# Introduction to Microphysiological Simulations Using MCell

Markus Dittrich
dittrich@psc.edu

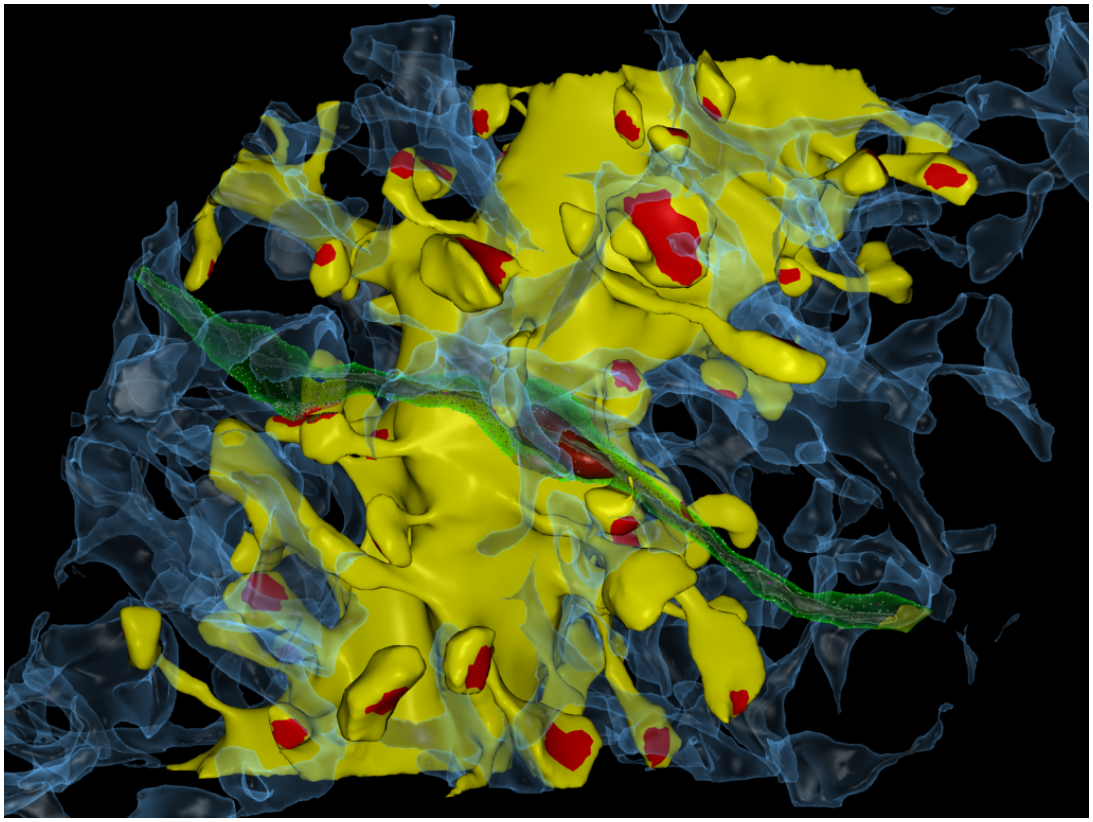
National Center for Multiscale Modeling of Biological Systems (MMBioS)
Biomedical Applications Group, Pittsburgh Supercomputing Center, CMU

April 27, 2015

# MCell, a Modeling Tool for Cellular Microphysiology

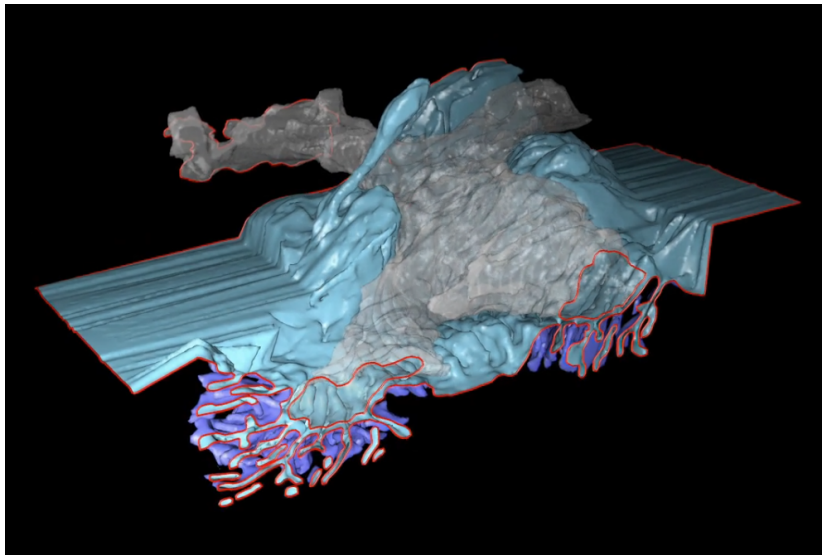
MCell is a Monte Carlo reaction-diffusion simulator for modeling cellular microphysiology in arbitrarily complex 3D spatial geometries.

Computational Microphysiology: The simulation of biological systems at micron to millimeter length scales (subcellular to cellular) using realistic 3D geometries over biological timescales from *ns* to *ms* to *s*.

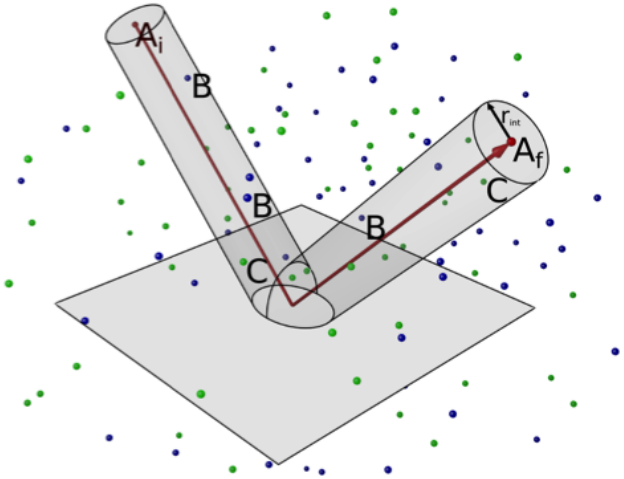


# Main Components of an MCell Model

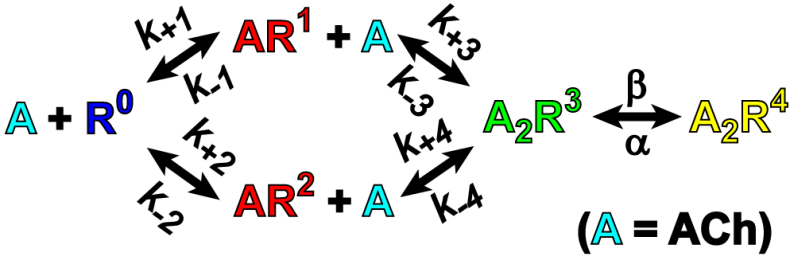
## 1. Realistic 3D Geometry

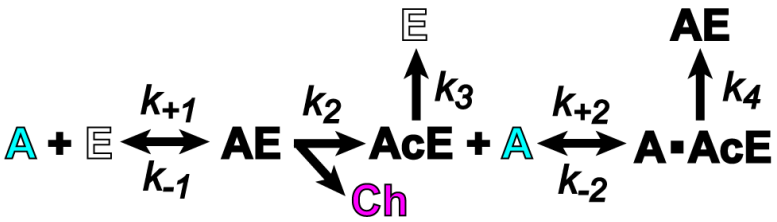


## 2. Random Walk Diffusion




## 3. Stochastic Biochemical State Transition



# Software Pipeline for Building MCell Models

Obtain or Create Model Mesh Geometry

↓

Annotate Meshes

↓

Specify Non-Spatial Model Parameters

↓

Simulate Model

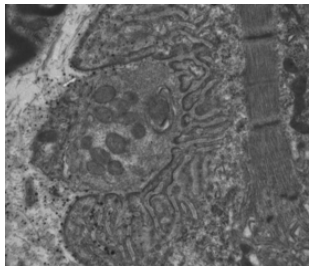
↓

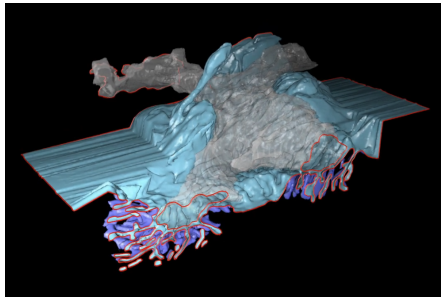
Visualize and Analyze Results

# Geometry Creation, Mesh Generation, Annotation

Model geometries for MCell simulations can be obtained via:

