MCell3 Quick Tutorial and Reference Guide

Rex Kerr

29th September 2005

In this document, the main text is in a serif font. Command-line entries, mdl file commands, and code is in a `fixed-width` font. Values that must be supplied by the user are in an *italicized sans-serif* font.

This document isn't finished. It's barely even started. But, here it is. This document was created using LyX 1.3.4.

1 Running MCell3

MCell3 runs on the command line. The format is

`mcell3` *options filename*

By default, MCell3 sends informational messages, such as simulation progress, to stdout (which will normally appear on the screen); error messages are sent to stderr (which will also normally appear on the screen). Results of simulations are written to files and do not appear as MCell3 is running.

A brief summary of MCell3 optional command-line arguments is given below.

Argument	**Explanation**
`-seed` *N*	Start with random number seed *N* instead of 1 (the default).
`-iterations` *N*	Run the simulation with *N* timesteps (overrides the value in the mdl file)
`-help`	Print out a basic help screen.
`-info`	Same as `-help`
`-logfile` *filename*	Send messages to *filename* instead of stdout/stderr
`-logfreq` *N*	Valid syntax, but not implemented. Does nothing.
`-checkpoint_infile` *filename*	Use *filename* as a checkpoint file for the current simulation (overrides any value in the mdl file).

2 Model Description Language overview

MCell3 runs simulations that are specified in *model description language* (mdl) format. These files typically have the extension .mdl, but are not required to. A mdl file is a text file with commands separated by whitespace. The nature and type of whitespace (space, tab, newline) is unimportant to MCell3. You are thus free to use whitespace to clarify the contents of the mdl file.

Commands fall into five general groups, which usually should be given in the order presented below. Although this is not always required, there are some commands (e.g. defining a molecule) that must be used before others (e.g. defining a reaction that uses that molecule). The order below should always be safe:

1. Initialization. These commands set global parameters such as the timestep, spatial partitioning, and duration of

the simulation.

2. Molecule definitions. These commands specify the names and diffusion constants of molecules in the simulation.

3. Reaction definitions. These commands specify the reactions that can occur between molecules and the rate at which those reactions occur.

4. Geometry specification. These commands describe the membranes and other boundaries within which the simulation occurs, plus where in the world to place molecules initially.

5. Output specification. These commands specify what data should be output as the simulation is running; this can include graphical snapshots of the simulation in progress, as well as lists of numbers of molecules or reactions as a function of time.

In addition, there are utility commands–defining variables and including other mdl files–that can appear nearly anywhere.

3 MDL commands

3.1 Initialization commands

The following initialization commands are required in every mdl file.

Command	Explanation
`TIME_STEP` = *t*	Set the simulation time step to *t* seconds. `1e-6` is a common value.
`ITERATIONS` = *N*	Run the simulation for *N* iterations.

The following initialization commands are optional.

Command	Explanation
`TIME_STEP_MAX` = *N*	MCell3 will move longer than the specified simulation time step if it seems safe. This command makes sure that the longest possible time step is no longer than *N* seconds, even if MCell3 thinks a longer step would be safe. The default is ???
`SPACE_STEP` = *N*	Have all diffusing molecules take time steps of different duration, chosen so that the mean diffusion distance is *N* microns for each molecule. By default, all molecules move the same time step.
`EFFECTOR_GRID_DENSITY` = *N*	Tile all surfaces so that they can hold molecules at *N* different positions per square micron. The default is 10000.
`INTERACTION_RADIUS` = *N*	Diffusing molecules will interact with each other when they get within *N* microns of each other. The default is 0.01.
`PARTITION_`*D* = [*list*]	*D* must be `X`, `Y`, or `Z`. Subdivide that axis of space at the boundaries given in *list* (in microns). In future versions, MCell3 will further subdivide space if it is computationally advantageous. By default, each axis will be split into between five and fifteen equal partitions. If you do not explicitly partition all three axes, MCell3 is likely to ignore your request and perform automatic partitioning.
`RADIAL_DIRECTIONS` = *N*	Specifies how many different directions to put in the lookup table. The default is sensible. Don't use this unless you know what you're doing. Instead of a number, you can specify `FULLY_RANDOM` to generate the directions directly from double precision numbers (but this is slower).

