

The MCell Documentation

Version 2.72

April 28th, 2004

Lin-Wei Wu Thomas M. Bartol Jr.

Copyrights(C) 2001-2004

Table of Contents

Chapter 1. Introduction	4
1.1 Simulation of the Microphysiological Models.....	5
1.2 MCell: Simulator of Cellular Microphysiology	7
1.2.1 Running MCell	7
1.2.2 The MCell Simulation	8
Chapter 2. Anatomy of the MCell Model Description Language	9
2.1 Anatomy of MDL Files	9
2.2 How To Create MCell User Variables.....	11
2.3 MCell Arithmetic Operators And Operations	12
Chapter 3. The MCell Model Descriptive Language (MDL)	14
3.1 The MCell Comments.....	14
3.2 Two Required Simulation Parameters	14
3.3 How To Include Other MDL Files	15
3.4 Checkpointing in MCell.....	17
3.5 MCell Initialization and the Related MDL Parameters	19
3.5.1 MCell Initialization	19
3.5.2 MCell Time Step Iterations	21
3.5.3 RADIAL_SUBDIVISIONS.....	22
3.5.4 RADIAL_DIRECTIONS.....	23
3.5.5 EFFECTOR_GRID_DENSITY.....	23
3.5.6 Spatial Partitioning.....	24
3.6 Logical Objects Used in MCell	25
3.6.1 Defining A Ligand Molecule	26
3.6.2 The Ligand Release Pattern.....	27
3.6.3 The Chemical Reaction Mechanisms.....	28
3.7 Templates of The Physical Object	32
3.7.1 The Ligand Release Site	32
3.7.2 The Box Object	35
3.7.3 The Polygon List Object	37
3.7.4 The Meta Object	39
3.7.5 The Instantiated Object.....	40
3.8 Surface Modifiers Definition of the Surface Regions.....	41
3.8.1 Operations Applied to the Surface Element	42
3.8.2 Permeability of Physical Objects	42

3.8.3 Defining Surface Regions on Object Templates.....	43
3.8.4 How to Add Effectors	44
3.9 Geometrical transformation (geometric_transformation)	47
3.9.1 TRANSLATE.....	48
3.9.2 ROTATE	49
3.9.3 SCALE	53
3.10 MCell Output.....	54
3.10.1 The Visualization Output	55
3.10.2 The Chemical Reaction Output.....	58
3.10.3 The Generic "C" Language Output.....	61

Chapter 1 Introduction

MCell is a general Monte Carlo simulator of cellular microphysiology. MCell may be used for the study of biological signal transactions, namely the microphysiology of synaptic transmission, and other areas, including: statistical chemistry, molecular diffusion, single channel or multi-channel simulation, data analysis, noise analysis, Markov processes, and other molecule dynamic applications.

MCell simulations take place on a sub-cellular biological scale (Figure 1). In areas of computational neurobiology, sub-cellular communication is based on a wide variety of chemical signaling pathways. A process like synaptic transmission encompasses neurotransmitter and neuromodulator molecules, proteins involved in exo- and endocytosis, receptor proteins, transport proteins, and oxidative and hydrolytic enzymes. MCell makes it possible to incorporate high resolution ultrastructure into models of ligand diffusion and signaling, and to provide highly realistic 3D simulations of sub-cellular architecture and physiology.

Physical Scales in Computational Neurobiology

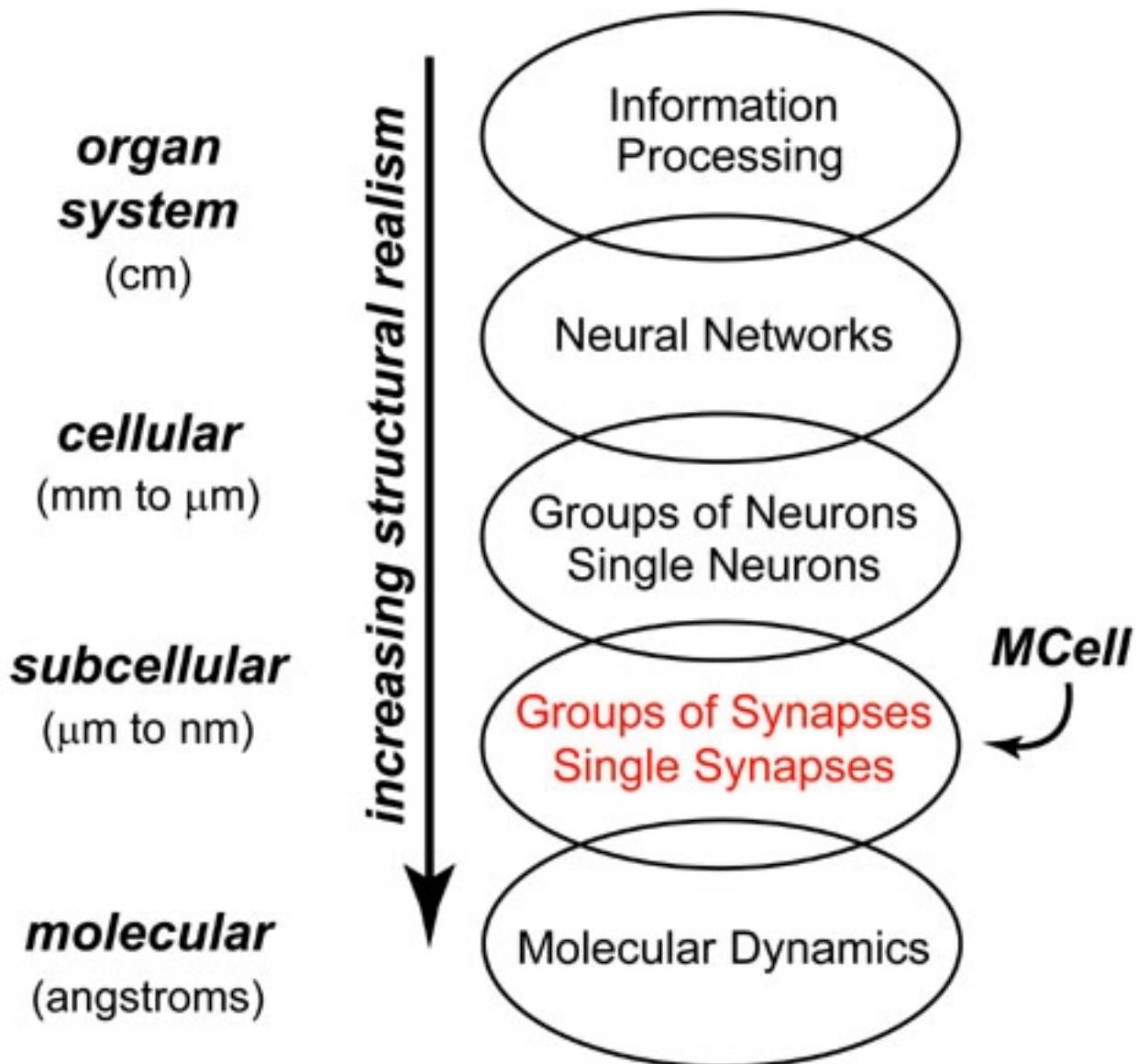


Figure 1.1 Physical Scales of Computational Neurobiology

Section 1.1 Simulation of the Microphysiological Models

With increased computing power, computer simulations may be performed for larger-scale models. Computer simulation may be used to help solving problems in a very efficient and economic way, and testing or verifying scientific hypotheses so as to better direct research experiments. All simulations essentially encompass three steps: a. modeling; b. simulation; c. visualization (or result analysis). These three steps are strongly correlated: While a well designed model is key to success, visualization also plays an important role: since simulation data can be converted into 3D graphics, it is possible to compare a model with the real object

and as a result further improve the simulation. Table 1.1 shows the basic steps involved in a computer simulation.

Table 1.1 A Generalized Computer Simulation

Table1.1

Modeling	Defining simulation parameters Organizing simulation objects Designing simulation algorithms
Simulation	Initialization Iteration Exporting simulation results
Visualization	Visualization mode selection Simulation results animation

Modeling is the first step for any kind of simulation. A model is the definition of an application and a detail plan about how to perform this application. A well designed simulation model should include all the information needed by the simulator, such as simulation objects (physical objects and logical objects), simulation parameters, simulation mechanisms, optimizing algorithms and so on. The model may be translated into a simulation input file in the format required by the simulator to perform the simulation.