1. Reconstruction of model mesh geometry from electron microscopy data

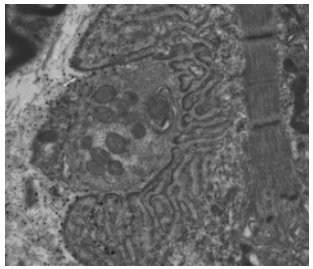


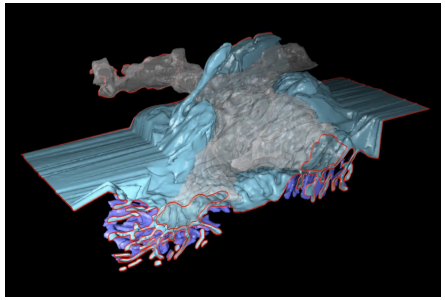


# Geometry Creation, Mesh Generation, Annotation

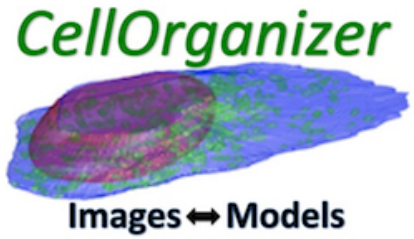
Model geometries for MCell simulations can be obtained via:

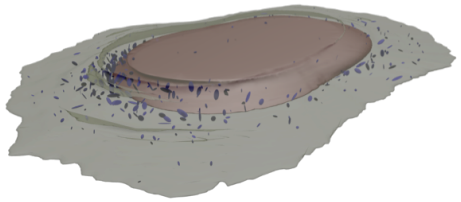
1. Reconstruction of model mesh geometry from electron microscopy data





2. Generative models of cell organization from fluorescence microscopy imaging (**Day 3**)

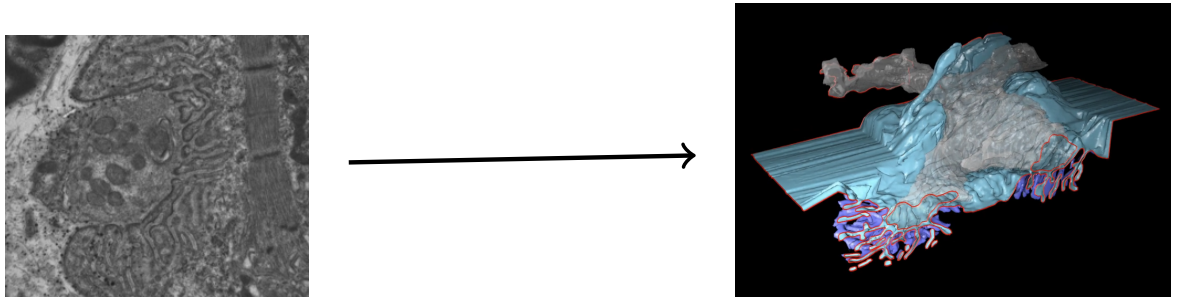


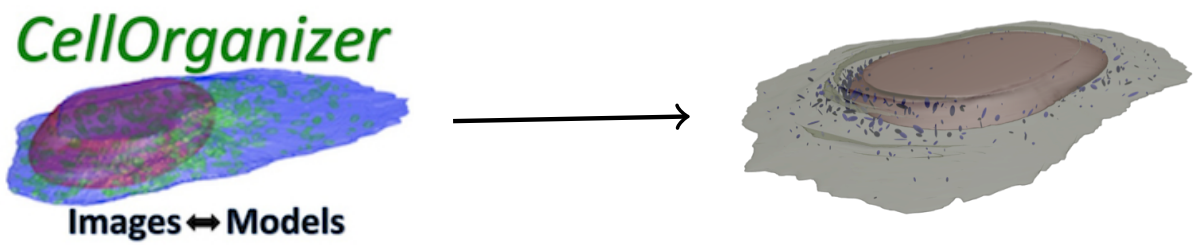
# Geometry Creation, Mesh Generation, Annotation

Model geometries for MCell simulations can be obtained via:

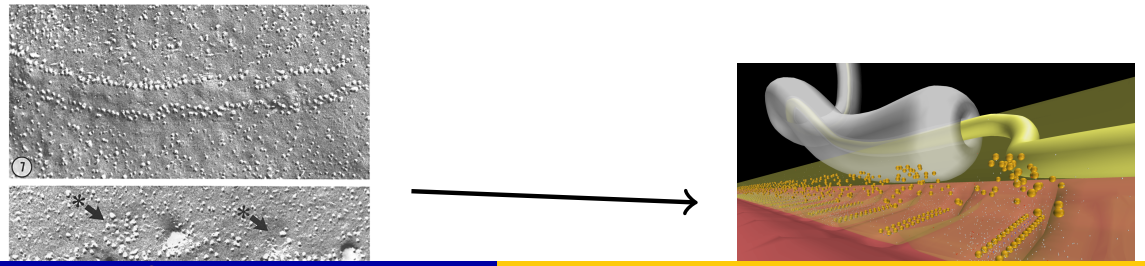
1. Reconstruction of model mesh geometry from electron microscopy data



