Kaminski (NBRSC)
<table>
<thead>
<tr>
<th>Problem/Method</th>
<th>Typical Application</th>
<th>Software Examples</th>
<th>Resolution (Scale)</th>
<th>Spatial Realism</th>
<th>Stochastic Realism</th>
<th>Time Step</th>
<th>Time Scale</th>
<th>Serial/Parallel</th>
<th>Computer Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Networks of Reactions/Sets of ODEs</td>
<td>Metabolic or signaling pathways</td>
<td>Virtual Cell, ECell, Gepasi, XPPAUT</td>
<td>N/A (cell)</td>
<td>N/A</td>
<td>&lt;none&gt;</td>
<td>ms</td>
<td>ms - hrs</td>
<td>serial</td>
<td>minimal</td>
</tr>
<tr>
<td>Excitation/Compartmental Circuit</td>
<td>Nerve signaling</td>
<td>NEURON, GENESIS, NEOSIM</td>
<td>μm - mm (cell - multicell)</td>
<td>low - medium</td>
<td>none</td>
<td>ms</td>
<td>ms - hrs</td>
<td>usually serial</td>
<td>usually low</td>
</tr>
<tr>
<td>Reaction Kinetics/Stochastic</td>
<td>Gene regulation/transcription</td>
<td>BioSpice, StochSim, XPPAUT, MCell</td>
<td>N/A (cell)</td>
<td>N/A</td>
<td>high</td>
<td>ms</td>
<td>ms - hrs</td>
<td>serial</td>
<td>low</td>
</tr>
<tr>
<td>3-D Reaction Diffusion/Finite Element</td>
<td>Flow models, calcium dynamics</td>
<td>Virtual Cell, FIDAP, Kaskade</td>
<td>&lt;μm (cell)</td>
<td>medium-high</td>
<td>&lt;none&gt;</td>
<td>μs - ms</td>
<td>μs - sec</td>
<td>either</td>
<td>low - high</td>
</tr>
<tr>
<td>3-D Reaction Diffusion/Monte Carlo</td>
<td>Micro-physiological processes</td>
<td>MCell, ChemCell, SmolDyn</td>
<td>nm – mm (subcell - cell)</td>
<td>high</td>
<td>high</td>
<td>ps - ms</td>
<td>μs - sec</td>
<td>either</td>
<td>low - high</td>
</tr>
<tr>
<td>Macromolecular Machinery/GNM</td>
<td>Collective dynamics</td>
<td>GNM, ANM</td>
<td>Å - 100 nm (complexes)</td>
<td>high</td>
<td>none</td>
<td>N/A</td>
<td>&lt;ns – μs&gt;</td>
<td>N/A (analytic)</td>
<td>minimal</td>
</tr>
<tr>
<td>Diffusion in Potential Field/Poisson-Nernst-Planck</td>
<td>Electrostatic interactions, ion channels</td>
<td>UHBD, Delphi, CHARMM</td>
<td>Å - nm (membrane proteins)</td>
<td>high (implicit solvent)</td>
<td>none</td>
<td>N/A</td>
<td>&lt;ns – μs&gt;</td>
<td>parallel</td>
<td>low - medium</td>
</tr>
<tr>
<td>Macromolecular Motions/Brownian Dynamics (BD)</td>
<td>Conformational dynamics (in flow fields)</td>
<td>CHARMM, GROMOS, UHBD</td>
<td>Å - nm (macromolecules)</td>
<td>high (implicit solvent)</td>
<td>high</td>
<td>5 - 10 fs</td>
<td>&lt;ns – μs&gt;</td>
<td>parallel</td>
<td>medium - high</td>
</tr>
<tr>
<td>Molecular Structure/Molecular Dynamics (MD)</td>
<td>Conformational dynamics &amp; free energies</td>
<td>NAMD, AMBER, CHARMM, GROMOS</td>
<td>Å (macromolecules)</td>
<td>exact (explicit solvent)</td>
<td>exact</td>
<td>1 - 2 fs</td>
<td>&lt;ns – μs&gt;</td>
<td>parallel</td>
<td>very high</td>
</tr>
<tr>
<td>Molecular Structure/Ab initio simulations</td>
<td>Solution of the Schrodinger equation</td>
<td>Gaussian</td>
<td>&lt;Å (electrons - atoms)</td>
<td>exact</td>
<td>exact</td>
<td>N/A</td>
<td>N/A</td>
<td>parallel</td>
<td>highest</td>
</tr>
</tbody>
</table>
MCell is a Monte Carlo reaction-diffusion simulator for modeling computational microphysiology in arbitrarily complex 3D spatial geometries.