MCell was initially designed for microphysiological simulations. To simulate a realistic microphysiological system, representations of the relevant cellular and sub-cellular structures must be generated at very high resolutions (the electron microscope level). To achieve this, we first use biochemical methods to treat some tissue for Electron Microscope (EM) to take pictures (this step is also called image processing). Then the EM image is scanned to transform the image into digital data, the digital data are then used to reconstruct a 3D digital image for further study (simulation); this step is called 3D reconstruction. The simulation object may be an object from the real world (part of neural muscular junction, a channel, a clamp, a piece of membrane etc.) or a computer-generated three-dimensional object. The user may also use the MCell Model Descriptive Language (MDL) to define a physical object as the 3D simulation object.

After you have the simulation object and the simulation model in hand, you are ready to run a simulation. The simulation will be done in three steps:

- a. Simulation initialization;
- b. Time-step iteration;
- c. Output the simulation results.

After a simulation is finished, the user may collect the data for visualization; the simulation model may be further improved based on the effect of the visualization. A more refined simulation can be started after modifications of the model.

MCell can export the simulation results in a variety of data formats: 2D numerical data, 3D animation data, and "C" style language data. There are also several popular visualization data format available for the user's convenience. To define 3D visualization outputs, just select for the right output mode in the MCell visual data output statement. The available visualization modes in MCell are DX (IBM's open source, [DataExplorer](#)), IRIT (open source, [irit](#)), RAYSHADE (open source, [rayshade](#)), POVRAY (open source, [povray](#)) and RENDERMAN

(open source, [Pixar's Renderman](#)). It is also possible to export numerical results and "C" language format results from a MCell simulation, and display them in various ways (Matlab, Mathematica and so on).

Section 1.2 MCell: Simulator of Cellular Microphysiology

MCell was initially designed for computational neurobiology but it is not limited to that area. It can also be applied to a wide range of micro-studies such as diffusion-reaction applications, molecular dynamics, and computational chemistry. The MCell team is working to add electronic physiology into the simulation in future versions.

In computational neurobiology, there are four typical events that are modeled by an MCell simulation. They are:

- a. The release of molecules from a structure (e.g., vesicle release);
- b. Molecular diffusion within spaces defined by arbitrary surfaces (e.g., pre- and postsynaptic membranes, or a cell membrane with attached patch clamp micropipette);
- c. Creation or destruction of molecules (e.g., synthesis, hydrolysis processes, or redox reactions);
- d. Chemical reactions undergone by ligand and "effector" (e.g., receptor or enzyme) molecules.

Ligands, effectors, reaction mechanisms, 3D surfaces, and other simulation components are specified using a simple descriptive language in MCell. The language is called MCell Model Description Language (MDL) and was designed with the biology-oriented users in mind - it is used to describe the simulation model.

Section 1.2.1 Running MCell

MCell can be executed under different operating systems: Windows, Windows NT, Free BSD, and UNIX. The MCell execution command is the same for each operating system. If the simulation input file is **file.mdl**, then the MCell execution command will be:

mcell [options] file.mdl

Options

[-help]	print help message
[-info]	print MCell info message
[-seed n]	choose random sequence number (default: 1), n is an integer between 1 and 3000
[-iteration n]	override iterations in the MDL file (file.mdl)
[-logfile filename]	send the output log to file (default:stderr)
[-logfreq n]	output log frequency (default:100)

The MCell simulation input file is defined by the MCell Model Description Language (MDL), and it provides MCell with all the necessary information for a simulation, which may include simulation components, simulation objects, output modes, simulation mechanisms and the simulation structure. There are no restrictions on the name of the input file, the file extension **.mdl** is not required by the MCell. We recommend the name format **file.mdl** only for the convenience of file organization.

Based on output specifications given in the simulation input file, MCell will export various

output files (space-dependent or time-dependent) for each specified molecule into a user-defined directory. The MCell rendering tool [DreaMM \(Design, render, animate MCell Model\)](#) is available from the MCell download site. MCell also has a selection of different visualization output modes, so the user is able to visualize any simulation result using a graphical tool of preference without the need for data conversion.

Section 1.2.2 The MCell Simulation

First, one or more MDL input files are interpreted to create the simulation objects. Next, MCell will start the simulation iterations. Each MCell iteration equals one Monte Carlo time-step. Via a feature called **checkpointing**, MCell can break up a simulation into smaller sets, each one simulating the model over given amounts of time. The simulation is based on the following assumptions being true at each time step:

- a. Every ligand (small molecule) has a position and is flagged as either free or bound by an identity value (flag).
- b. Free ligands move randomly and may interact with effectors (large molecule).
- c. The position of each effector is fixed; each effector is considered to be in one of the states of a given reaction schema.
- d. The state of an effectors is associated with a surface area and a probability of binding, given that a transmitter molecule hits the effector surface.

A MCell simulation contains two operations: initialization and time-step iteration. Movements of the molecules are calculated by Brownian random walk dynamics. The simulation is designed based on the Monte Carlo algorithms and some highly optimized 3D spatial partitioning algorithms. Interactions and chemical reactions between the molecules are calculated by the Monte Carlo probabilities. Detailed descriptions about the MCell algorithm will be discussed in [chapter 3](#).

Chapter 2 Anatomy of the MCell Model Description Language

The MCell Model Description Language (MDL) is the interface between the user and the MCell simulator. As you can see from the MCell simulation command (**mcell file.mdl n**), simulation models are described by MDL input files typically named, **file.mdl**.

MDL is a model description language. The syntax of MDL is similar to that of other high level computer languages. Users can create their own variables in MDL, but they cannot create functions because MDL is not a programming language. Likewise there are no loops or conditional statements allowed in a MDL file. If you really want to use a loop or conditional statement, you can use a scripting language to control your MDL file.

MDL consists of keyword statements (capitalized) that can be organized into functional blocks contained within a pair of curly braces. Each function consists of required arguments and optional arguments. We categorize keyword statements and functional blocks into 6 groups:

- a. comments;
- b. definition of the variables;
- c. simulation distribution;
- d. definition of simulation parameters;
- e. definition of logical and physical objects;
- f. output specification.

Section 2.1 Anatomy of MDL Files

MDL files are order-dependent. Whenever you run MCell with an MDL input file, MCell will interpret the MDL file in a top-to-bottom order. Initialization will be done during MDL parsing. Most MDL keyword statements are optional, which means you only use them as needed. However, there are two required MDL keywords for all MDL files: time step of each simulation iteration and the total number of iterations of the simulation. The anatomy of a well organized MDL file is given below.