Command	Explanation
RADIAL_SUBDIVISIONS = *N*	Specifies how many distances to put in the diffusion lookup table. Again, the default is sensible. FULLY_RANDOM is not implemented here.
ACCURATE_3D_REACTIONS = *boolean*	Specifies which method to use for computing 3D molecule-molecule interactions. If *boolean* is TRUE, then molecules will look through partition boundaries for potential interacting partners–this is slower but more accurate. If *boolean* is FALSE, then molecule interaction disks will be clipped at partition boundaries and probabilities adjusted to get the correct rate–this is faster but can be less accurate. The default is TRUE.
CENTER_MOLECULES_ON_GRID = *boolean*	If *boolean* is set to TRUE, then all molecules on a surface will be located exactly at the center of their grid element. If FALSE, the molecules will be randomly located when placed, and reactions will take place at the location of the target (or the site of impact in the case of 3D molecule/surface reactions). The default is currently TRUE, but this should probably be changed.
VACANCY_SEARCH_DISTANCE = *r*	Normally, a reaction will not proceed on a surface unless there is room to place all products on the single grid element where the reaction is initiated. By increasing *r* from its default value of 0, one can specify how far from the reaction's location, in microns, the reaction can place its products. To be useful, *r* must be larger than the longest axis of the grid element on the triangle in question. The reaction will then proceed if there is room to place its products within a radius *r*, and will place those products as close as possible to the place where the reaction occurs (deterministically, so small-scale directional bias is possible).

3.2 Molecule definition commands

All molecules must be defined by name in a DEFINE_MOLECULES block. For users of MCell 2, note that there is no longer a distinction between a receptor and a ligand. Everything is a molecule, and every different bound state of a receptor must have a unique name (since it must be a unique molecule). The names must be unique in the entire simulation (not just the MDL file).

A define molecule block can be one of the following:

Command	Explanation
DEFINE_MOLECULE *name* {...}	Define a single molecule called *name*. The molecule's properties are specified by commands inside braces.
DEFINE_MOLECULES {...}	Define a series of molecules by name, using the following command.
name {...}	Within a DEFINE_MOLECULES block, define a new molecule called *name*. The molecule's properties are specified by commands inside braces.

Each molecule must have a diffusion constant set using one of the following commands:

Command	Explanation
DIFFUSION_CONSTANT = *D*	This molecule diffuses in space with diffusion constant *D*. *D* can be zero, in which case the molecule doesn't move. Synonyms for this command are DIFFUSION_CONSTANT_3D and D_3D. The units of *D* are cm^2/s.
DIFFUSION_CONSTANT_2D = *D*	This molecule is constrained to a surface and diffuses with diffusion constant *D*. Until surface diffusion is implemented, only 0 is a valid value for *D*. D_2D is a synonym for this command.

The following optional commands can be applied to each molecule (and must appear in this order, and after the

diffusion constant is set):

Command	Explanation
`CUSTOM_TIME_STEP` = *t*	This molecule should take timesteps of length *t* (in seconds). Use either this or `CUSTOM_SPACE_STEP`, not both.
`CUSTOM_SPACE_STEP` = *L*	This molecule should take steps of average length *L* (in microns). If you use this directive, do not set `CUSTOM_TIME_STEP`.
`TARGET_ONLY`	This molecule will not initiate reactions when it runs into other molecules. This setting can speed up simulations when applied to a molecule at high concentrations that reacts with a molecule at low concentrations (it is more efficient for the low-concentration molecule to trigger the reactions). This directive does not affect unimolecular reactions.

3.3 Reaction definition commands

All reactions must be defined inside a reaction definition block:

Command	Explanation
`DEFINE_REACTIONS {...}`	Define a series of reactions inside braces.

Reactions are specified using arrow notation:

Command	Explanation
reactants `->` *products* `[`*rate*`]`	Define a reaction that occurs between one or two *reactants* and produces an arbitrary number of *products*, with a specified *rate*. If a molecule is in the *reactants* list and not in the *products* list, it is destroyed in the reaction. One reactant may be a surface type (see section 3.4.1).
reactants `->` *products* `[`*rate*`]:`*name*	As above, and call the reaction *name* so it can be referred to later.

This notation is perhaps best explained through examples. In the most basic form, reactants and products are just the names of molecules, separated by `+`:

Example	Explanation
`A -> B [100]`	Molecule `A` changes into molecule `B` at a rate of $100\,\mathrm{s}^{-1}$.
`A -> A + B [100]`	Molecule `A` emits molecules of `B` at a rate of $100\,\mathrm{s}^{-1}$.
`A + B -> A [1e6]`	Molecule `A` destroys molecule `B` at a rate of $10^6\,\mathrm{M}^{-1}\cdot\mathrm{s}^{-1}$.
`A + B -> A + C [1e6]`	Molecule `A` catalytically converts `B` to `C` at a rate of $10^6\,\mathrm{M}^{-1}\cdot\mathrm{s}^{-1}$.
`A+B -> A+B+C [1e6]`	Collision of `A` and `B` catalytically generates `C` at a rate of $10^6\,\mathrm{M}^{-1}\cdot\mathrm{s}^{-1}$.

Orientation classes are a fundamentally new concept introduced in MCell3. They replace the MCell 2 idea of `POSITIVE_POLE`, `NEGATIVE_POLE`, and `BOTH_POLES` specifications for receptors.