2. Generative models of cell organization from fluorescence microscopy imaging (**Day 3**)

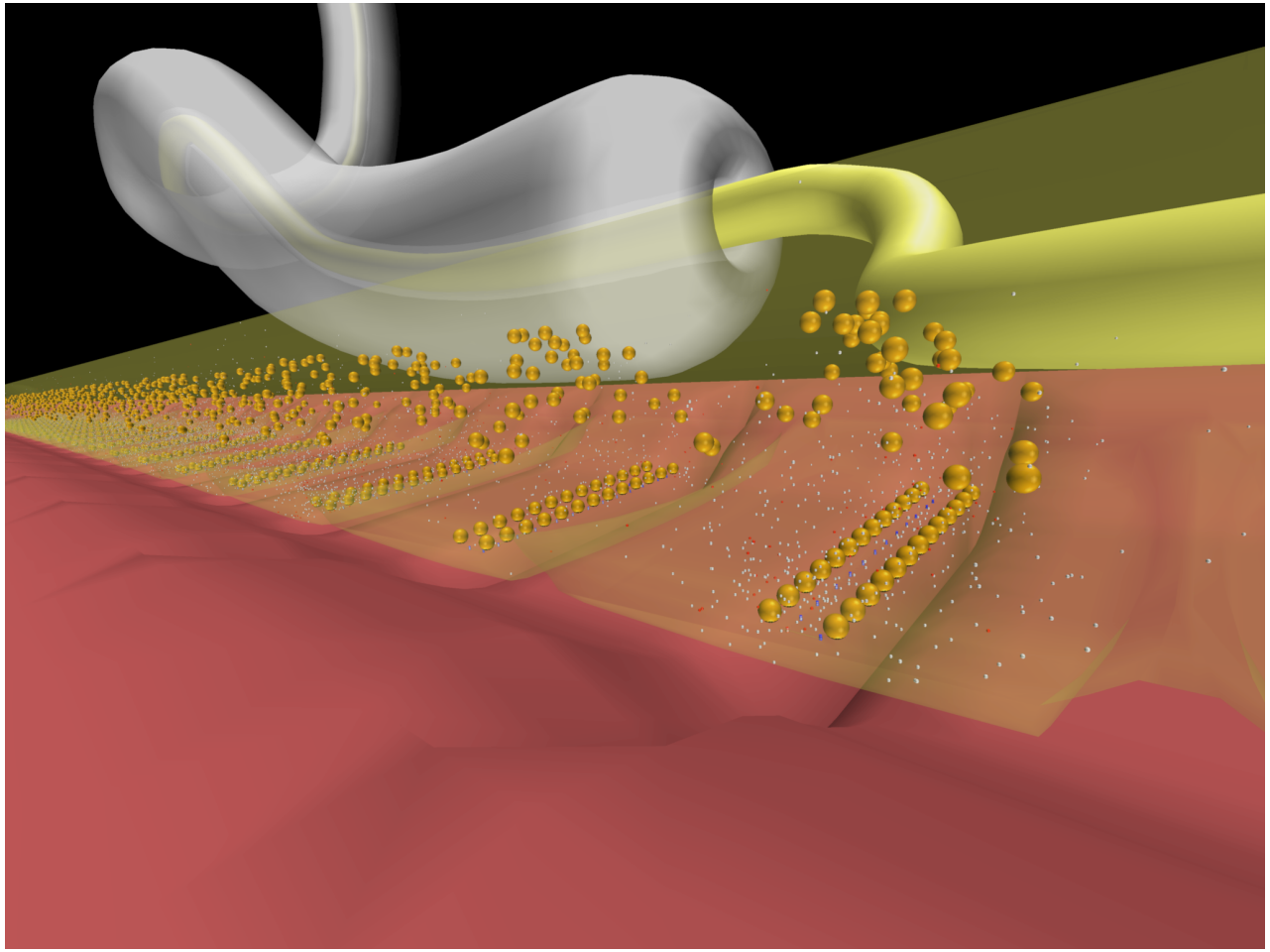



3. *In silico* geometry construction based on average geometry (**Day 1**)



# *In silico* Geometry Construction

Show 24 active zone model



# Software Pipeline to Build an MCell Model

Obtain or Create Model Mesh Geometry

↓

Annotate Meshes

↓

**Specify Non-Spatial Model Parameters**

↓

Simulate Model

↓

Visualize and Analyze Results

# Non-spatial Model Parameters

Beyond a geometry specification, MCell models typically also contain

- Molecules: can diffuse in space (volume molecules) or on mesh surfaces (surface molecules); requires knowledge of diffusion coefficients, concentrations/densities, locations.
- Reactions: requires knowledge of elementary reaction pathways (uni and bimolecular) and the corresponding reaction rate constants.
  Complex reaction networks can be described using rule based approaches via **BioNetGen** and then be imported into MCell models via CellBlender (**Day 2**).
- Output Specification: request output for visualization and analysis purposes; may require definition of additional geometry objects such as counting or sampling boxes.

# Software Pipeline to Build an MCell Model

Create Geometry

↓

Generate Meshes

↓

Annotate Meshes

↓

Specify Non-Spatial Model Parameters

↓

Simulate Model

↓

Visualize and Analyze Results

# Simulate Your Model

MCell simulations are run on the command line or from within CellBlender.

```
# mcell my_model.mdl
```

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

# Simulate Your Model

MCell simulations are run on the command line or from within CellBlender.

```
# mcell my_model.mdl
```

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

```
# mcell -seed 10 my_model.mdl
```

# Software Pipeline to Build an MCell Model

Create Geometry

↓

Generate Meshes

↓

Annotate Meshes

↓

Specify Non-Spatial Model Parameters

↓

Simulate Model

↓

Visualize and Analyze Results

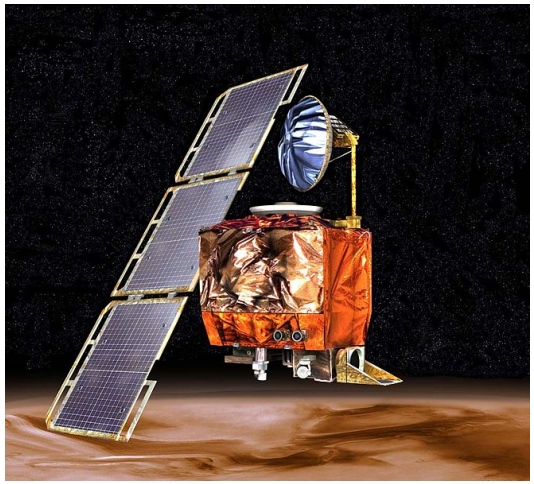
# Visualize and Analyse Results

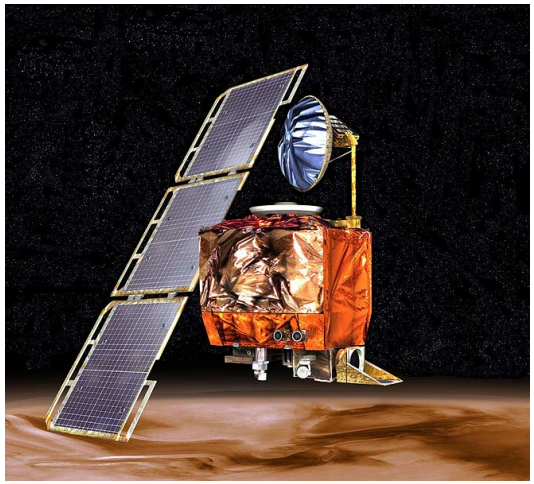
MCell provides two kinds of output (you specify how much of each, what molecules, etc.):

- Visualization Output for viewing via CellBlender, contains the location and orientation of surface and volume molecules.
- Reaction data output, contains time series of
  - volume molecule counts in specified regions of the model
  - surface molecule counts on specified regions of the model
  - reaction counts
  - hits or crossings of specific surface regions by surface and/or volume molecules
  - ...

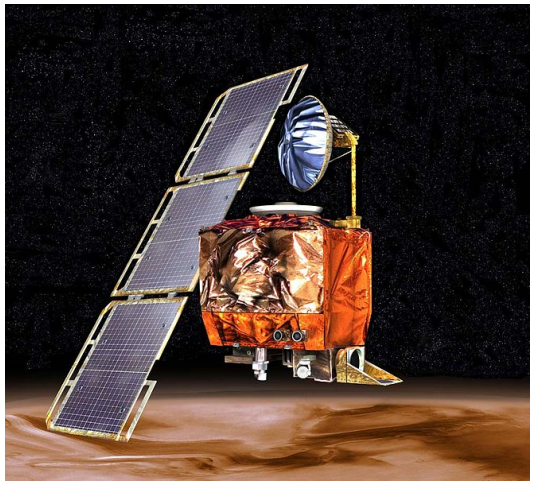
  Reaction data can be plotted with CellBlender and also be analyzed using external tools, e.g. `R`, `python`, `octave`, `MATLAB`, etc..

# MCell - Basic Definitions and Units



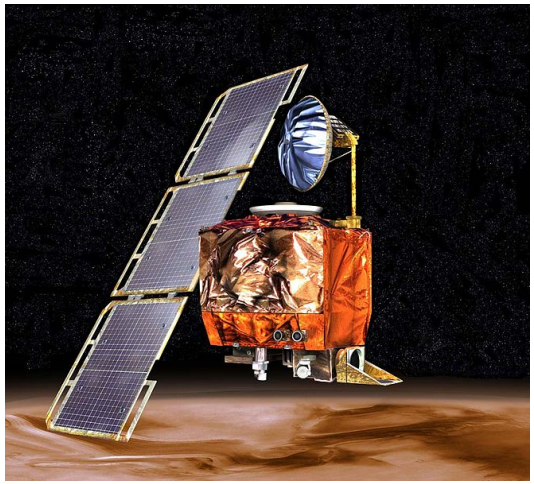


Mars Climate Orbiter, September 23, 1999



## 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!

# MCell - Basic Definitions and Units

- Spatial dimensions are in units of microns ($\mu m, 10^{-6} m$).
- Time is in units of seconds.
- Diffusion coefficients are in units of $cm^2 s^{-1}$.
- Uni-molecular reaction rate constants are in $s^{-1}$.
- Bi-molecular reaction rate constants between two volume or a volume and surface molecule are in $Mol^{-1} s^{-1}$.
- Bi-molecular reaction rate constants between two surface molecules are in $\mu m^2 \#^{-1} s^{-1}$.
- Each simulation runs for a specified number of iterations.
- The duration of an iteration is one timestep.