The simulation of biological systems at micron to millimeter length scales (subcellular to cellular) using realistic 3D geometry over biological timescales from $ns$ to $ms$ to $s$. 
Components of a Microphysiological Model
Software Pipeline to Build a Microphysiological Model

Create Geometry

Generate Meshes

Annotate Meshes

Specify Non-Spatial Model Parameters

Simulate Model

Visualize and Analyze Results
Model geometries for MCell simulations can be obtained via:

1. Reconstruction of model mesh geometry from electron microscopy data
Geometry Creation, Mesh Generation, Annotation

Model geometries for MCell simulations can be obtained via:

1. Reconstruction of model mesh geometry from electron microscopy data

2. *In silico* geometry construction (e.g. via CellBlender) based on known average geometry
Show NMJ recon movie
Show 24 active zone model
Model geometry via EM reconstruction

pros: highly accurate, significant structural and spatial detail that may be crucial for the underlying biology

cons: segmentation and mesh generation from EM stacks is difficult, time consuming; no well developed computational pipeline
Which Geometry Construction Method Is Best?

Model geometry via EM reconstruction

**pros:** highly accurate, significant structural and spacial detail that may be crucial for the underlying biology

**cons:** segmentation and mesh generation from EM stacks is difficult, time consuming; no well developed computational pipeline

*In silico* geometry generation

**pros:** straightforward (and fast) with CellBlender; easy to explore different geometry; meshes often computationally less expensive; focuses on the "essential" features

**cons:** less accurate
Software Pipeline to Build a Microphysiological Model

1. Create Geometry
2. Generate Meshes
3. Annotate Meshes
4. Specify Non-Spatial Model Parameters
5. Simulate Model
6. Visualize and Analyze Results
Beyond a geometry specification, MCell models typically also contain

- **Molecules**: can diffuse in space (volume molecules) or on mesh surfaces (surface molecules); need diffusion coefficients, concentrations/densities, locations.

- **Reactions**: requires knowledge of all elementary reactions (uni and bimolecular) mechanisms and their reaction rate constants.

- **Output Specification**: request output for visualisation and analysis purposes; may require definition of additional geometry objects such as counting boxes.
Software Pipeline to Build a Microphysiological Model

Create Geometry

Generate Meshes

Annotate Meshes

Specify Non-Spatial Model Parameters

Simulate Model

Visualize and Analyze Results
At present, running of MCell simulations is done on the command line on UNIXy operating systems (Linux, Mac OSX). Future integration into CellBlender is planned.

```bash
# mcell my_model.mdl
```

`my_model.mdl` contains the complete model description as Model Description Language (MDL).
At present, running of MCell simulations is done on the command line on UNIXy operating systems (Linux, Mac OSX). Future integration into CellBlender is planned.

```bash
# mcell my_model.mdl
```

`my_model.mdl` contains the complete model description as Model Description Language (MDL).

```bash
# mcell -seed 10 my_model.mdl
```
Software Pipeline to Build a Microphysiological Model

1. Create Geometry
2. Generate Meshes
3. Annotate Meshes
4. Specify Non-Spatial Model Parameters
5. Simulate Model
6. Visualize and Analyze Results
MCell can produce two types of output (you specify how much of each, what molecules, etc.):