In MCell3, molecules in a surface have an orientation. If a reaction is geometrically constrained–for example, if a ligand can only bind to the extracellular side of a receptor–you can enforce this geometric constraint in MCell3 by writing down the relative orientations of the reactants and products. The two orientations are specified by `'` and `,` (comma and apostraphe) after the molecule's name. For example, a surface-bound molecule `B` has the orientations `B'` and `B,`. You can think of these as the top and bottom of `B`. For example:

Example	Explanation
`B' -> B, [10]`	Molecule `B` flips (changes its orientation) at a rate of $10s^{-1}$.
`B' -> B' + A' + C, [10]`	Molecule `B` emits molecules of `A` on one side and C on the other at a rate of $10s^{-1}$

In the second example, `A` comes off of the top of `B` (e.g. on the extracellular side) while `C` comes off on the bottom (e.g. on the intracellular side).

Molecules diffusing in 3D do not have an orientation, but they adopt one when they strike a surface. The side that they strike is their "top" side. The far side is the "bottom" side. (In MCell 2 terminology, the side they hit is their positive pole, and the other side is their negative pole.) Thus, if `A` diffuses in 3D but `B` and `C` are on a surface:

Example	Explanation
`A' + B' -> C' [1e5]`	Molecule `A` binds to the top side of `B`, producing a `C` facing the same way as `B`, at a rate of $10^5\,M^{-1}\cdot s^{-1}$.
`A' + B, -> C' [1e5]`	Molecule `A` binds to the bottom side of `B`, producing a `C` facing the opposite way as `B`, at a rate of $10^5\,M^{-1}\cdot s^{-1}$.

So far, all examples have used the first orientation class, specified with `'` and `,`. By default, if no orientation is specified, the molecule is in the zeroth class, and the reaction refers to the top of the molecule. The second orientation class is specified by `''` and `,,`. The third is `'''` and `,,,` and so on. A molecule can be in only one orientation class, and molecules in different classes do not pay attention to each others' orientation. Therefore:

Example	Explanation
`A + B, -> C' [1e5]`	Molecule `A` binds to either side of `B`, producing a `C` facing the opposite way as `B`, at a rate of $10^5\,M^{-1}\cdot s^{-1}$.
`A + B -> C [1e5]`	Molecule `A` binds to the top side of `B`, producing a `C` facing the same way as `B`, at a rate of $10^5\,M^{-1}\cdot s^{-1}$. (Same as `A'+B'->C'`)
`A, + B' -> C [1e5]`	Molecule `A` hits the other side from the top of `B` and produces `C` that is equally likely to point either way, at a rate of $10^5\,M^{-1}\cdot s^{-1}$. Since hitting the "other side from the top" is the same as "hitting the bottom", this is the same as `A'+B,->C`
`A, + B' -> C'' [1e5]`	Same as above, since `C` is still not in the same orientation class as the others.
`A'+B'' -> A,+B''' [1e5]`	Molecule `A` hits molecule `B` on either side, causing `A` to go to the other side and `B` to tumble to a random orientation, at a rate of $10^5\,M^{-1}\cdot s^{-1}$. (`A'+B->A,+B''` would do the same thing, as would `A,+B->A'+B,`, and so on.)

There are examples of how one would use this syntax to model well-known biological reactions at the end of this file.

3.4 Geometry definition commands

3.4.1 Surface properties

MCell3 allows the user to specify properties of the surfaces of objects. For example, one may wish to specify that a surface does not block the diffusion of molecules. Each type of surface is defined by name, and each surface name must be unique in the simulation and should not match any molecule names. Surface properties are specified inside a surface definition block:

Command	Explanation

Command	Explanation
DEFINE_SURFACE_CLASS *name* {...}	Define a single surface type called *name*. The properties are specified by commands inside braces.
DEFINE_SURFACE_CLASSES {...}	Define a series of surface types by name, using the following command.
name {...}	Within a DEFINE_SURFACES block, define a new surface type called *name*. The properties are specified by commands inside braces.

To define surface properties, use the following commands:

Command	Explanation
REFLECTIVE = *name*	The molecule called *name* is reflected by this surface. This is the default behavior.
TRANSPARENT = *name*	The molecule called *name* passes through this surface.
ABSORPTIVE = *name*	The molecule called *name* is destroyed if it touches this surface.
MOLECULE_DENSITY {...}	Add molecules at the specified density (units = μm^{-2}) using the format *name=density* inside the braces.
MOLECULE_NUMBER {...}	Add this exact number of molecules onto any region that is made out of this surface class, using the format *name=number* inside the braces. Note: this usage is not recommended; it is better to add exact numbers of molecules to the region.