Anatomy of the MDL files

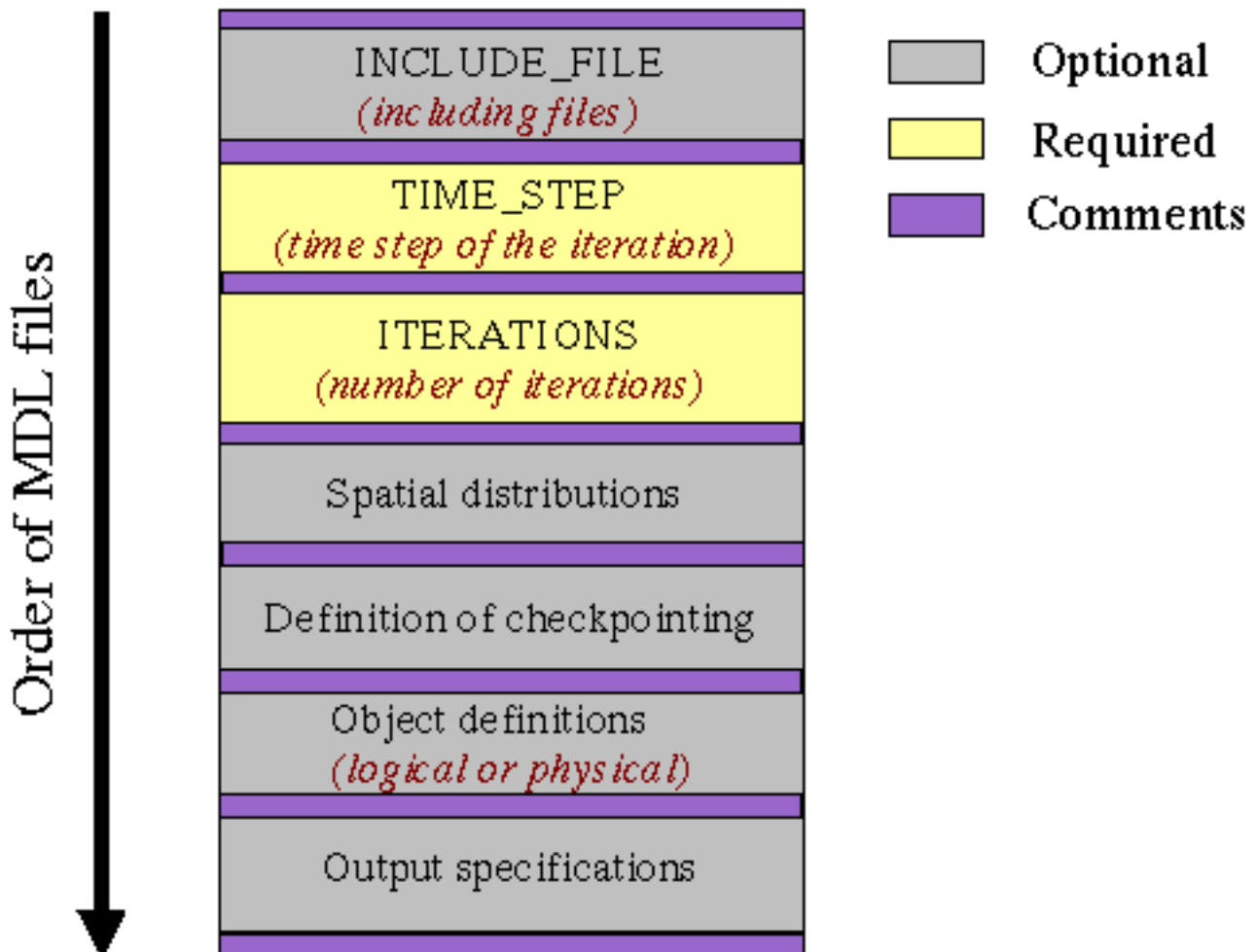


Figure 2.1 General anatomy of an MDL file. MDL files are order dependent as shown in the figure. Comments may appear anywhere inside these files; time step and number of iterations are the only required parameters.

Comments

Comments are optional statements in a MDL file. They may appear anywhere inside the file. Any contents present inside the tag `/*` and `*/` are regarded as comments.

Include file

This is an optional method for the MCell simulation. The MDL keyword `INCLUDE_FILE` is used to include other MDL files into the current MDL file. The parameters and functions defined inside the included files may be used freely in the current MDL file after this command.

Check-pointing

Check-pointing is an optional method in the MCell simulations, it is defined by the following MDL commands:

```
CHECKPOINT_ITERATIONS
CHECKPOINT_INFILE
CHECKPOINT_OUTFILE
```

TIME_STEP & ITERATIONS

The two required parameters for a MCell simulation are:

ITERATIONS - Number of the iterations in the simulation.

TIME_STEP - The time intervals between two consecutive iteration steps.

Spatial distribution

Spatial distributions are used to initialize a MCell simulation--if no spatial distributions are specified, MCell will assign pre-defined default values to the simulation. Spatial distribution parameters include:

RADIAL_SUBDIVISION

RADIAL_DIRECTIONS

EFFECTOR_GRID_DENSITY

PARTITION_X, PARTITION_Y, PARTITION_Z (spatial partitioning)

Logical object definition

Logical objects include small molecule "ligand", large molecule "effector", and any chemical reactions between them. MDL functions that are used to define logical objects are:

DEFINE_LIGAND

DEFINE_RELEASE_PATTERN

ADD_EFFECTOR

DEFINE_REACTION

Physical object definition

Physical objects are objects which occupy a volume in 3-D space. In MCell physical objects include: box objects, spherical ligand release site, and polygon list objects. You can put physical objects into a meta object template or instantiate them. These physical objects are defined by the following functions:

BOX

SPHERICAL_RELEASE_SITE

POLYGON_LIST

OBJECT

INstantiate ... OBJECT

Output specification

You can select three different output formats for your MCell simulation results:

visualization output (VIZ_DATA_OUTPUT)

Chemical reaction output (REACTION_DATA_OUTPUT)

General C language output (fwrite, fprintf, fopen, fclose).

Section 2.2 How To Create MCell User Variables

You may assign text variables, numerical variables, and numerical arrays (whose elements are numerical values or numerical expressions) in a MDL file. You may also express text and numerical expressions in MCell. A text expression is a series of concatenated text, and a numerical expression is an arithmetic relational expression.

MDL recognizes text variables when they are enclosed by quotation marks. All numerical data are regarded as double precision internally, so there is no need for variable-type definition when creating numerical variables. Also, user-created variables do not need to be capitalized like MDL keywords, and they may be used anywhere in a MDL file. There is no restriction for naming variables, but by using carefully chosen variable names you can make your program self-documenting - MDL lines themselves would tell whoever might be reading the code what it does. By reading the definition in Table 2.2-1, you will learn how to create your own simulation variable.

Table 2.2-1 How To Create The User Variables

Table2.2-1

Variable	Definition	Format	Annotation
Text variable	Assign literal text or combinations of literal text and text variables to the defined text variable.	text_variable_name ="literal text as text value" text_variable = "literal text piece 1"&"literal text piece 2"& ... & text_variable_name & ...	text_variable can be expressed by a single literal text string or several concatenated text segments.
Numerical variable	Assign a numerical value to a variable, the assignment can be combined with mathematical operations.	numerical_variable_name = numerical_value numerical_variable_name = numerical_expression	None
Numerical array	Two ways to define a numerical array in MCell: List each element of the array in the square brace, separate each element with a comma; - OR - Express the array by specifying the range and step of the array.	array_variable_name = [a1, a2, ... , an] - OR - array_variable_name = [a1 TO an STEP delta]	Array elements can only be numerical data. MDL doesn't support text arrays. Index of an array always starts from zero.

Example 1

Definition of a text variable using text expressions.

MCell Code

In this example we first define a text variable named "parameter_definition"; then two literal text expressions and one text variable are used to define a new compound string, which is named "MCELL_MENU".

```
parameter_definition = "MDL commands and functions."
MCELL_MENU = "The MCell menu consists of an introduction part,"&"a theory part, and"&
parameter_definition
```

The result is:

```
MCELL_MENU = "The MCell menu consists of an introduction part , a theory part, and MDL
commands and functions."
```

Example 2

Express a numerical variable and an array in MCell. The numerical array CHECK_STEPS_ARRAY (10 elements inside) is defined by two different methods.

MCell Code

```
OSCILLATION_FREQUENCY = 5*103
channel_width = 0.12
scalar = 1
CHECK_STEPS_ARRAY = [100 TO 1000 STEP 100]
CHECK_STEPS_ARRAY = [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]
```

Section 2.3 MCell Arithmetic Operators And Operations

MDL supports the typical arithmetic operators and numerical operations. Arithmetic operators

can apply to both the user-created variables and MDL-defined variables (the MDL keywords). Numerical expressions can appear anywhere inside or between MDL blocks in a MDL file.

MCell supports five standard numerical operators, they are:

- +** plus operator
- minus operator
- *** times operator
- /** divided operator
- ^** exponential operator

A numerical expression combines standard arithmetic operators (+, -, *, /, ^), mathematical functions, the value p (can be expressed as PI in the MCell), user created variables, and MDL keywords. The available mathematical functions are given in Table 2.3-1.

Table 2.3-1 Mathematical Functions of MCell

Table2.3-1

SQRT(n)	compute the square root of n.
EXP(n)	compute the exponential value of n.
LOG(n)	compute the native logarithm of n.
LOG10(n)	compute the base 10 logarithm of n.
SIN(n)	compute the sine value of the radians n.
ASIN(n)	compute the arcsine value of n.
COS(n)	compute the cosine value of the radians n.
ACOS(n)	compute the arccosine value of n.
TAN(n)	computes the tangent value of the radians n.
ATAN(n)	Compute the arctangent value of n.
ABS(n)	compute the absolute value of n.
MOD(n1, n2)	compute the modulus of n1 with respect to n2.
x = MIN(y, z)	Assign the minimum value of y and z to x.
x = MAX(y, z)	Assign the maximum value of y and z to x.
x = CEIL(y)	x is assigned smallest integer not less than y.
x = FLOOR(y)	x is assigned largest integer not greater than y.
x = ROUND_OFF(y,z)	x is assigned z rounded to y significant digits (NOT y decimal places) with trailing 0's truncated.

Chapter 3 The MCell Model Descriptive Language (MDL)

The MDL file is the core part of the MCell simulation, MDL file will be edited and saved as a simulation input file to execute the MCell simulation, it will provide the simulator all the necessary information for the simulation.