# Your First Model - MDL

MCell models are described using Model Description Language (MDL) (see
`http://www.mcell.org/documentation/qrg/index.html`
for a comprehensive list of keywords.)

MDL commands are written in ALL CAPS and consist of

- simple statements of the form

```
MDL_COMMAND = <value>
```

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

```
MDL_BLOCK { <block content> }
```

where `<block content>` are other `MDL_COMMAND`s or `MDL_BLOCK`s.

Each MCell simulation needs to define the simulation *timestep* (TIME_STEP) and the number of *iterations* (ITERATIONS).

**first_model.mdl**

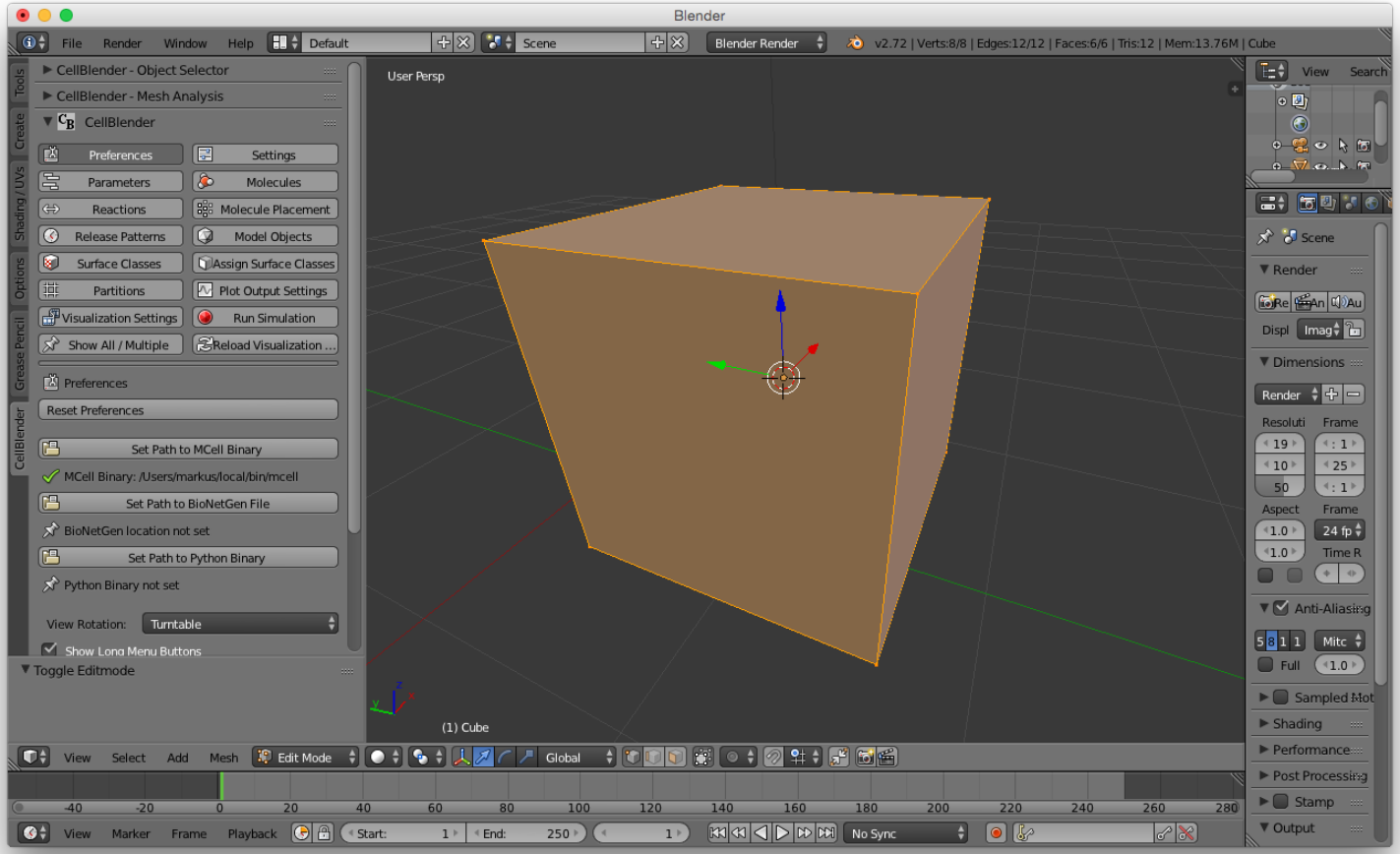
```
/* define variables for timestep
   and iterations */
iters = 100
dt    = 1e-6

/* 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 (`/* */` and single line `//`).

# Your First Model - Defining Geometry

Next, we use CellBlender to construct a mesh which defines the 3D geometry of our model. We create a simple cube shaped object and export it in MDL format, `Cube.mdl`.



Then we add `Cube.mdl` to our MDL file ....

**first_model.mdl**

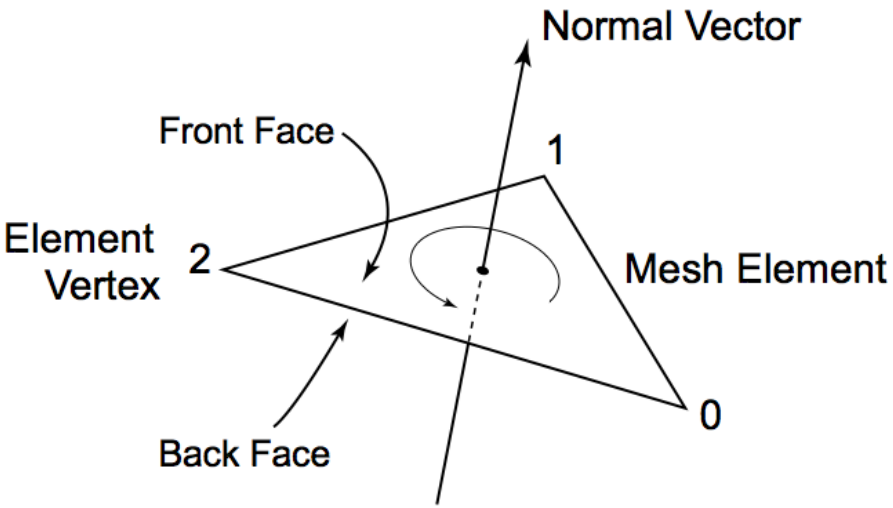
```
iters = 100
dt    = 1e-6
ITERATIONS = iters
TIME_STEP = dt

INCLUDE_FILE = "./Cube.mdl"
```

Notes:

- MDL files can include other MDL files which is very useful for organizing them, e.g., according to model component (geometry description, molecule definitions, etc.).
- Inclusion is lexical and thus the order of `INCLUDE` statements does matter.

## Cube.mdl

```
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 ]
    [ -1.00000035762787, -0.999999642372131, 1 ]
    [ -0.999999940395355, 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]
    }
  }
}
```



## Cube.mdl

```
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 ]
    [ -1.00000035762787, -0.999999642372131, 1 ]
    [ -0.999999940395355, 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 of specifying a geometrical shape within MDL.
- MCell also provides geometric primitives to define object geometry and molecule `RELEASE_SITE`s.
- `DEFINE_SURFACE_REGIONS` allows one 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*.