3.4.2 Geometrical objects

Two types of geometrical objects are supported in MCell3. Objects should not have coincident surfaces unless neither surface contains a molecule that any moving molecule can react with. Also, all coincident surfaces should agree on whether they are transparent, reflective, or absorptive to each molecule that might strike them. Geometrical objects can be defined using:

Command	Explanation
name BOX { ... }	This defines a box object called *name*. The position of the object is defined inside braces. Internally, a box is just stored as a set of triangular surface elements.
CORNERS = [*x1,y1,z1*],[*x2,y2,z2*]	The box object has corners as specified. The first coordinates should be less than the second set of coordinates, although MCell3 may fix it if you do it incorrectly.
ASPECT_RATIO = *a*	Make sure that the ratio of the long to short side of each triangle making up the box is no more than *a*. The smallest allowed value is 2. The default is to not care about triangle shape.
name POLYGON_LIST { ... }	This defines a polygon list object called *name*. Polygon list objects explicitly give their triangular surface elements.
VERTEX_LIST { ... }	Specify the vertices of the triangles inside a polygon list object inside braces. Each vertex is given by the triple [*x,y,z*], and vertices are listed one after another without commas.
ELEMENT_CONNECTIONS { ... }	Specify the triangles by vertex indices. The vertices are numbered from 0 upwards in the order they were given in the vertex list. The direction of the surface normal is determined by the right-hand rule while following the vertices. Each triangle is given by a triple [*i,j,k*], and the triangles are listed one after another without commas.

A variety of optional commands can be used inside a geometrical object definition block, after corners or vertex list / element connections are specified, to modify the basic composition of the object and its surface properties. These are

described below.

Command	Explanation
REMOVE_ELEMENTS { ... }	Remove the portion of the object specified inside braces using element specifiers described below. You can think of this as a special type of region that defines the removed portions of the object. No real region exists on any part of the object that has been removed. You can use a list of element numbers/names instead of the full element specifiers if you wish, but you cannot mix element numbers/names with the element specifier syntax defined below.
DEFINE_SURFACE_REGIONS { ... }	Define regions on the object. You can have an arbitrary number of regions on an object, and they may overlap if you wish. Molecules added to overlapping regions accumulate; surface properties are those of the last region applied.
name { ... }	Inside a define region block, this specifies a new region called *name*. Inside braces, you first specify the elements to be included using the element specifiers below and then list any special properties of that region, such as surface-bound molecules in the region.

Element specifiers consist of one or more of the following, which are applied in order:

Command	Explanation
INCLUDE_ELEMENTS = [*list*]	Include the elements specified by number or name. For polygon objects, these refer to the triangles defined by the element connections, counting from zero upwards in the order given. For rectangles, the side names LEFT, RIGHT, FRONT, BACK, BOTTOM, and TOP can be used to refer to the sides, where left/right corresponds to the x axis (left is lower x values), front/back to y, and bottom/top to z. ALL_ELEMENTS refers to the entire object. Numbers can be specified individually (separated by commas) or in ranges with the format *N* TO *M*. The two styles can be mixed (separated by commas).
EXCLUDE_ELEMENTS = [*list*]	Exclude the elements listed. If this is the first element specifier, assume that all elements not listed are included. If not, subtract from the existing list.
INCLUDE_REGION = *name*	Include the existing region on this object called *name* into this region, too.
EXCLUDE_REGION = *name*	Exclude the existing region on this object called *name* from this new region.
INCLUDE_PATCH=[*x1,y1,z1*],[*x2,y2,z2*]	This specifier is only valid on box objects, and the corners must define a rectangular patch that is on exactly one side of the box. The box will be divided into triangles in such a way that this patch consists of separate triangles and will form a region.
EXCLUDE_PATCH=[*x1,y1,z1*],[*x2,y2,z2*]	Exclude the patch from this region.

After element specifiers, regions can specify a surface type and add extra molecules using

Command	Explanation
SURFACE_CLASS = *name*	Set the surface type of this region to the previously defined surface class called *name*.
MOLECULE_DENSITY {...}	Add molecules at the specified density (units = μm^{-2}) using the format *name*=*density* inside the braces.

Command	Explanation
MOLECULE_NUMBER {...}	Add this exact number of molecules onto any region that is made out of this surface class, using the format *name=number* inside the braces. Note: this usage is not recommended; it is better to add exact numbers of molecules to the region.

3.4.3 Release objects

Release objects place molecules into three-dimensional space. These molecules must be defined with a 3D diffusion constant. Release objects are defined using the following commands:

Command	Explanation
name RELEASE_SITE {...}	Create a release site called *name*. The shape and method of release is specified later.
name CUBIC_RELEASE_SITE {...}	Create a cubic release site called *name*. Molecules are released in a box as specified by the radius. (This is the same as using the SHAPE=CUBIC command inside RELEASE_SITE.)
name SPHERICAL_RELEASE_SITE {...}	Create a spherical release site called *name*. Molecules are released uniformly within the sphere depending on the defined radius of the object. (This is the same as using the SHAPE=SPHERICAL command inside RELEASE_SITE.)
name SPHERICAL_SHELL_SITE {...}	Create a spherical shell release site called *name*. Molecules are distributed on a spherical shell at the defined radius of the object. For now, you must specify the number to distribute, not a concentration. (This is the same as using the SHAPE=SPHERICAL_SHELL command inside RELEASE_SITE.)