MDL file is consisted with the MDL keyword (commands), MDL functions, MDL templates and the user created variables. All the MDL keyword statement (both the commands and the functions) are capitalized. The MDL functions are defined inside a pair of curly brace {}, each MDL function contains one or more MDL commands as arguments. Some of the arguments of the function are the required arguments, and some are optional arguments. In this chapter, we introduce the format and usage of all the MDL commands and functions.

Section 3.1 The MCell Comments

Like any other programming language, comments are widely used in MDL files to increase the readability of the files. Comments are allowed anywhere (inside or outside the functional block) in a MDL file. The MCell parser will ignore comments during simulation.

Comments are any contents stated between the start tag (*/**) and end tag (**/*). The format of a MCell comments is shown in Table 3.1-1.

Table 3.1-1 The MCell Comments

Table3.1-1

Format	<i>/*</i> All contents inside here are MCell comments. <i>*/</i>
--------	--

Section 3.2 Two Required Simulation Parameters

In an MCell simulation, molecular diffusion and chemical reactions are the two major applications. Parameter time plays an important role in both by controlling the simulation process. Total time of a simulation is the product of number of iterations and time step of the iteration. The MDL commands ITERATIONS(total number of iteration of the simulation) and TIME_STEP(time interval between two adjacent iterations) are the only two required simulation parameters in a MCell simulation. The simulation will print out an error message if you try to run a simulation without specifying ITERATIONS and TIME_STEP.

Table 3.2-1 shows the definition of the MDL command ITERATIONS and TIME_STEP.

Table 3.2-1 ITERATIONS and TIME_STEP

Table3.2-1

Format	ITERATIONS = n	TIME_STEP = c
Annotation	Number n is an integer number assigned to ITERATIONS. <i>ITERATIONS is a required simulation parameter.</i>	Number c is a real value assigned to TIME_STEP. <i>TIME_STEP is a required simulation parameter.</i>
Unit	None	Seconds
Default	None	None

ITERATIONS

The MDL command ITERATIONS defines the number of iterations that will be executed in a MCell simulation. There is no default value for this parameter in the MCell simulator. It is a required simulation parameter for any MCell simulation.

TIME_STEP

The MDL command TIME_STEP specifies the time interval between each iteration. TIME_STEP is expressed in *seconds* and there is no default value in the MCell simulator for this parameter. TIME_STEP is a required simulation parameter. TIME_STEP is also the only parameter that would effect the convergence of a MCell simulation.

Section 3.3 How To Include Other MDL Files

You may add pre-defined MDL files into your simulation input file using the MDL keyword INCLUDE_FILE. The goal of MCell INCLUDE_FILE is to use MCell conveniently from an on-line repository. To this end, included files should have a consistent structure to minimize the need for editing. The design of the MCell INCLUDE_FILE is similar to functional design in procedure programming, where frequently used (or modified) functions or parameter groups are saved into a separate file as the included file, then we can assign its name to the MDL command INCLUDE_FILE. The MCell parser will insert the included file automatically when it meets the keyword INCLUDE_FILE.

INCLUDE_FILE is the MDL command to include an existing MDL file into a simulation input file. It can only include one file per usage, but you are allowed to use the command INCLUDE_FILE as many time as you like in your MDL file. There are two ways to include files using the command INCLUDE_FILE:

- a. Double quote the name of the file that you are going to include and assign it to the keyword INCLUDE_FILE;
`INCLUDE_FILE = "file_name"`
- b. If the included file name has been assigned to a user-defined variable in the pre-included file, you can assign the related user-defined variable name to INCLUDE_FILE.
`INCLUDE_FILE = variable_name`

There are no naming restrictions for the included file. When you assign a filename to INCLUDE_FILE, just double quote the file name; No double quotes are needed for a variable name. You can freely use variables and parameters defined in the included file after the INCLUDE_FILE statement in the simulation. Figure 3.1 shows the usage of the command INCLUDE_FILE in an MCell simulation.

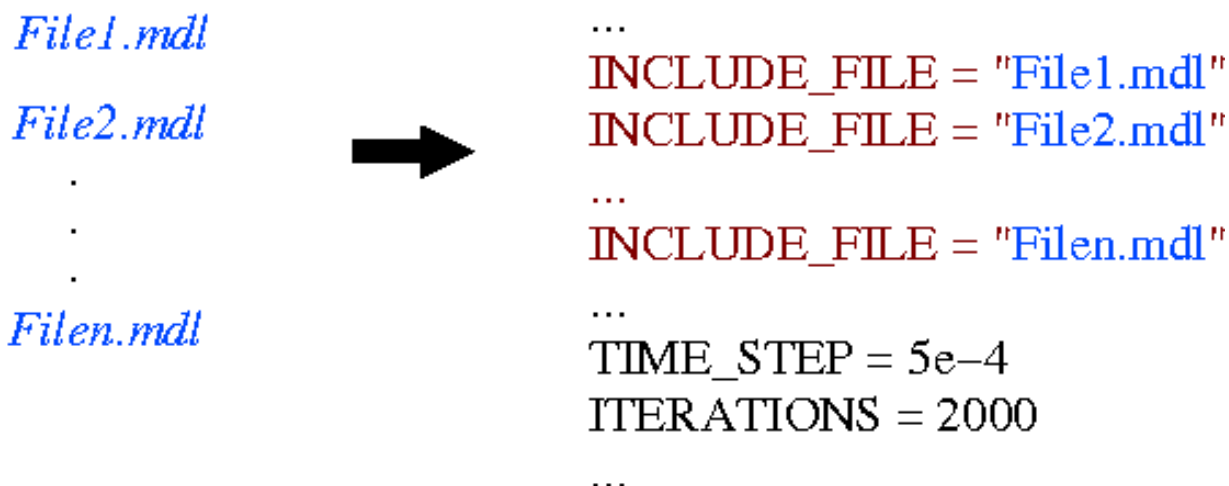
*Pre-defined MDL files**Simulation input file*

Figure 3.1 If you have the function blocks and simulation parameters saved in two separate MDL files (as *File1.mdl*, *File2.mdl* in the figure), you can use the MDL command `INCLUDE_FILE` to add them into your simulation input file. You can have as many include files as you need.

We give two examples to introduce how to use the MDL command `INCLUDE_FILE` in a MCell simulation.

Example 1

The simulation input file named "unbound_diffusion.mdl" is used to simulate the unbounded diffusion of two ligands (red ligand and blue ligand). We first created two MDL files: "run_parameters" and "ligand_parameters". Information about the ligand is stored in the file "ligand_parameter", simulation run parameters are saved in the file "run_parameters". In this example, we will show how to include the files "run_parameters" and "ligand_parameter" into the simulation input file "unbound_diffusion.mdl".

MCell Code

```

/* Include files "run_parameters" and "ligand_parameters" */
INCLUDE_FILE = "run_parameters"
INCLUDE_FILE = "ligand_parameters"
/* Using parameters from the included file to define simulation */
TIME_STEP = dt
ITERATIONS = iterations
/* Using MDL function DEFINE_LIGAND to define red ligand */
DEFINE_LIGAND red-ligand { DIFFUSION_CONSTANT = red_ligand_diffusion_constant}
.....
/* run parameters for unbound_diffusion simulation */
  dt = 1.0e-6
  iterations = 300
/* Information about red and blue ligands */
  red_ligand_diffusion_constant = 6.0e-6
  blue_ligand_diffusion_constant = 5.0e-6
  number_red_ligand = 5.0e3
  number_blue_ligand = 2.0e3
  release_site_diameter = 0.03

```

Example 2

As we did in **Example 1**, this time we assign the file "ligand_parameters" to the variable "ligand_variable" in "run_parameters". Then we will INCLUDE_FILE "run_parameters", and INCLUDE_FILE "ligand_parameters" by using the variable "ligand_variable".

Notice: Even though the file "ligand_parameters" has been assigned to the variable of the included file "run_parameters", the MCell parser won't recognize "ligand_parameters" unless you include it directly.

MCell Code

```
/* Include files "run_parameters" and "ligand_parameters" */
INCLUDE_FILE = "run_parameters"
INCLUDE_FILE = ligand_variable
/* Using parameters from the included file to define simulation */
TIME_STEP = dt
ITERATIONS = iterations
/* Using MDL function DEFINE_LIGAND to define red ligand */
DEFINE_LIGAND red-ligand { DIFFUSION_CONSTANT = red_ligand_diffusion_constant}
.....
/* run parameters for unbound_diffusion simulation */
  dt = 1.0e-6
  iterations = 300
  ligand_variable = "ligand_parameters"
/* Information about red and blue ligands */
  red_ligand_diffusion_constant = 6.0e-6
  blue_ligand_diffusion_constant = 5.0e-6
  number_red_ligand = 5.0e3
  number_blue_ligand = 2.0e3
  release_site_diameter = 0.03
```

Section 3.4 Checkpointing in MCell

It is possible to insert check points to check or adjust a simulation before it terminates. This method is called *checkpointing*. Checkpointing is a commonly-used method to control a long simulation process; it is also a powerful and simple way to change run-time simulation parameters.