... and then make it part of the actual simulation model by adding it to the `INSTANTIATE`ed main object.

### first_model.mdl

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

INCLUDE_FILE = "./Cube.mdl"

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

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`).
- Within meta objects new geometry objects can be created, or exisiting objects can be copied and transformed (translation, rotation, scaling).
- Meta objects can be nested, i.e. contain other meta objects.

# Your First Model - Defining Molecules

Molecules are defined within a `DEFINE_MOLECULES` block. Our cube 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}
}

INCLUDE_FILE = "./Cube.mdl"

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

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

# Your First Model - Adding Molecules

Up to this point we only **defined** the molecules in our simulation but we haven't **released** any!

## 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.

# Your First Model - Adding Molecules

Up to this point we only **defined** the molecules in our simulation but we haven't **released** any!

## 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 needs to reference a **closed** geometry object (qualified starting from the instantiated top level object) in which to release the molecules.

# Your First Model - Adding Molecules

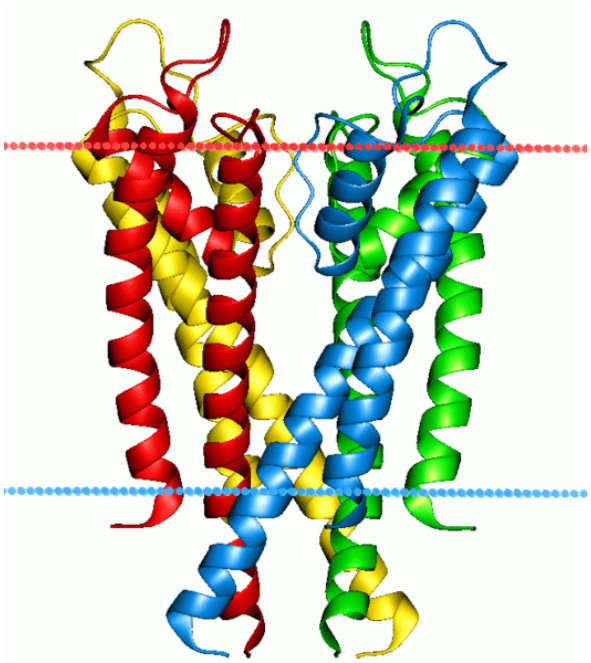
Up to this point we only **defined** the molecules in our simulation but we haven't **released** any!