The following commands define where, what, and when a release object releases molecules:

Command	Explanation
SHAPE = *geometry*	Release molecules in the specified shape (only used with RELEASE_SITE). Valid shapes are CUBIC, SPHERICAL, and SPHERICAL_SHELL, or the name of a region on which to release. The region must already be instantiated or be inside the same OBJECT as the release site (see OBJECT command). Region names can be combined with + to indicate release on both regions, - to indicate the release occurs on the first and not the second, and * to indicate the release occurs only where the two regions overlap. Parentheses may be used for grouping. If the molecule has a 3D diffusion constant, it will be released in the volume bounded by the regions and each region must be closed; otherwise, it will be released on the surface.
LOCATION = [*x*,*y*,*z*]	The release occurs centered at this location. Only used for geometrical shapes, not releases on regions.
MOLECULE = *name*	The named molecule is the one that will be released.
SITE_DIAMETER = *d*	The release site has diameter *d*. Not used for releases on regions.
SITE_DIAMETER = [*x*,*y*,*z*]	Release is asymmetric with a different diameters in different directions, as indicated by the vector. Not used for releases on regions.
RELEASE_PROBABILITY = *p*	This release does not occur every time, but rather with probability *p*. (If omitted, the default is to release without fail.)

Command	Explanation
NUMBER_TO_RELEASE = *n*	Release *n* molecules. For releases on regions, *n* can be negative, and the release will then remove molecules of that type from the region. To remove all molecules of a type, just make *n* large and negative. It is unwise to both add and remove molecules on the same timestep—the order of addition and removal is not defined in that case.
CONCENTRATION = *c*	Release molecules at concentration *c* molar for volumes and *c* molecules per square micron for surfaces.
GAUSSIAN_RELEASE_NUMBER {...}	Release molecules according to a Gaussian distribution, specified by the following two commands.
MEAN_NUMBER = *n*	The mean of the Gaussian distribution.
STANDARD_DEVIATION = *s*	The standard deviation of the Gaussian distribution.
RELEASE_PATTERN = *name*	Use the named release pattern instead of the default. The default is to release the specified number of molecules at the beginning of the simulation.

Release patterns are defined as follows.

Command	Explanation
DEFINE_RELEASE_PATTERN *name* {...}	Define a new release pattern that contains the commands below.
DELAY = *t*	The release pattern will start at time *t*. (Default is to start at time zero.)
RELEASE_INTERVAL = *t*	During a train of releases, release molecules after every *t* seconds.
TRAIN_DURATION = *t*	The train of releases lasts for *t* seconds.
TRAIN_INTERVAL = *t*	A new train of releases happens every *t* seconds. Behavior is undefined if the interval is less than the duration.
NUMBER_OF_TRAINS = *n*	Repeat the release process for *n* trains of releases.
NUMBER_OF_TRAINS = UNLIMITED	Repeat trains forever.

3.4.4 Geometrical transformations

At the end of the definition of a release object or geometrical object, or in the block where an object is instantiated, it can be moved using the following transformation commands (placed at the end of the block before the closing brace).

Command	Explanation
TRANSLATE = [*x*,*y*,*z*]	Move the object by the specified vector.
SCALE = [*x*,*y*,*z*]	Scale the object by multiplying each coordinate by the corresponding value in the vector.
ROTATE = [*x*,*y*,*z*] , *A*	Rotate *A* degrees about the axis defined by the supplied vector.

3.4.5 Instantiation of objects

Objects can contain lists of other objects using

Command	Explanation
name OBJECT { ... }	Define a new object called *name*. Inside the braces, list other objects one at a time to be added (see below).
INSTANTIATE *name* OBJECT { ... }	Define a new object called *name* and insert it into the world.
newname OBJECT *oldname* { ... }	Add the object called *oldname* into the existing object and label it *newname*. You can add extra commands (e.g. transformation) inside the braces. The two names can be the same thing. Thereafter, this object can be referred to in the world as *name*.*newname*.

You can also directly add release sites, boxes, and polygon objects into another object.

3.5 Output specification commands

There are two forms of output in MCell3, visualization output and count output. Visualization output typically contains the molecules and/or geometry of the model in a form suitable for visualization or analysis that requires knowledge of the precise location of particles. Count output reports running totals of summary statistics such as the total number of molecules of a certain type in the world, the number of times a reaction has occurred inside some object in the world, and so on.