The checkpointing method is performed by inserting checkpoints into specific iterations of a simulation. When the simulation reaches a checkpoint, it will pause at that point. Checkpoints, therefore, can split a long simulation into several segments that can be executed sequentially. If you have M checkpoints, there will be M+1 simulation segments. You may use a Perl script file to operate the simulation automatically, the script file can load each simulation segment sequentially after each checkpoint. The three MDL keywords used to define the MCell checkpoints are:

- a. CHECKPOINT_ITERATIONS
- b. CHECKPOINT_INFILE
- c. CHECKPOINT_OUTFILE

Notice: The MDL command CHECKPOINT_ITERATIONS is a required argument for the checkpointing method, the other two are optional.

When a simulation pauses at a checkpoint you can resume the simulation from that checkpoint by assigning the name of the CHECKPOINT_OUTFILE (which is defined in the

previous simulation segment) to the MDL command CHECKPOINT_INFILE in the following segment. The necessary condition to continue the MCell simulation at a checkpoint is that the CHECKPOINT_OUTFILE is defined at the checkpoint, so that the simulator would store the previous segment to the checkpoint, and later use that information to resume the simulation. CHECKPOINT_OUTFILE will be read into the next segment by CHECKPOINT_INFILE. If CHECKPOINT_INFILE is not defined in the following segment, the simulation will start over from the beginning. Definition of checkpointing is given in Table 3.4-1.

Table 3.4-1 Definition of The Checkpoint

Table3.4-1

Format	Description
CHECKPOINT_ITERATIONS = N	<i>Use integer number N is the iteration number, the checkpoint is set at iteration step ITERATIONS = N.</i>
CHECKPOINT_OUTFILE = "checkpoint output filename"	<i>File name must be contained within double quotes.</i>
CHECKPOINT_INFILE = "checkpoint input filename"	<i>The checkpoint input file name must be contained within double quotes.</i>

CHECKPOINT_ITERATIONS

The MDL command CHECKPOINT_ITERATIONS specifies the location of the checkpoint within a simulation. The MCell simulation will automatically stop when it reaches the point of CHECKPOINT_ITERATIONS. The checkpoint method is only valid when CHECKPOINT_ITERATIONS is less than ITERATIONS.

CHECKPOINT_OUTFILE

The CHECKPOINT_OUTFILE saves the simulation results before the checkpoint. CHECKPOINT_OUTFILE should be defined after CHECKPOINT_ITERATIONS. After the checkpoint is reached, the primary simulation results will be saved to the file specified by CHECKPOINT_OUTFILE. These results may be read by the command CHECKPOINT_INFILE in the next simulation segment.

CHECKPOINT_INFILE

The simulation input file is split into sequential segments by checkpoints. In order to maintain consistency, the MDL command CHECKPOINT_INFILE should be specified in the second simulation segment. The output file name defined by CHECKPOINT_OUTFILE in the previous segment should be assigned to CHECKPOINT_INFILE in the next simulation input segment. The command CHECKPOINT_INFILE is an optional statement for the checkpointing method; however, if no CHECKPOINT_INFILE is specified, the simulation will begin from the beginning of the simulation, iteration 0.

Example 1

A MCell simulation has a total of 5000 iterations. We would like to put a checkpoint at iteration 1500. To do this, we edit two simulation files: "part1.mdl" and "part2.mdl". In the first simulation segment "part1.mdl", we assign 1500 to the MDL command CHECKPOINT_ITERATIONS, and save the primary simulation results into file "chkpt1". In the second simulation segment "part2.mdl", we assign the file name "chkpt1" to the MDL command "CHECKPOINT_INFILE", so the simulation will continue from iteration step 1501 to the end. If you run the simulation segments "part1.mdl" and "part2.mdl" sequentially, you will get a complete simulation.

MCell Code

```
/* The first simulation segment is executed up to the checkpoint ITERATIONS = 1500 */
TIME_STEP = 0.5e-6
ITERATIONS = 5000
/* Define the checkpoint and the output file name */
CHECKPOINT_ITERATIONS = 1500
CHECKPOINT_OUTFILE = "chkpt1"
...
/* The second simulation segment after the CHECKPOINT_ITERATIONS = 1500 */
TIME_STEP = 0.5e-6
ITERATIONS = 5000
/* Define the checkpoint input file to continue the simulation based on the primary results from
"chkpt1" */
CHECKPOINT_INFILE = "chkpt1"
...
```

Section 3.5 MCell Initialization and the Related MDL Parameters

There are three basic MDL commands that are used to set up the general spatial distribution of a MCell simulation. They are also the main parameters for initializing the MCell simulation. The MDL keywords for these are `RADIAL_SUBDIVISIONS` (number of subdivisions along any radial direction), `RADIAL_DIRECTIONS` (number of radial directions in a single octant of a unit sphere), and `EFFECTOR_GRID_DENSITY` (number of effector grids over the unit area surface). The MDL commands `RADIAL_SUBDIVISIONS` and `RADIAL_DIRECTIONS` initialize a MCell simulation process and are used to construct the two look-up tables for computations of molecular movements. Movements of molecules are decided by their step lengths and moving directions at each time step. For computational convenience, step lengths and moving directions of the molecules are defined in the polar coordinate system. They are computed from the look-up tables generated by `RADIAL_SUBDIVISIONS` and `RADIAL_DIRECTIONS`. Details are given in the following sections.

Section 3.5.1 MCell Initialization

A MCell simulation is a general Monte Carlo simulation. Like the Monte Carlo simulator, MCell focuses on molecular diffusions of short distances. The majority of movements of the simulated molecules is diffusion, which is an irregular movement that will result in a uniform distribution of the diffusing molecule from the initial non-uniform state. This type of movement is characterized as Brownian motion; therefore the movements of the molecules are simulated by an optimized Brownian algorithm

Because of the random characteristic of molecular diffusion, the step length and the moving direction of a molecule at each iteration time step are chosen randomly. The diffusion equation is solved first to find the key parameters of molecular diffusion. For computational conveniences, movements of molecules are expressed in polar coordinates. If we denote the concentration of the simulated molecules $C(r, t)$ as a function of the radial distance r , and time t , with a diffusion constant D_1 , then the diffusion equation expressed in polar coordinates will be an one dimensional equation:

$$\frac{(\partial C)}{(\partial t)} = D_i \frac{(\partial^2 C)}{(\partial^2 r)} \quad (3.1)$$

Equation (3.1) may be solved analytically. If the total amount of molecules in the whole system is M , then the concentration of the molecule $C(r, t)$ will be expressed as:

$$C(r, t) = e^{\frac{-r^2}{4D_i t}} \frac{M}{(4\pi D_i dt)^{3/2}} \quad (3.2)$$

Since the total number of molecules (expressed as N) in volume dV is the product of the volume and the concentration of the molecule in the system. Then N will be computed by:

$$N = C(r, t) \cdot dV \quad (3.3)$$

We define p_r as the fractional probability. It is the probability of obtaining a net radial displacement between distance r and $r+dr$ for each single diffusing molecule. The volume between distance r and $r+dr$ is dV , number of molecules in dV is N , so the the fractional probability p_r is N divided by the total number of molecules in the whole system, which is:

$$p_r = \frac{N}{M} \quad (3.4)$$

We use the fractional probability p_r to weigh the movements of molecules. After weighing, the mean radial displacement (l_r : the step length of the molecule diffusion.) of the molecule would be the expected value for a radial distance r . If we integrate p_r over r for each molecule at each time step to get the mean radial step length l_r , the computation will be too expensive. The MCell simulator optimizes this computation by introducing a dimensionless parameter s into the computation, s is defined as:

$$s = \frac{r}{\sqrt{4D_i dt}} \quad (3.5)$$

Substitute s into equation (3.4), we get:

$$p_r = \frac{4}{\sqrt{\pi}} s^2 e^{-s^2} ds \quad (3.6)$$

By integrating equation (3.6), we obtain a set of s with equal probability values. If we substitute s into equation (3.5), we will get a set of radial step lengths with equal probability values for any molecular diffusion. The radial step length is the product of s and the constant $(4D_i dt)^{1/2}$. You will have as many radial step lengths to choose from as the number of s values you have. Since all radial step lengths have the same probability, and diffusion of the molecules are random, we only need to pick one radial step length from the set randomly to move the molecules at each time step. MCell's optimization is to save all the equal probability s values

into a look-up table. During each MCell time step iteration, we only need to pick one s value randomly from the table, multiply s by the constant $(4D|dt)^{1/2}$ to get the radial step length. This MCell optimized algorithm is faster and easier compared to the traditional Brownian algorithm, which integrates the probability value for every time step to compute the radial step length.