- **Visualization Output**, contains the specification of geometric meshes and the location and orientation of surface and volume molecules. Can be visualized using CellBlender (previously DReAMM).
- **Reaction data output**, contains timeseries of
  - volume molecules counts in specified regions of the model
  - surface molecule counts on specified regions of the model
  - reaction counts
  - hits or crossings of surface regions by volume molecules

Reaction data is typically analyzed using external tools, e.g. R, python, octave, MATLAB, etc.
The root cause for the loss of the MCO spacecraft was the failure to use metric units in the coding of a ground software file. The output from the SM_FORCES application code as required by a MSOP Project Software Interface Specification (SIS) was to be in metric units of Newtonseconds (N-s). Instead, the data was reported in English units of pound-seconds (lbf-s). An erroneous trajectory was computed using this incorrect data.

Don't let your MCell model become the next MCO!
The MCO MIB has determined that the root cause for the loss of the MCO spacecraft was the failure to use metric units in the coding of a ground software file [...]. The output from the SM_FORCES application code as required by a MSOP Project Software Interface Specification (SIS) was to be in metric units of Newton-seconds (N-s). Instead, the data was reported in English units of pound-seconds (lbf-s). [...]. An erroneous trajectory was computed using this incorrect data.

Don't let your MCell model become the next MCO!
Mars Climate Orbiter, September 23, 1999

The MCO MIB has determined that the root cause for the loss of the MCO spacecraft was the failure to use metric units in the coding of a ground software file [...] The output from the SM_FORCES application code as required by a MSOP Project Software Interface Specification (SIS) was to be in metric units of Newtonseconds (N-s). Instead, the data was reported in English units of pound-seconds (lbf-s). [...] An erroneous trajectory was computed using this incorrect data.
Mars Climate Orbiter, September 23, 1999

The MCO MIB has determined that the root cause for the loss of the MCO spacecraft was the failure to use metric units in the coding of a ground software file [...] The output from the SM_FORCES application code as required by a MSOP Project Software Interface Specification (SIS) was to be in metric units of Newtonseconds (N-s). Instead, the data was reported in English units of pound-seconds (lbf-s). [...] An erroneous trajectory was computed using this incorrect data.

Don’t let your MCell model become the next MCO!
• Spatial dimensions are in units of microns \((\mu m, 10^{-6} m)\)
• Time is in units of seconds
• Each simulation runs for a specified number of iterations
• Each iteration corresponds to a duration of one timestep
MCell models are described using Model Description Language (MDL).

MDL commands are written in ALL CAPS and consist of

- simple statements of the form

  \[
  \text{MDL\_COMMAND} = \text{<value>}
  \]

- statement blocks enclosed in curly braces describing a certain aspect of the simulation (molecule definition, geometry definition)

  \[
  \text{MDL\_BLOCK} \{ \text{<block content>} \}
  \]

where \text{<block content>} are other MDL\_COMMANDs or MDL\_BLOCKs.
Each MCell simulation needs to define the simulation *timestep* (TIME_STEP) and the number of *iterations* (ITERATIONS).

```
/* define variables for timestep and iterations */
iters = 100
dt = 1e-6

/* use variable to set actual keywords */
ITERATIONS = iters
TIME_STEP = dt
```

**Notes:**
- Within MDL you can define arbitrary variables (such as iters, dt).
- MDL is case sensitive - do not use ALL_CAPS for variables to avoid clashes with MCell keywords.
- You can use C-style comments.
Molecules are defined within a `DEFINE_MOLECULES` block. Our model will contain two volume molecules `Vol1`, `Vol2` and a surface molecule `Surf`.

```
first_model.mdl

iters = 100
dt = 1e-6
ITERATIONS = iters
TIME_STEP = dt

DEFINE_MOLECULES {
    Vol1 {DIFFUSION_CONSTANT_3D = 1E-6}
    Vol2 {DIFFUSION_CONSTANT_3D = 1E-6}
    Surf {DIFFUSION_CONSTANT_2D = 1E-7}
}
```

Notes:

- Volume and surface molecules are distinguished via providing either a 3D or 2D diffusion coefficient.
- The units of the diffusion coefficient are $cm^2 s^{-1}$.
- Molecules in MCell are point particles and have no spatial dimension.
Next, we need to construct a mesh which defines the 3D geometry of our model. Using CellBlender we create a simple cube shaped object and export it in MDL format, `Cube.mdl`. 
Then we add `Cube.mdl` to our model MDL file.

```mdl
first_model.mdl

iters = 100
dt = 1e-6
ITERATIONS = iters
TIME_STEP = dt

DEFINE_MOLECULES {
    Vol1 {DIFFUSION_CONSTANT_3D = 1E-6}
    Vol2 {DIFFUSION_CONSTANT_3D = 1E-6}
    Surf {DIFFUSION_CONSTANT_2D = 1E-7}
}

INCLUDE_FILE = "./Cube.mdl"

INSTANTIATE World OBJECT {
    Cube OBJECT Cube{}
}
```

Notes:

- MDL files can include other MDL files which is very useful for organizing them, e.g., according to topic (geometry description, molecule definitions, etc.).
Cube POLYGON_LIST
{
    VERTEX_LIST
    {
        [ 1, 0.999999940395355, -1 ]
        [ 1, -1, -1 ]
        [ -1.00000011920929, -0.999999821186066, -1 ]
        [ -0.999999642372131, 1.00000035762787, -1 ]
        [ 1.00000047683716, 0.999999463558197, 1 ]
        [ 0.999999344348907, -1.00000059604645, 1 ]
        [ -1.000000035762787, -0.999999642372131, 1 ]
        [ -0.99999940395355, 1, 1 ]
    }
    ELEMENT_CONNECTIONS
    {
        [ 4, 0, 3 ]
        [ 4, 3, 7 ]
        [ 2, 6, 7 ]
        [ 2, 7, 3 ]
        [ 1, 5, 2 ]
        [ 5, 6, 2 ]
        [ 0, 4, 1 ]
        [ 4, 5, 1 ]
        [ 4, 7, 5 ]
        [ 7, 6, 5 ]
        [ 0, 1, 2 ]
        [ 0, 2, 3 ]
    }
    DEFINE_SURFACE_REGIONS
    {
        top
        {
            ELEMENT_LIST = [8, 9]
        }
    }
}
Notes:

- Our Cube object is a POLYGON_LIST object, the most general way to specify a geometrical shape within mdl.
- Other geometry objects are BOX objects and RELEASE_SITEs for molecules.
- DEFINE_SURFACE_REGIONS allows to group subsets of triangles into surface regions.
- Meshes are by default reflective to all diffusing volume molecules but can be made absorptive or transparent via surface classes.
Your First Model - Defining Geometry

first_model.mdl

...  

INCLUDE_FILE = "./Cube.mdl"

INSTANTIATE World OBJECT {
   Cube OBJECT Cube{}
}

...

Notes:

- Geometry objects can be combined into "meta" objects (here a single one called World).
- In addition to adding existing objects, within meta objects new geometry objects can be created, or existing objects can be transformed (translation, rotation, scaling).
- An MCell model must contain at least one INSTANTIATED object.
- Meta objects can be nested, i.e. contain other meta objects.
Up to this point we only **defined** the molecules in our simulation but we haven’t **released** any!

### first_model.mdl

```md
... 
DEFINE_MOLECULES {
  Vol1 {DIFFUSION_CONSTANT_3D = 1E-6}
  Vol2 {DIFFUSION_CONSTANT_3D = 1E-6}
  Surf {DIFFUSION_CONSTANT_2D = 1E-7}
}

INCLUDE_FILE = "./Cube.mdl"

INSTANTIATE World OBJECT {
  Cube OBJECT Cube{}

  vol1_rel RELEASE_SITE {
    SHAPE = World.Cube
    MOLECULE = Vol1
    NUMBER_TO_RELEASE = 2000
  }

  surf1_rel RELEASE_SITE {
    SHAPE = World.Cube[top]
    MOLECULE = Surf'
    NUMBER_TO_RELEASE = 2000
  }
}
```