Each mdl file can have multiple visualization blocks, but they must all have the same mode (see below). The visualization commands are:

Command	Explanation
`VIZ_DATA_OUTPUT { ... }`	Define a new visualization data output block which contains the commands below.
`MODE =` *mode*	Use *mode* as the output mode. Unless you know what you're doing, use `DX`.
. . .	(This is the same as MCell 2, I think.)

Each mdl file can also have multiple reaction data output blocks. Each block consists of the following:

Command	Explanation
`REACTION_DATA_OUTPUT { ... }`	Define a new count data output block which contains the commands below.
`OUTPUT_BUFFER_SIZE =` *N*	Write output to disk after every *N* blocks. The default is *N*=10000. This command is optional, but must be first if it is used. The output will also always be written when the simulation terminates, regardless of *N*.
`STEP =` *t*	Output this block every *t* seconds.
`TIME_LIST = [` *list* `]`	Output this block at the times specified in the list.
`ITERATION_LIST = [` *list* `]`	Output this block at the iteration numbers specified in the list (i.e. after that number of timesteps).
`{` *value* `} => "`*file*`"`	Output the value in braces to the filename in parentheses. The first column will be the timestep; the second will be the value listed. This command can be repeated to send different output to many files.
`{` *value* `,` *value* `, ... } => "`*file*`"`	Output the list of values in braces, one to a column, in the order listed. The first column will be the timestep. This command can be repeated.

The values specified in braces are count statements, or mathematical operations applied to count statements (e.g. you can add, subtract, etc. count statements to each other and to constants and so on). The count statements themselves have the following syntax:

Command	Explanation
`COUNT[`*name* `, WORLD]`	Count things in the world. If *name* refers to a molecule, count how many of that molecule are in the world. If it refers to a reaction, count how many times that reaction has occurred so far.

Command	Explanation
COUNT[*name , object*]	Count things inside the object called *object*. This must be an instantiated object. For example, if you have instantiated an object called my_world with a box called my_box inside it, *object* would be my_world.my_box. If you are counting surface molecules or reactions at a surface, only the ones that actually occur on *object* will be counted. Molecules with a 3D diffusion constant will be counted inside the object, but the object must be closed.
COUNT[*name , region*]	Count things inside the named region. It must be referenced fully. E.g. if my_box (from above) has a region called my_region, the name would be my_world.my_box[my_region]
COUNT[*name,region,*ALL_ENCLOSED]	Count all reactions that occur inside this region (not counting those that occur on the surface of the region). This lets you count surface molecules contained within a box, for example. This will work with objects as well as regions, but the object or region must be closed.
COUNT[*name,region,*CONCENTRATION]	Estimate the concentration of the molecule at that region, averaged since the beginning of the simulation (output has units of μM). A single object can be used instead of a region, but multiple objects cannot be used. The region/object does not need to be closed. To find the average concentration during one count interval, let t_i be the time of the ith output, let t_j be some earlier output, and let $\bar{c}(t)$ be the concentration averaged up to time t. Then the average concentration between times t_j and t_i is $\bar{c}(t_j \to t_i) = \frac{t_i\bar{c}(t_i) - t_j\bar{c}(t_j)}{t_i - t_j}$. Note that this is the concentration all around the surface, so if the molecule can only reach one side, the concentration on that side will be twice what is reported here.
COUNT[*name,region,hits*]	Output the number of times the named molecule has hit the named region (or object). The *hits* specifier should be one of FRONT_HITS, BACK_HITS, ALL_HITS, FRONT_CROSSINGS, BACK_CROSSINGS, and ALL_CROSSINGS.

3.6 Utility commands

MCell3 understands the standard numeric operations + - * / as well as the following standard functions:

Command	Explanation
SQRT(*x*)	Return the square root of *x*
EXP(*x*)	Return the value of *e* raised to the x^{th} power
LOG(*x*)	Return the natural logarithm of *x*
LOG10(*x*)	Return the base 10 logarithm of *x*
SIN(*x*)	Return the sine of *x*
COS(*x*)	Return the cosine of *x*
TAN(*x*)	Return the tangent of *x*
ASIN(*x*)	Return the inverse sine of *x*
ACOS(*x*)	Return the inverse cosine of *x*
ATAN(*x*)	Return the inverse tangent of *x*
ABS(*x*)	Return the absolute value of *x*
CEIL(*x*)	Return the smallest integer at least as big as *x*
FLOOR(*x*)	Return the largest integer at no bigger than *x*
MAX(*x,y*)	Return the larger of *x* and *y*
MIN(*x,y*)	Return the smaller of *x* and *y*
RAND_UNIFORM	Return a random number uniformly distributed between 0 and 1
RAND_GAUSSIAN	Return a random number from a Gaussian distribution with mean 0 and standard deviation 1.