The more s values we have, the more choices we have for the radial step length. Precision of the simulation therefore increases with the number of s values available. The trade-off is that more computer memory is needed to save the look-up table for s . You may assign the number of s by the MDL command **RADIAL_SUBDIVISIONS**. The default value of the RADIAL_SUBDIVISIONS is 1024 and the maximum value of s is 4096.

To determine the movement of a molecule we need to know its radial direction in addition to its radial step length. The direction of a molecule in Brownian motion is also random. MCell cuts a octant of a unit sphere into many equal divisions. Each division in the sphere represents one radial direction. Since the divisions all have equal probabilities, we just need to choose one randomly as the diffusion direction for the simulation. The divisions of the entire 3D space can be filled up by symmetry of the coordinates from one octant to the other seven. The user may define the number of possible directions by the MDL command RADIAL_DIRECTIONS. All equally possible directions are saved in another lookup table similar to that for the radial step length. The default number of radial directions is 131072 (2^{17}), which is adequate for most simulations. At each time step we only need to read two random values from the lookup tables for radial step lengths and radial directions of the molecules. The random number used to pick the table index are generated by the random number stream. The random number stream is a set of independent random numbers generated by the data encryption method from a [random seed number](#) (the random seed number is an integer number that you entered in your MCell execution command, it is an random integer number that is between 1 and 3000.). During initialization, the MCell simulator will read the random seed number and the RADIAL_SUBDIVISIONS and RADIAL_DIRECTIONS to set up the random stream and the two look-up tables for the MCell time step iterations.

Section 3.5.2 MCell Time Step Iterations

Knowing the radial step length and radial direction, molecules are moved from one point P1 to the next point P2 using ray tracing algorithms. Ray tracing was originally a technique used for rendering realistic images of 3D scenes in computer graphics. Imaginary, where rays are extended against the objects in the scene and the closest intersecting object is found.

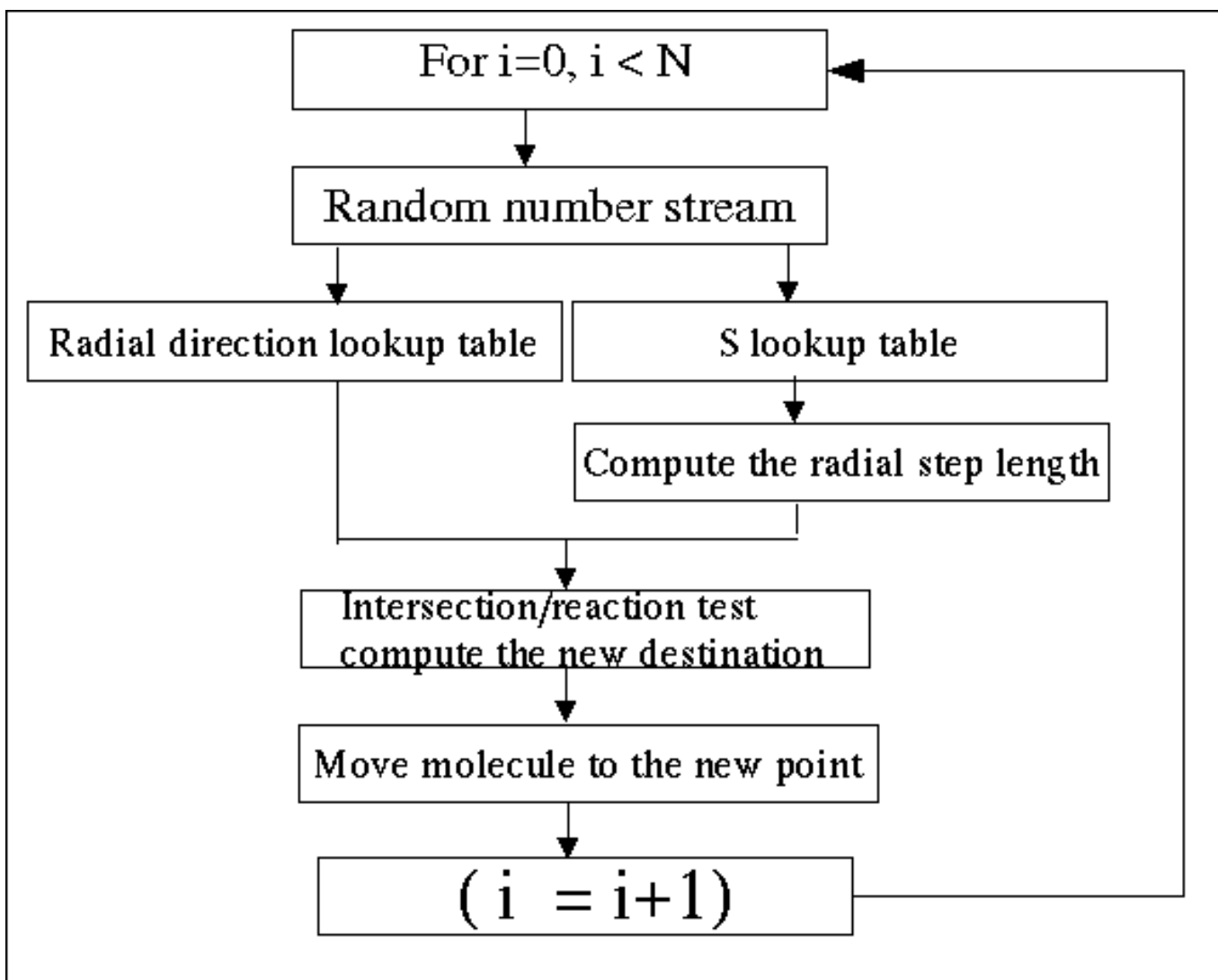
In each MCell time step, we imagine that a ray is extended from the original location of each molecule to the selected radial direction. The ray will go through a distance determined by the computed radial step length, then we determine the location of the point P2 and any intersection(s) of the ray with the objects on its way.

If the ray doesn't hit anything, it means that the molecule will not intersect with any object as a result of its movement, so we move the molecule from point P1 to point P2 along the picked radial direction with the computed step length.

In the case that the ray does intersect with an object, the first step is to judge what kind of object it is; we should know if it is a wall of a physical object or an other molecule. If it is a wall of a physical object, we need to check further if there is a molecule on the specific surface area. If there is no molecule, we then continue to check the surface permeability of the wall:

- A reflective wall would reflect the ray and a new destination would be computed;
- If the wall is transparent, it would let the molecule go through and finish its trip to the destination P2;
- The wall could also be absorptive, in which case the molecule will be absorbed by the wall, removing it from the molecule list.

If there is a molecule on the intersecting surface, MCell will check the status of that molecule to compute the probability of the reaction etc. After all of the possibilities are computed, the molecule will be moved to the new destination P2 ready for the next time step movements. The time step iterations of the MCell simulation is illustrated by the following flow chart:



Section 3.5.3 RADIAL_SUBDIVISIONS

RADIAL_SUBDIVISIONS is the number of divisions in an unit length along a radial direction. Therefore the RADIAL_SUBDIVISIONS are a set of step lengths with equal probability values. These equal probability values are saved in a lookup table during MCell initialization; they are used to compute the radial step length of the molecule during the time step iterations. The default number of RADIAL_SUBDIVISIONS is 1024, which provides enough precision for the general MCell simulations; larger RADIAL_SUBDIVISIONS value has a trade off of larger

computer memory requirement.

Table 3.5-1 The MDL Command RADIAL_SUBDIVISIONS

Table3.5-1

Format	RADIAL_SUBDIVISIONS = N
Annotation	N is an integer number, which is also the power of 2. $2^{10} \leq N \leq 2^{12}$
Unit	None
Default	2^{10}

Section 3.5.4 RADIAL_DIRECTIONS

RADIAL_DIRECTIONS is the number of the radial directions in a single octant of a unit sphere. They are the possible radial directions that we can use as the molecules' diffusing directions. The directions are saved into the second lookup table during MCell initialization. The default RADIAL_DIRECTIONS is 16384, which requires approximately 400 Kbytes computer memory for the computation, and the maximum value will need almost 3.2 Mbytes memory.