- Volume and surface molecules can be released within RELEASE_SITE blocks.
Up to this point we only **defined** the molecules in our simulation but we haven’t **released** any!

```plaintext
... DEFINE_MOLECULES {
  Vol1 {DIFFUSION_CONSTANT_3D = 1E-6}
  Vol2 {DIFFUSION_CONSTANT_3D = 1E-6}
  Surf {DIFFUSION_CONSTANT_2D = 1E-7}
}

INCLUDE_FILE = "./Cube.mdl"

INSTANTIATE World OBJECT {
  Cube OBJECT Cube{}

  vol1_rel RELEASE_SITE {
    SHAPE = World.Cube
    MOLECULE = Vol1
    NUMBER_TO_RELEASE = 2000
  }

  surf1_rel RELEASE_SITE {
    SHAPE = World.Cube[top]
    MOLECULE = Surf'
    NUMBER_TO_RELEASE = 2000
  }
}
```

- Volume and surface molecules can be released within **RELEASE_SITE** blocks.
- For volume molecules the **SHAPE** keyword defines the **closed** geometry object (qualified starting from the instantiated top level object) in which to release the molecules.
Up to this point we only **defined** the molecules in our simulation but we haven’t **released** any!

```plaintext
first_model.mdl

... 

DEFINE_MOLECULES {
  Vol1 {DIFFUSION_CONSTANT_3D = 1E-6}
  Vol2 {DIFFUSION_CONSTANT_3D = 1E-6}
  Surf {DIFFUSION_CONSTANT_2D = 1E-7}
}

INCLUDE_FILE = ".../Cube.mdl"

INSTANTIATE World OBJECT {
  Cube OBJECT Cube{
    vol1_rel RELEASE_SITE {
      SHAPE = World.Cube
      MOLECULE = Vol1
      NUMBER_TO_RELEASE = 2000
    }
    surf1_rel RELEASE_SITE {
      SHAPE = World.Cube[top]
      MOLECULE = Surf'
      NUMBER_TO_RELEASE = 2000
    }
  }
}
```

- Volume and surface molecules can be released within `RELEASE_SITE` blocks.
- For volume molecules the `SHAPE` keyword defines the **closed** geometry object (qualified starting from the instantiated top level object) in which to release the molecules.
- For surface molecules the `SHAPE` keyword defines a surface region on a geometry object (here `top` on `World.Cube` via the `GeometryObject[<region specifier>]` syntax.
Surface molecules have a top and bottom domain. Surfaces (membranes) have a front and back.
Surface molecules have a top and bottom domain.

Surfaces (membranes) have a front and back.

KcsA (image from Wikipedia)
Surface molecules have a top and bottom domain. Surfaces (membranes) have a front and back.

KcsA (image from Wikipedia)
Surface molecules have a **top** and **bottom** domain.

KcsA (image from Wikipedia)
Surface molecules have a top and bottom domain.
Surface molecules have a top and bottom domain.
Surfaces (membranes) have a front and back.

KcsA (image from Wikipedia)
How does MCell handle surface molecule orientation?

- mesh tiles and thus surfaces have a unique front and back according to the right hand rule.

**Note:** Your mesh tiles need to have a consisted orientation!

- surface molecules are either placed with their top at the front or the back of the surface!

- each mesh element is subdivided into (triangular) tiles which can each accommodate only a single surface molecules; surface molecules ”acquire” a certain surface area. The tile density can be set via SURFACE_GRID_DENSITY (default 10000 $\mu m^{-2}$).
How does MCell handle surface molecule orientation?

- mesh tiles and thus surfaces have a unique front and back according to the *right hand rule*.  
  **Note:** Your mesh tiles need to have a consisted orientation!
- surface molecules are either placed with their top at the front or the back of the surface!
- each mesh element is subdivided into (triangular) tiles which can each accommodate only a single surface molecules; surface molecules ”acquire” a certain surface area. The tile density can be set via `SURFACE_GRID_DENSITY` (default 10000 \(\mu m^{-2}\)).
Now we can understand MCell's surface molecule placement syntax.