## first_model.mdl

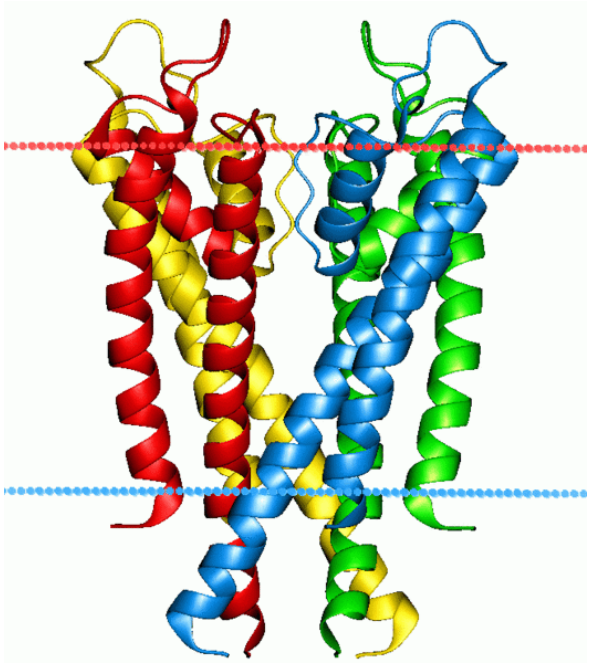
```
...

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 needs to reference a **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).

# Digression - Surface Molecules



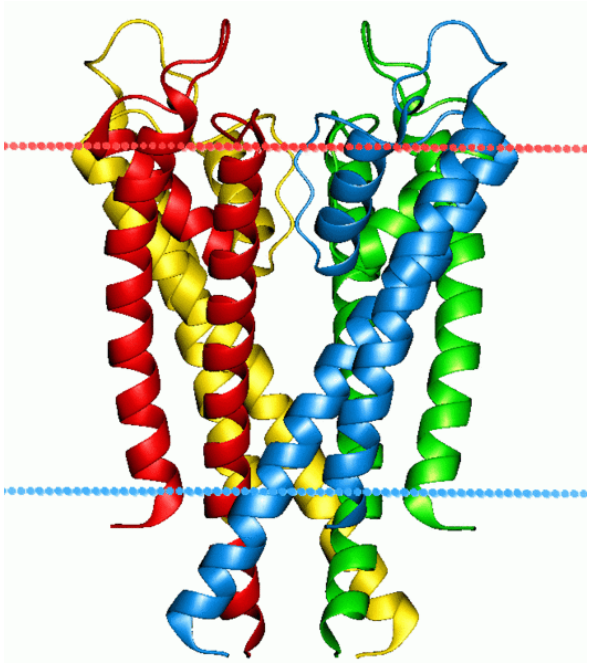
# Digression - Surface Molecules



KcsA (image from Wikipedia)

# Digression - Surface Molecules

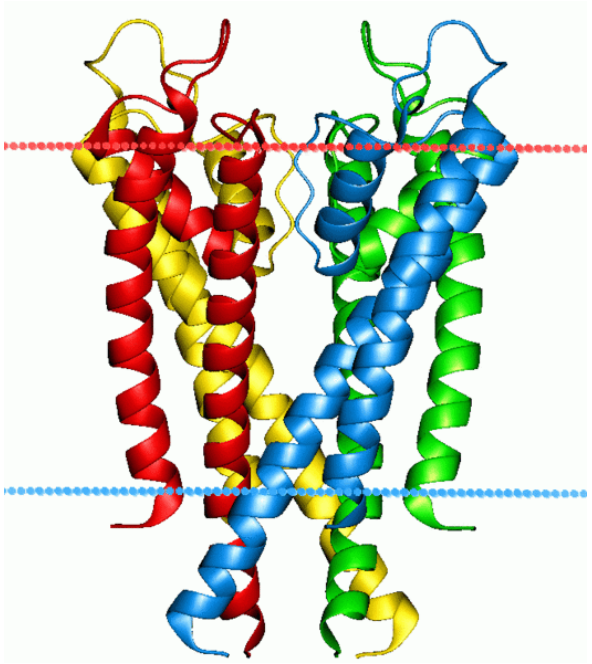
top



KcsA (image from Wikipedia)

# Digression - Surface Molecules

top

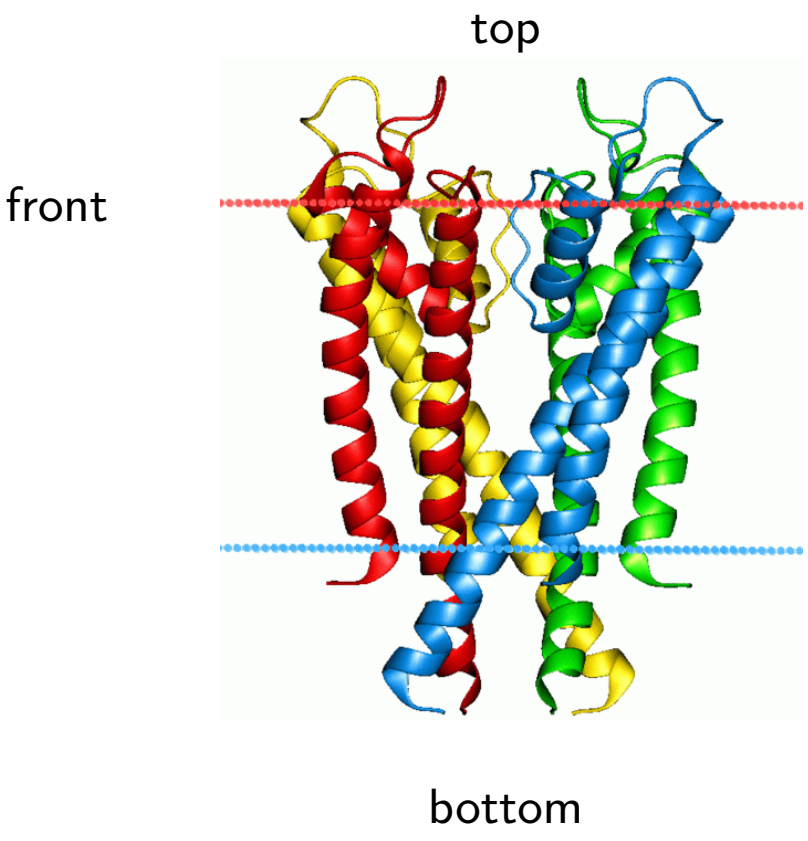


bottom

KcsA (image from Wikipedia)

- Surface molecules have a **top** and **bottom** domain.

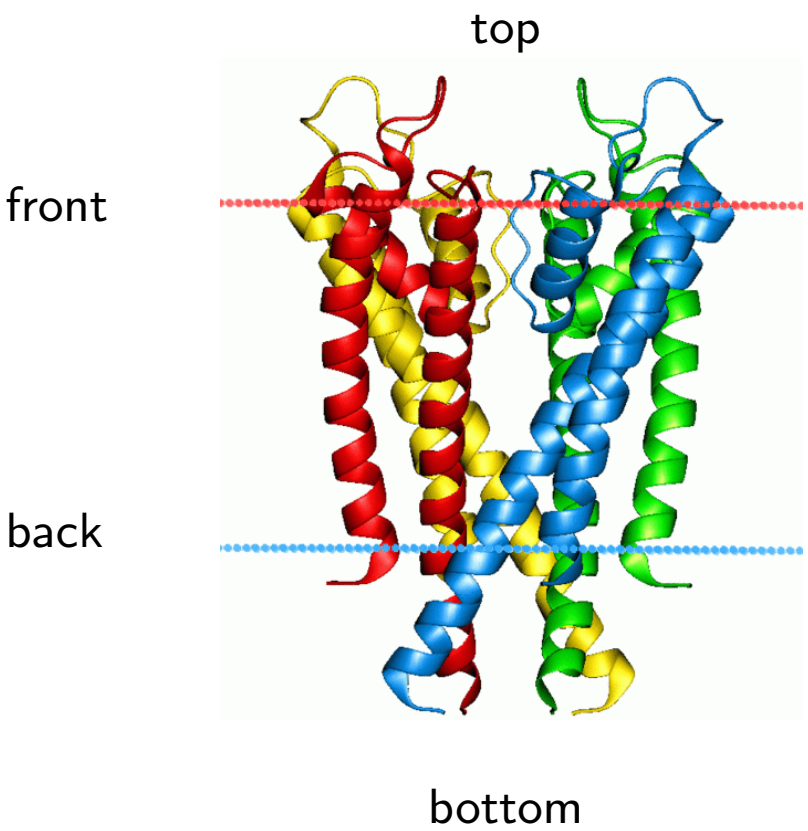
# Digression - Surface Molecules




KcsA (image from Wikipedia)

- Surface molecules have a **top** and **bottom** domain.

# Digression - Surface Molecules

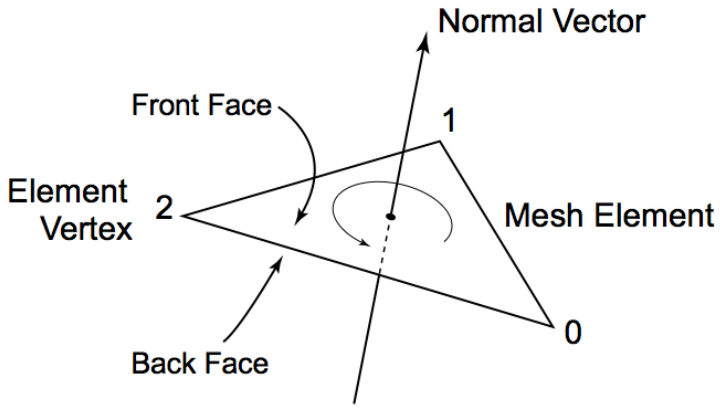


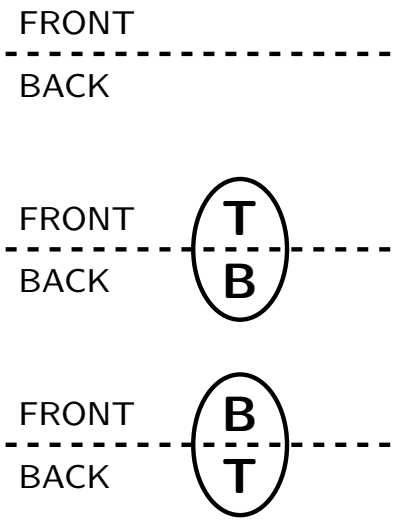

KcsA (image from Wikipedia)

- Surface molecules have a **top** and **bottom** domain.
- Surfaces (membranes) have a **front** and **back**.

# Digression - Surface Molecules

How does MCell handle surface molecule orientation?

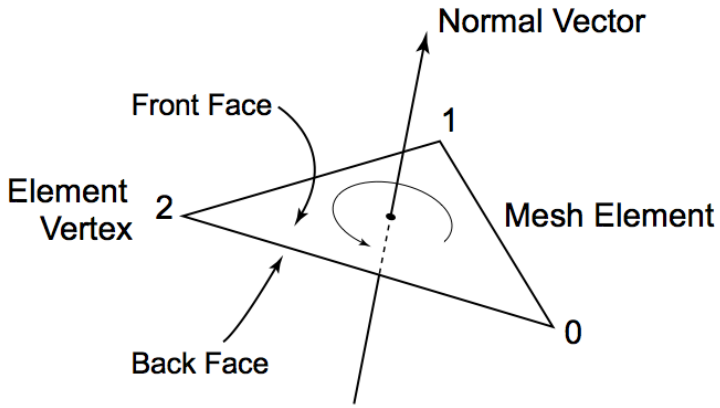


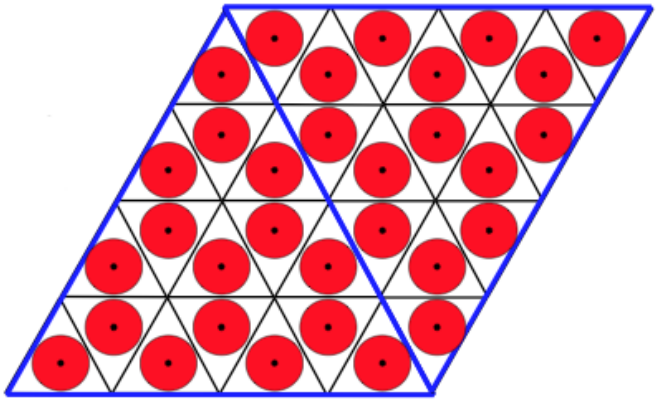




- mesh tiles and thus surfaces have a unique front and back according to the *right hand rule*.
  **Note:** Your mesh tiles need to have a consisted orientation!
- surface molecules are either placed with their top at the front or the back of the surface!
- each mesh element is subdivided into (triangular) 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 $\mu m^{-2}$).

# Digression - Surface Molecules

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 $\mu m^{-2}$).

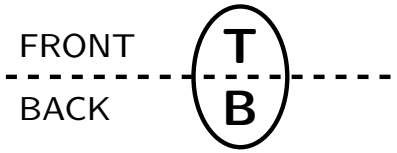
# Digression - Surface Molecules

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.



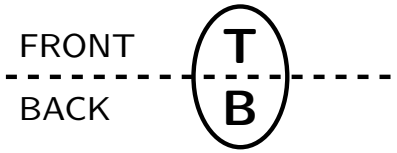
# Digression - Surface Molecules

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

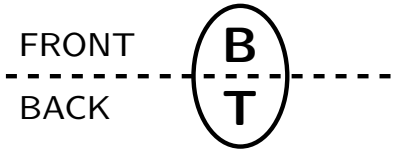
**first_model.mdl**

```
...