You have two options to define the command RADIAL_DIRECTIONS in the MCell simulation: The first way is to assign an integer number between 0 and 131072 to RADIAL_DIRECTIONS, but this integer number must be a power of 2; The second way is to assign the MDL keyword FULLY_RANDOM to RADIAL_DIRECTIONS. The MDL keyword FULLY_RANDOM means that the direction of the molecule movements will be computed randomly at each iteration time step instead of choosing a radial direction from the look up table. The simulation will slow down because of the extra computation at each time step.

Table 3.5-2 The MDL Command RADIAL_DIRECTIONS

Table3.5-2

Format	RADIAL_DIRECTIONS = N (or FULLY_RANDOM)
Annotation	N is an integer number which is a power of 2. $2^0 \leq N \leq 2^{17}$ FULLY_RANDOM will slow down the simulation by generating a new random number for the radial direction at each time step instead of choosing one from the lookup tables.
Unit	None
Default	2^{14}

Section 3.5.5 EFFECTOR_GRID_DENSITY

EFFECTOR_GRID_DENSITY is the number of the effector grids per unit area on the surface of a simulation object. The effector grid is the mesh element on which the effector sites could be placed. This value decides the number of the effector tiles that will be distributed on the surface of the simulation objects. The effector tiles are possible locations for the effector sites. The default EFFECTOR_GRID_DENSITY value is 1 mm^{-2} . The definition of the MDL command EFFECTOR_GRID_DENSITY is given in Table 3.5-3.

Table 3.5-3 The MDL Command EFFECTOR_GRID_DENSITY

Table3.5-3

Format	EFFECTOR_GRID_DENSITY = c
Annotation	c is a real number which is assigned to the EFFECTOR_GRID_DENSITY.
Unit	mm ⁻²
Default	1 mm ⁻²

Section 3.5.6 Spatial Partitioning

Spatial partitioning is the MCell run time optimization. Spatial partitions are transparent planes that the user places on the simulation object along the X, Y or Z axes to make spatial sub-volumes. To further understand the advantages of spatial partitions in an MCell simulation, we introduce the computational mechanism of the MCell simulator. All MCell simulations are carried on spatial objects which occupy certain volumes in space. From the computational point of view, the movements of the molecules inside the simulation objects should be calculated with the consideration of the whole volume, we call this volume the computational volume. If we have a larger number of molecules moving inside a bigger object, it means that we need to simulate more molecules inside a bigger computational volume at each time step and it will require more time and more memory. The larger the object is, the more computer memory needed. If we use spatial partitions to divide the original spatial volume into smaller sub-volumes, then the size of computational volume is reduced to that of each sub-volume; at the same time the number of the molecules inside each computational volume is reduced to those inside each sub-volume. As a result, the size of the computational object and the number of molecules decreases substantially; consequently the computation for the intersections between the molecules, as well as the computation for the movements of the molecules are reduced. Thus a well designed spatial partition will increase the simulation speed dramatically.

Spatial partitions can be created with any combination of the x,y, and z axial partitions. The design of the spatial partitions depends on the object structure and the distribution of the molecules inside the object. Spatial partitions are defined by the MDL command PARTITION_X, PARTITION_Y and PARTITION_Z, which are applied to the X, Y and Z axes separately. They are all represented by a numerical array, whose elements are the coordinates of the partitioning points in each direction. The numerical array elements are contained within a pair of square brackets []. Definitions of the PARTITION family are listed in Table 3.5-4. An example of spatial partitioning is given afterward.

Table 3.5-4 The Spatial Partitioning in The MCell

Table3.5-4

Format	PARTITION_X = [x ₁ , x ₂ , ... , x _m] PARTITION_Y = [y ₁ , y ₂ , ... , y _n] PARTITION_Z = [z ₁ , z ₂ , ... , z _l]
Annotation	The elements of the numerical arrays are the coordinates of the partitioning points in the Cartesian coordinate system. The number m, n, l are the number of the partitioning points along the X, Y, and Z axes.

Unit	μm
Default	No partitioning.

Example 1

Assume that we have a box object in the MCell simulation, the lower-front-left corner of the box is at (0,0,0), the upper-back-right corner is at (2,1,1). We insert 5 partitioning points along the x direction, the coordinates are: 0.25, 0.5, 0.55, 0.6, 0.75; one partitioning point each along the Y and Z directions, they are the middle points. The box is divided into 24 sub-volumes by this partitioning.

MCell Code

```
...  
PARTITION_X = [0.25, 0.5, 0.55, 0.6, 0.75]  
PARTITION_Y = [0.5]  
PARTITION_Z = [0.5]  
...
```

Section 3.6 Logical Objects Used in MCell

There are two types of defined objects in a MCell simulation: logical objects and physical objects. Logical objects are objects without definite or fixed physical locations in 3D space. Objects like ligands, ligand release patterns and chemical reaction mechanisms are logical objects defined in the MCell simulator. Physical objects are objects with definite locations in 3D space. In this section, we discuss the definition of logical objects in a MCell simulation.

The MCell simulation computes the behavior of each individual molecule. Two types of molecules are typically used in a simulation: small molecules called ligands, and large molecules called effectors. The effectors are usually added onto surfaces of physical objects, we will discuss it later in the [surface modifiers](#) section of this chapter.

The small molecules (such as ions, amino acids, Acetylcholine, ATP and adenosine, sugars, peptide, fats, neuro-transmitter, hormones, etc.) are defined as ligands. Ligands diffuse with their concentration gradients, react with other molecules under certain conditions, and may bind and unbind with the effectors as various conditions are satisfied. The movement of ligands are simulated by the Brownian random walk algorithm. The ligand is defined by the MDL function `DEFINE_MOLECULE` with a given name. The defined ligand is released from a physical object: ligand release site (defined by the MDL function `SPHERICAL_RELEASE_SITE`) with a certain temporal pattern, which is defined by the function `DEFINE_RELEASE_PATTERN`. The ligand release site is the source of the ligand, the function `SPHERICAL_RELEASE_SITE` specifies the physical location of the ligand source, the size of the release site, the number of ligands released from this site, as well as the ligand releasing pattern from this site. The optional function `DEFINE_RELEASE_PATTERN` is used to define the temporal pattern of ligand release. If release patterns are not specified, the MCell simulator will release all the ligands at the moment when the simulation starts.

A MDL function consists of a function name (capitalized MDL keyword) and a function body (contained within the brace `{}`). Each MDL function contains both required and optional elements.

Section 3.6.1 Defining A Ligand Molecule

The function `DEFINE_MOLECULE` defines one type of ligand molecule with a specified name. This MDL function contains the element `DIFFUSION_CONSTANT` (the required element) and the element `REFERENCE_DIFFUSION_CONSTANT` (the optional element). After the ligand is defined, it may be used in a diffusion model, or in the chemical reaction definition and the surface permeability definition of the simulation by referencing the name of the ligand. The usage of the function `DEFINE_MOLECULE` is shown in Table 3.6-1.

Table 3.6-1 Definition of The Molecule

Table3.6-1

Format	<code>DEFINE_MOLECULE ligand_name { REFERENCE_DIFFUSION_CONSTANT = c1 DIFFUSION_CONSTANT = c2 CHARGE = c3 }</code>
Annotation	Text string <i>ligand_name</i> is the user defined name for ligand. Numerical value <i>c1</i> , <i>c2</i> and <i>c3</i> are real numbers. The MDL keyword <code>DIFFUSION_CONSTANT</code> is the required argument in this function. The MDL keyword <code>REFERENCE_DIFFUSION_CONSTANT</code> and <code>CHARGE</code> are optional arguments in this function.
Unit	<code>REFERENCE_DIFFUSION_CONSTANT</code> : [cm^2s^{-1}] <code>DIFFUSION_CONSTANT</code> : [cm^2s^{-1}] <code>CHARGE</code> : [C]
Default	None.

DIFFUSION_CONSTANT

The diffusion constant of the defined ligand. This parameter measures the rate at which particles can diffuse through a medium; it depends on the substance, the medium, and the temperature. The diffusion constant has a dimension of area/time. In a MCell simulation, this value is used to calculate the radial step length of the ligand.