- `Surf’` places the molecule with its top at the FRONT of the surface.

```plaintext
first_model.mdl

... 
surf1_rel RELEASE_SITE {
  SHAPE = World.Cube[top]
  MOLECULE = Surf’
  NUMBER_TO_RELEASE = 2000
}
...
```
Now we can understand MCell’s surface molecule placement syntax.

```
first_model.mdl
...
  surf1_rel RELEASE_SITE {
    SHAPE = World.Cube[top]
    MOLECULE = Surf'
    NUMBER_TO_RELEASE = 2000
  }
...
```

- **Surf’** places the molecule with its top at the **FRONT** of the surface.

```
  FRONT  T  BACK
    B
```

- **Surf,** places the molecule with its top at the **BACK** of the surface.

```
  FRONT  B  BACK
    T
```

Now we can understand MCell’s surface molecule placement syntax.

*Surf’* places the molecule with its top at the FRONT of the surface.

*Surf, places the molecule with its top at the BACK of the surface.

*Surf; places the molecule with its top randomly at the FRONT or BACK of the surface.
Now that we have defined molecules and placed them in the world we can define reactions between them.

```
... 
DEFINE_MOLECULES {
    Vol1 {DIFFUSION_CONSTANT_3D = 1E-6} 
    Vol2 {DIFFUSION_CONSTANT_3D = 1E-6} 
    Surf {DIFFUSION_CONSTANT_2D = 1E-7} 
}

DEFINE_REACTIONS {
    /* creation of Vol2 */
    Vol1 -> Vol2 [1e4]

    /* annihilation of Vol2 */

    /* transport of Vol1 across surface
       * via Surf */
    Vol1, + Surf' -> Vol1' + Surf' [1e7]
}
```

- The general reaction syntax is
  `reactant(s) -> product(s) [rate] : name`
- Reactions can be named and then counted.

What is going on in the reaction involving Vol1 and Surf?
Now that we have defined molecules and placed them in the World we can define reactions between them.

The general reaction syntax is

\[ \text{reaction} \text{antisymmetric} \text{rate} \text{name} \]

Reactions can be named and then counted.

The units are

\[ \text{s}^{-1} \] for unimolecular reactions,
\[ \text{M}^{-1} \text{s}^{-1} \] for bimolecular reactions between two volume or a volume and a surface molecules

Make sure your simulation parameters are such that reaction probabilities remain < 1.
Now that we have defined molecules and placed them in the World we can define reactions between them.

The general reaction syntax is
\[
\text{reactant(s)} \rightarrow \text{product(s)} \ [\text{rate}] : \text{name}
\]
Reactions can be named and then counted.
The units are
\[
[\text{s}^{-1}] \text{ for unimolecular reactions,} \\
[\text{M}^{-1}\text{s}^{-1}] \text{ for bimolecular reactions between two volume or a volume and a surface molecules}
\]
Make sure your simulation parameters are such that reaction probabilities remain $< 1$.

What is going on in the reaction involving Vol1 and Surf?
How do we properly write reactions involving surface molecules in MCell?

Any reaction involving surface molecules requires orientation specifiers for each molecular player (volume and surface) involved.

The *relative* location of orientation specifiers determines the relative orientation molecules need to have in order for the reaction to proceed.
How do we properly write reactions involving surface molecules in MCell?

Since Vol1 and Surf have

- opposite specifiers on the reactant side, Vol1 reacts with the **bottom** of Surf.
- matching specifiers on the product side, Vol1 reacts with the **top** of Surf.

```
first_model.mdl

... 

DEFINE_REACTIONS

// creation of Vol2 */
Vol1 -> Vol2 [1e4]

// annihilation of Vol2 */

// transport of Vol1 across surface
// * via Surf */
Vol1, + Surf' -> Vol1' + Surf' [1e7]
```
How do we properly write reactions involving surface molecules in MCell?