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.




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



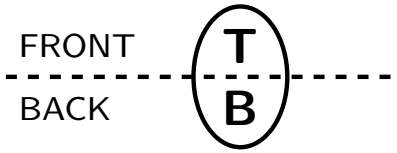
# Digression - Surface Molecules

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

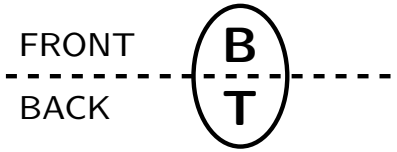
**first_model.mdl**

```
...

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

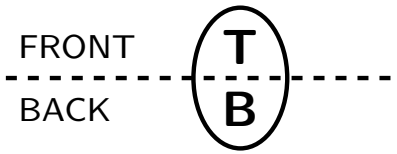
- `Surf'` places the molecule with its top at the FRONT of the surface.



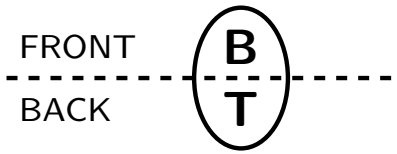

- `Surf,` places the molecule with its top at the BACK of the surface.




- `Surf;` places the molecule with its top randomly at the FRONT or BACK of the surface.




or



# Your First Model - Defining Reactions

Now that we have defined molecules and placed them in the `World` we can define reactions between them.

**first_model.mdl**

```
...

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 */
  Vol2 + Vol2 -> NULL [1e6] : annih

  /* 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.

# Your First Model - Defining Reactions

Now that we have defined molecules and placed them in the `World` we can define reactions between them.

**first_model.mdl**

```
...

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 */
  Vol2 + Vol2 -> NULL [1e6] : annih

  /* 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.
- The units are
  - $[s^{-1}]$ for unimolecular reactions,
  - $[M^{-1}s^{-1}]$ for bimolecular reactions between two volume or a volume and a surface molecule.
  - $[\mu m^2 \#^{-1} s^{-1}]$ for bimolecular reactions between two surface molecules.
- Make sure your simulation parameters are such that reaction probabilities remain $< 1$.

# Your First Model - Defining Reactions

Now that we have defined molecules and placed them in the `World` we can define reactions between them.

**first_model.mdl**

```
...

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 */
  Vol2 + Vol2 -> NULL [1e6] : annih

  /* 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.
- The units are
  - $[s^{-1}]$ for unimolecular reactions,
  - $[M^{-1}s^{-1}]$ for bimolecular reactions between two volume or a volume and a surface molecule.
  - $[\mu m^2 \#^{-1} s^{-1}]$ for bimolecular reactions between two 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?

**first_model.mdl**

```
...

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

  /* annihilation of Vol2 */
  Vol2 + Vol2 -> NULL [1e6] : annih

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

- 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.

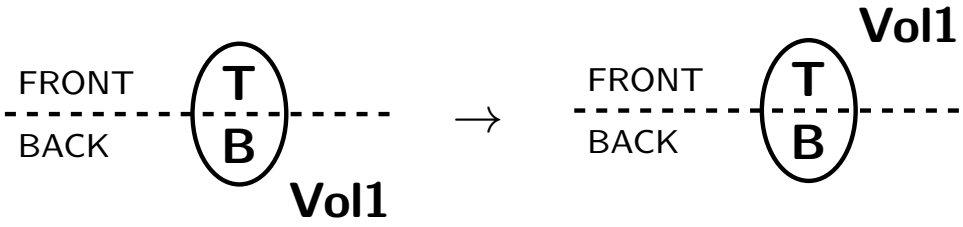
How do we properly write reactions involving surface molecules in MCell?

**first_model.mdl**

```
...

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

  /* annihilation of Vol2 */
  Vol2 + Vol2 -> NULL [1e6] : annihi

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

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`.



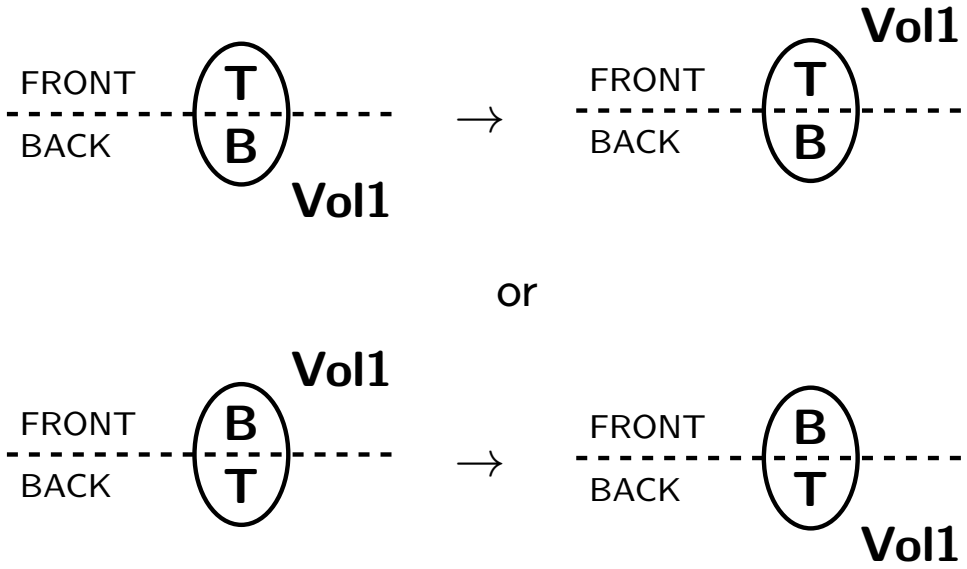
How do we properly write reactions involving surface molecules in MCell?

**first_model.mdl**

```
...

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

  /* annihilation of Vol2 */
  Vol2 + Vol2 -> NULL [1e6] : annihi

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

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`.



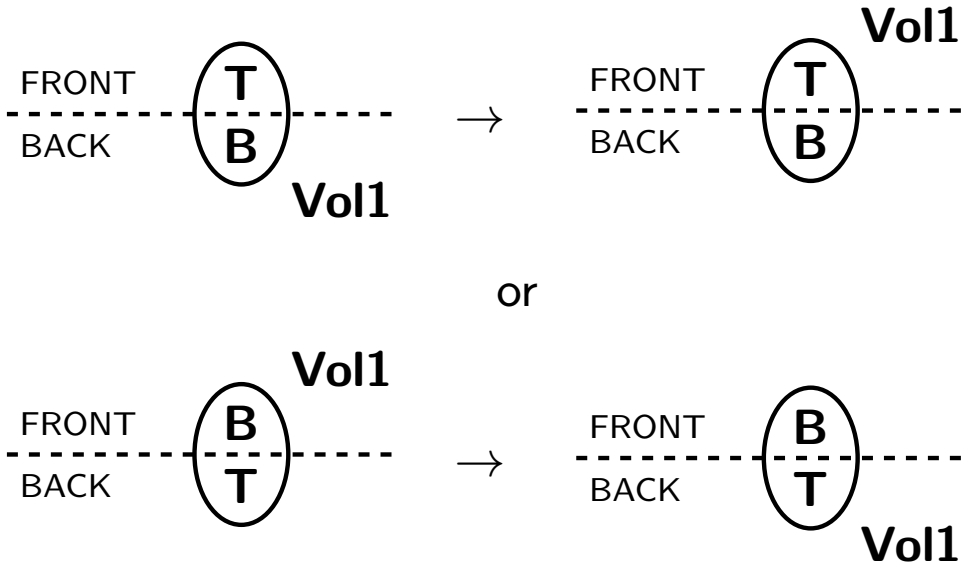
How do we properly write reactions involving surface molecules in MCell?

**first_model.mdl**

```
...