Command	Explanation
PI	The numeric value $\pi = 3.14159265358979323846$
SEED	The value of the random number generator seed

At any outer block in MCell3, one can define variables simply by assigning a value to the name of the variable. E.g. `my_lucky_number=13` would be a valid (if unusual) way to define a variable. Variables can take numeric, array, or string values. String values consist of text between double quotes. Array values are lists of numbers inside brackets separated by commas, or starting and ending values plus a step size, as exemplified below (note the double brackets):

```
my_lucky_number = 13
my_favorite_array = [1,3,5,7,11,17]
my_second_favorite_array = [[1.3 TO 2.75 STEP 0.331]]
my_boring_string = "la la la, la la la"
```

The C-style `printf` and `sprintf` commands work too, pretty much the way you'd expect them to. MCell3 comments are delimited by `/*` and `*/`.

4 Technical details affecting simulation speed and accuracy

4.1 Partitioning

In future releases, MCell3 will automatically partition space to improve execution speed. Currently, however, this must be performed manually. In general, partitions should be chosen to avoid having too many surfaces and molecules in one subvolume defined by the partitions. Molecules that are specified as `TARGET_ONLY` or which do not interact with other molecules diffusing in 3D need only have relatively few surfaces in one subvolume.

If there are few surfaces and/or molecules in a subvolume, it is advantageous to have the subvolume as large as possible. Crossing partition boundaries takes a small amount of time, so it is rarely useful to have partitions more finely spaced than the average diffusion distance of the faster-moving molecules in the simulation.

In cases where the diffusing molecules do not interact with each other, they can safely take extended time-steps by measuring how far they are from things they could interact with. In this case, the partitions with no surfaces should be as large as possible. For example, a box works well with partitions just inside its outer walls.

Finally, note that partition placement is not exact. The model is divided into 16384 possible partition boundaries, so partitions may shift by up to about one part in twenty thousand of the size of the model. For instance, if the model has a structure that is 6μm long, partitions may vary by about 0.0003μm. Thus, do not place partitions too close to objects in your model or they may not appear on the side you expect them to appear.

4.2 Mean diffusion distance

Diffusion in MCell3 (and in earlier versions of MCell) is modeled as a series of motions in a straight line. This is a good approximation around geometry that is of a larger scale than the mean diffusion length for the timestep of the molecule in question. For accurate results around intricate geometry, it may be necessary to reduce the time step (or space step).

4.3 Reaction probabilities

MCell3 assigns a probability of reaction for each collision. These probabilities are chosen to match the bulk reaction rate specified in the mdl file. The match will not occur, however, if the probability goes above 1.0. Internal correction factors can also raise the actual probability above the typical probability specified at the beginning. Therefore, MCell3

will output a warning if the reaction probability goes above 0.8 for reactions where a 3D diffusing molecule hits a surface, or if the probability goes above 0.3 for a collision between pairs of 3D diffusing molecules.

If warnings are given (and possibly even if they are not), one should reduce the time step to lower the probabilities and see if the same results are generated. If not, simulations should be run with shorter time steps in order to avoid overly high probabilities.

Unimolecular reactions with half-lives of less than one time step are also not perfectly accurate. Although unimolecular transitions will always occur at the right rate, other molecules may not experience the right effective concentration of each state, since a short-lifetime species may not be converted to another species until the end of the time step after which many other molecules may have had a chance to interact with it. Thus, the shortest-lifetime species in a series of unimolecular transitions should not have a half-life of less than approximately one time step if other molecules can interact with that state.

4.4 Interaction radii

Bimolecular reactions occur within a distance specified by the `INTERACTION_RADIUS` command. In many cases, one may want to increase or decrease this value. In particular, in order to get the right probability of reaction, MCell3 increases the probability of reaction when near surfaces and partition boundaries. In general, the reaction rate has approximately 1-2% error if the average spacing between surfaces is at least 10 times the interaction radius, and the reaction probabilities are 0.3 or less.

For example, if one has partitions spaced 0.02μm apart, simulation accuracy will be poor with the default interaction radius of 0.01μm. Thus, one might wish to specify `INTERACTION_RADIUS=0.001`.

4.5 Placing molecules in the world

There are two ways to place molecules on surfaces: with a release site on a region, and as part of the property of a surface or region. Release sites are more flexible but slower; if you do not need the flexibility of release site notation, you're better off defining a region and using the `MOLECULE_DENSITY` or `MOLECULE_NUMBER` commands to add molecules at initialization.

All placement of molecules in volumes is done with release sites. However, the geometrical release sites (`CUBIC` and `SPHERICAL`) require less computation to place each molecule. Thus, these should be used preferentially for simple geometry. To release particles at a point, use a cubic release site and set the diameter to 0.