Since Vol1 and Surf have
- opposite specifiers on the reactant side, Vol1 reacts with the **bottom** of Surf.
- matching specifiers on the product side, Vol1 reacts with the **top** of Surf.
How do we properly write reactions involving surface molecules in MCell?

Since Vol1 and Surf have

- opposite specifiers on the reactant side, Vol1 reacts with the **bottom** of Surf.
- matching specifiers on the product side, Vol1 reacts with the **top** of Surf.

We will learn more about surface reactions later!
Partitioning into spatial subvolumes is used by MCell to significantly speed up simulations via divide and conquer.
Partitioning into spatial subvolumes is used by MCell to significantly speed up simulations via *divide and conquer*.

Especially for larger models experimenting to find the proper partitioning scheme can lead to significant simulation speedup.
• Partitioning into spatial subvolumes is used by MCell to significantly speed up simulations via *divide and conquer*.
• Especially for larger models experimenting to find the proper partitioning scheme can lead to significant simulation speedup.
• Good partitioning requires hand tuning.

```plaintext
PARTITION_X = [[-1.0 TO 1.0 STEP 0.1]]
PARTITION_Y = [[-1.0 TO 1.0 STEP 0.1]]
PARTITION_Z = [[-1.0 TO 1.0 STEP 0.1]]
...
Partitioning into spatial subvolumes is used by MCell to significantly speed up simulations via *divide and conquer*.

Especially for larger models experimenting to find the proper partitioning scheme can lead to significant simulation speedup.

Good partitioning requires hand tuning.

There is a "sweet spot" between speed up due to an increased number of spatial subvolumes and excessive memory consumption.
MCell provides two types of output:

- **Visualization Output** for viewing your simulation in CellBlender.
- **Reaction Data Output** provides counts of molecules, events, reactions etc., in plain text ASCII format for further processing in your favourite tool.
Your First Model - Reaction Data Output

**first_model.mdl**

```plaintext
iters = 100
t = 1e-6
ITERATIONS = iters
TIME_STEP = dt

cubeVolume = 1e-15
Na = 6.02214129e23
...

INSTANTIATE World OBJECT {
  Cube OBJECT Cube {}
  ...
}

REACTION_DATA_OUTPUT {
  STEP = dt
  {{COUNT[Vol1,WORLD]]/cubeVolume/Na} => "./react_data/vol1_conc.dat"
  {COUNT[Vol2,WORLD]} => "./react_data/vol2.dat"
}
```

- Reaction data output is requested via a **REACTION_DATA_OUTPUT** block.
- **STEP** defines the interval at which to produce output.
- **COUNT** statements define what events to count and output. The general syntax is
  ```plaintext
  {COUNT[name, WORLD]} => "<filename>"  or  
  {COUNT[name, object]} => "<filename>"  or  
  {COUNT[name, region]} => "<filename>"
  ``
  where *name* is the name of a molecule or reaction.
- Output is typically in two column format
  ```plaintext
  ...
time1  count1
time2  count2
time3  count3
  ...
  ```
first_model.mdl

```plaintext
iters = 100
dt = 1e-6
ITERATIONS = iters
TIME_STEP = dt
...

VIZ_OUTPUT {
  VIZ_MOLECULE_FORMAT = ASCII
  FILENAME = "viz_data"
  MOLECULES {
    NAME_LIST {ALL_MOLECULES}
    ITERATION_NUMBERS {ALL_DATA @ ALL_ITERATIONS}
  }
}
```

- Visualization data output is requested via a VIZ_OUTPUT block.
- VIZ_MOLECULE_FORMAT defines the format of output the output and needs to be —ASCII— for compatibility with CellBlender.
- FILENAME defines the name of the master viz header file.
- NAME_LIST is a whitespace separated list of molecule names to output. ALL_MOLECULES outputs all molecules.
- ITERATION_NUMBERS defines what to output (POSITIONS, ORIENTATIONS or ALL_DATA) and when to output as a list of iterations (or ALL_ITERATIONS).

Visualization data can be read and visualized by CellBlender (formerly we used DReAMM for visualization of MCell data).