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

  /* annihilation of Vol2 */
  Vol2 + Vol2 -> NULL [1e6] : annihi

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

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`.

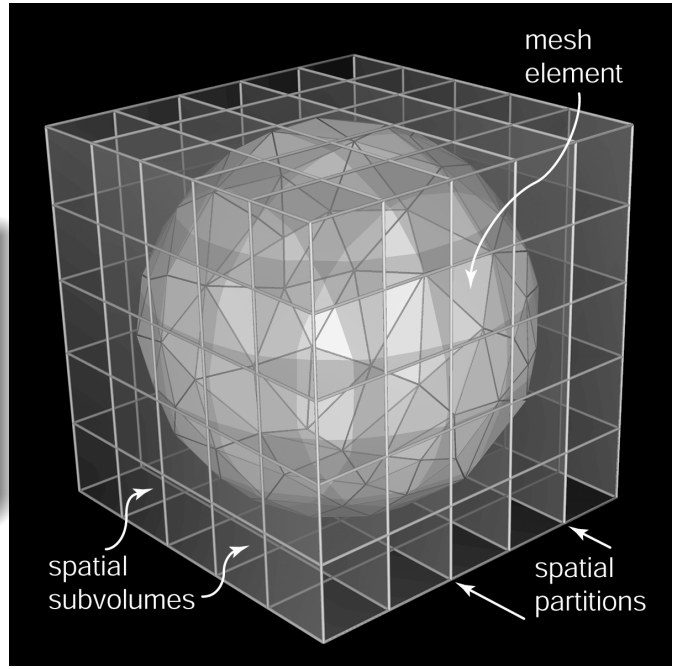



We will learn more about surface reactions later!

# Digression - Spatial Partitions For Efficiency

**first_model.mdl**

```
  ...
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]]
  ...
```

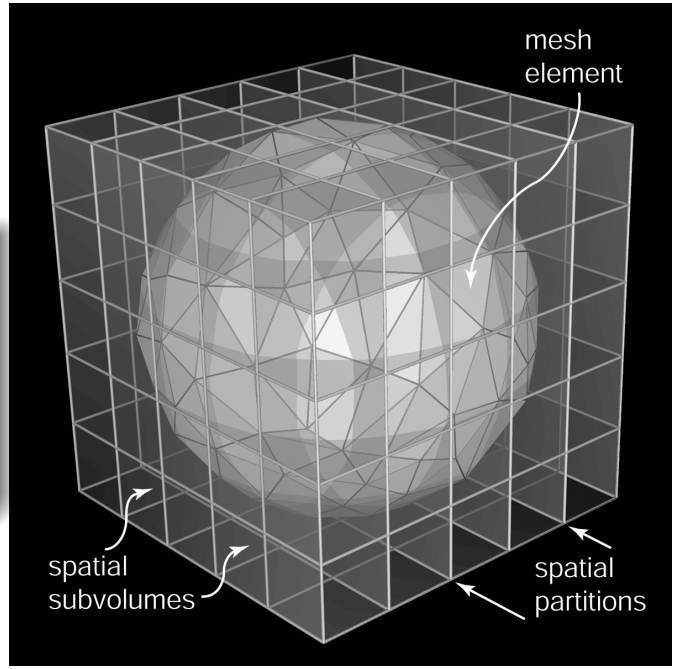



- Partitioning into spatial subvolumes is used by MCell to significantly speed up simulations via *divide and conquer*.

# Digression - Spatial Partitions For Efficiency

**first_model.mdl**

```
  ...
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]]
  ...
```

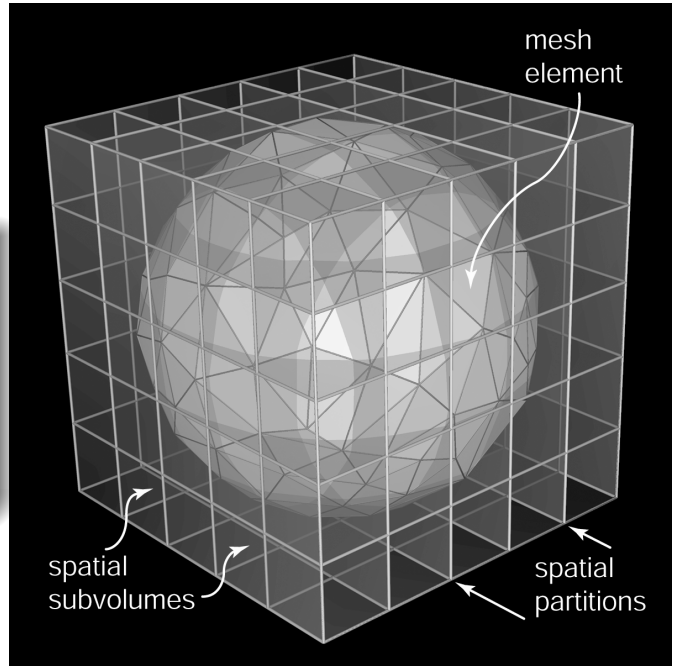



- Partitioning into spatial subvolumes is used by MCell to significantly speed up simulations via *divide and conquer*.
- Especially for larger models experimenting to find the proper partitioning scheme can lead to significant simulation speedup.

# Digression - Spatial Partitions For Efficiency

**first_model.mdl**

```
  ...
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]]
  ...
```

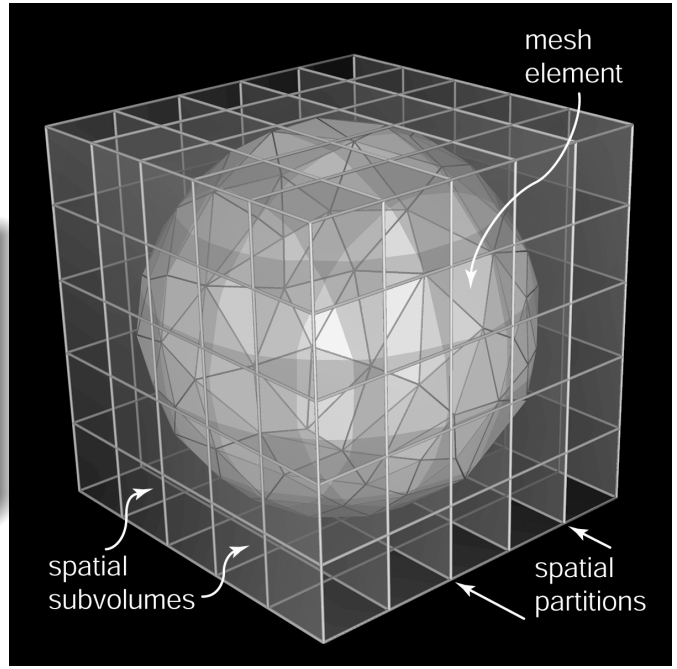



- Partitioning into spatial subvolumes is used by MCell to significantly speed up simulations via *divide and conquer*.
- Especially for larger models experimenting to find the proper partitioning scheme can lead to significant simulation speedup.
- Good partitioning requires hand tuning.

# Digression - Spatial Partitions For Efficiency

**first_model.mdl**

```
  ...
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.

# Your First Model - Generate Output

MCell provides to 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

```
iters = 100
dt    = 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

  ```
  {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

  ```
  ...
  time1   count1
  time2   count2
  time3   count3
  ...
  ```

# Your First Model - Visualization Data Output

**first_model.mdl**

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


VIZ_OUTPUT {
  VIZ_MOLECULE_FORMAT = CELLBLENDER
  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 should be `CELLBLENDER` 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.