5 Example models

5.1 Ligand-gated ion channel

Below are a set of molecule definitions and reactions that specify an ion channel that is gated by the binding of a single ligand.

```
DEFINE_MOLECULES {
  channel_unbound { D_2D=0 }
  channel_bound   { D_2D=0 }
  channel_open    { D_2D=0 }
  ligand          { D_3D=2e-8 }
  ion             { D_3D=3e-8 }
}
DEFINE_REACTIONS {
  channel_unbound' + ligand' -> channel_bound'             [1e7]
```

```
  channel_bound'             -> channel_unbound' + ligand' [2e2]
  channel_bound'             -> channel_open'              [5e2]
  channel_open'              -> channel_open'    + ion,    [8e4]
}
```

We have defined a reaction where a ligand binds to one end of a channel (presumably the extracellular face), which causes the channel to be in its bound state. In that state it can either release the ligand or become open. While open, it will emit ions on the other end (presumably the intracellular face). This would be suitable if the ion concentration is much higher outside than inside, or the membrane potential makes it highly favorable for the ion to move inside, so that we don't have to worry about the reverse reaction. If there is no electrical driving force, we might have to model ions both inside and outside:

```
DEFINE_REACTIONS {
  channel_unbound' + ligand' -> channel_bound'             [1e7]
  channel_bound'             -> channel_unbound' + ligand' [2e2]
  channel_bound'             -> channel_open'              [5e2]
  channel_open + ion'        -> channel_open     + ion,    [4e7]
}
```

Here, the ion travels in either direction just as easily since it pays no attention to the orientation of the channel. However, if there was a modest driving force, traveling in might be easier than traveling out, which would be reflected in the rates.

```
DEFINE_REACTIONS {
  channel_unbound' + ligand' -> channel_bound'             [1e7]
  channel_bound'             -> channel_unbound' + ligand' [2e2]
  channel_bound'             -> channel_open'              [5e2]
  channel_open' + ion'       -> channel_open'    + ion,    [4e8]
  channel_open' + ion,       -> channel_open'    + ion'    [1e8]
}
```

In this case, the ion is four times as likely to travel from outside to inside as inside to outside. Finally, note that most of the orientation marks are unnecessary given that the default is to be in orientation class zero facing up. Thus, we can write the last set of reactions as

```
DEFINE_REACTIONS {
  channel_unbound + ligand -> channel_bound            [1e7]
  channel_bound            -> channel_unbound + ligand [2e2]
  channel_bound            -> channel_open             [5e2]
  channel_open' + ion'     -> channel_open'   + ion,   [4e8]
  channel_open' + ion,     -> channel_open'   + ion'   [1e8]
}
```

to achieve the same effect.

5.2 Example bimolecular reaction

Here's a complete mdl file that implements a simple bimolecular reaction that should achieve equilibrium at 482 molecules of each species.

```
time_step = 1.0e-6
TIME_STEP = time_step
TIME_STEP_MAX = time_step
ITERATIONS = 1e-2/time_step
EFFECTOR_GRID_DENSITY = 10000
INTERACTION_RADIUS = 0.001
PARTITION_X = [ [-0.1 TO 0.1 STEP 0.01] ]
PARTITION_Y = [ [-0.1 TO 0.1 STEP 0.01] ]
PARTITION_Z = [ [-0.1 TO 0.1 STEP 0.01] ]
DEFINE_MOLECULES
{
  A { D_3D = 100e-8 }
  B { D_3D = 100e-8 }
  C { D_3D = 100e-8 }
}
/* Your basic reversable binding reaction */
DEFINE_REACTIONS
{
  A + B -> C [1e7]
  C -> A + B [1e3]
}
small_box BOX
{
  CORNERS = [-0.1,-0.1,-0.1] , [0.1,0.1,0.1]
  /* REMOVE_ELEMENTS { TOP,LEFT } */  /* Could remove sides ... */
  /* REMOVE_ELEMENTS { INCLUDE_PATCH = [0.1,0,0] , [0.1,0.05,0.05] } /*... or patch*/
}
INSTANTIATE my_world OBJECT
{
  A_release CUBIC_RELEASE_SITE {
    LOCATION=[0,0,0]
    MOLECULE=A
    NUMBER_TO_RELEASE=482
    SITE_DIAMETER=0.196
  }
  B_release CUBIC_RELEASE_SITE {
    LOCATION=[0,0,0]
    MOLECULE=B
    NUMBER_TO_RELEASE=482
    SITE_DIAMETER=0.196
  }
  C_release CUBIC_RELEASE_SITE {
    LOCATION=[0,0,0]
    MOLECULE=C
    NUMBER_TO_RELEASE=482
    SITE_DIAMETER=0.196
  }
 my_box OBJECT small_box {}
}
REACTION_DATA_OUTPUT
{
  STEP = 1e-5
  { COUNT [A,WORLD] } => "eq_A.dat"
  { COUNT [B,WORLD] } => "eq_B.dat"
```

```
  { COUNT [C,WORLD] } => "eq_C.dat"
}
```

This isn’t finished, but we’ll just stop here.