REFERENCE_DIFFUSION_CONSTANT

The reference diffusion constant. This value is used to calculate the probability of ligand binding.

CHARGE

Charge is the electrical charge that a ligand carries. The movements of the ligand form a flow of electrical charges, called current flow. The parameter `CHARGE` is particularly useful for the simulation of ion channel.

Example 1

In this example we define the ligand "ACh" with a diffusion constant of $2.1 \times 10^{-6} \text{ cm}^2\text{s}^{-1}$. The ligand "ACh2" has the same diffusion constant as ligand "ACh", the reference diffusion constant of "ACh2" is $1.9 \times 10^{-6} \text{ cm}^2\text{s}^{-1}$.

MCell Code

```
DEFINE_MOLECULE ACh {
```

```

DIFFUSION_CONSTANT = 2.1x10-6
}
DEFINE_MOLECULE ACh {
  REFERENCE_DIFFUSION_CONSTANT = 1.9x10-6
  DIFFUSION_CONSTANT = 2.1x10-6
}

```

Section 3.6.2 The Ligand Release Pattern

Sometimes you may want to define a temporal pattern to control the ligand release process. The release pattern is a series of discrete events, all of which are defined using the simulation start time $t = 0$ as reference. Each release is a pulse as illustrated in Figure 3.2.

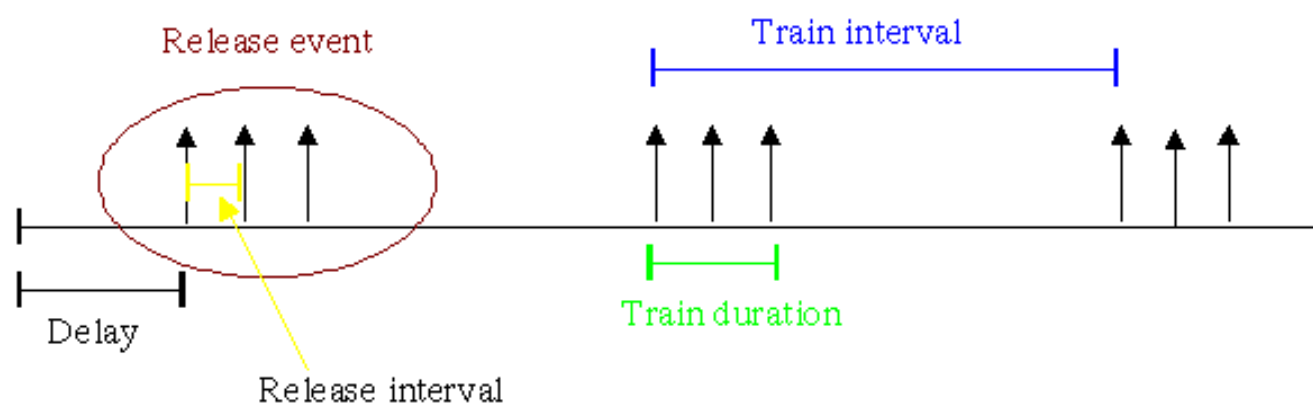


Figure 3.2 Timing Pattern of the ligand releasing

The temporal pattern of ligand release shown in Figure 3.2 is typically defined as "a train". The pattern of train distribution is the pattern of the ligand release. The release time depends on the number of trains, train duration, train interval and delay time. Each train contains one or more release events (pulses). The release pattern depends on the train duration, release interval and the train interval. The ligand release pattern is defined by the MDL function `DEFINE_RELEASE_PATTERN` with a user-given name. The defined release pattern will be assigned to the MDL command `RELEASE_PATTERN` in the function `SPHERICAL_RELEASE_SITE` by the defined pattern name. Table 3.6-2 shows the format of the function `DEFINE_RELEASE_PATTERN`.

Table 3.6-2 The MDL Function `DEFINE_RELEASE_PATTERN`

Table3.6-2

Format	<code>DEFINE_RELEASE_PATTERN</code> <i>pattern_name</i> { <code>DELAY = c1</code> <code>RELEASE_INTERVAL = c2</code> <code>TRAIN_INTERVAL = c3</code> <code>TRAIN_DURATION = c4</code> <code>NUMBER_OF_TRAINS = n (or UNLIMITED)</code> <code>}</code>
Annotation	Where <i>c1</i> , <i>c2</i> , <i>c3</i> , <i>c4</i> are real numbers, <i>n</i> is an integer number.
Unit	second(s)
Default	All the ligands will be released in 1 second from the moment when the simulation start, which is:

```
DELAY = 0;  
RELEASE_INTERVAL = 2;  
TRAIN_INTERVAL = 1;  
TRAIN_DURATION = 1;  
NUMBER_OF_TRAINS = 1.
```

DELAY

The time interval between the simulation start time $t=0$ and the first ligand release event in the ligand release process.

RELEASE_INTERVAL

The time interval between two consecutive release events (pulses) within a release train. It is the inverse of the frequency of the release event.

TRAIN_INTERVAL

The time interval between the beginning of one train to the beginning of next. It is the inverse of the train frequency.

TRAIN_DURATION

The duration of each train. It is the product of the number of events in a train and the `RELEASE_INTERVAL` of the train.

NUMBER_OF_TRAINS

The total number of trains in the defined release pattern. If the MDL keyword `UNLIMITED` is assigned to `NUMBER_OF_TRAINS`, then the train will last until the end of the simulation.

Example 1

Define a ligand release pattern named "star". There are 3 trains in this release. The first release event will happen at the moment $t=0.5$ seconds, the release interval is 1 second, the train will last 5 seconds. Train interval is 8 seconds, which means that the ligand will be released in 24.5 seconds by three trains.

MCell Code

```
DEFINE_RELEASE_PATTERN star {  
  DELAY = 0.5  
  RELEASE_INTERVAL = 1  
  TRAIN_INTERVAL = 8  
  TRAIN_DURATION = 5  
  NUMBER_OF_TRAINS = 3  
}
```

Section 3.6.3 The Chemical Reaction Mechanisms

In a MCell simulation, a chemical reaction is defined by the MDL function `DEFINE_REACTION` with a given name. Both unimolecular transitions and bimolecular associations can be expressed by this function. In a unimolecular reaction, a single reactant molecule rearranges to a single product molecule, or a single reactant molecule splits apart to give two or more product molecules. A biomolecular reaction involves two reactant molecules and occurs through collisions. Single path and multi-path reactions can also be illustrated easily with the `DEFINE_REACTION` function. The multi-path reaction would be divided into sequentially adjacent steps, the function `DEFINE_REACTION` could then be used to represent each step,

which would involve different reaction states. Table 3.6-3 lists the chemical reaction expression in MCell.

Table 3.6-3 Chemical Reactions in MCell

Table 3.6-3

Format	<pre> DEFINE_REACTION <i>reaction_name</i> { state_name[>adjacent_state_name {rate_constant}] ... state_name[>adjacent_state_name {rate_constant: % <i>ligand_name, polarity</i>}] ... REFERENCE_STATE <i>state_name</i> { <i>ligand_name</i> NUMBER_BOUND = <i>n</i> ...} } </pre>
Annotation	<ol style="list-style-type: none"> 1. The symbol % represents one of the 6 ligand interaction operators: binding, transporting, unbinding, degradation, producing and producing_poisson; 2. The keyword <i>polarity</i> is the polarity value of the indicated ligand, it depends on the ligand interaction operators; 3. The function REFERENCE_STATE is an optional argument for this function; 4. The MDL keyword NUMBER_BOUND is the required argument of the function REFERENCE_STATE.
Unit	rate_constant: [s ⁻¹] ([M ⁻¹ s ⁻¹] for binding case).
Default	None.

We give two examples of the DEFINE_REACTION function: the single path reaction and the multi-path reaction, shown in Figures 3.3 and 3.4, respectively. In Figure 3.3, we give a single path chemical reaction, in which state A will react with (bind) state E when the reaction constant k1 is reached, this reaction will produce a new state--AE; State E and A will unbind and be released from state AE when the reaction constant k2 is satisfied.

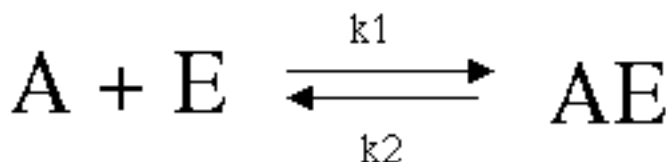


Figure 3.3 Single path chemical reaction

We can write this reaction with the MDL function DEFINE_REACTION with two simple statements as in the following example.

Example 1

The reaction shown in Figure 3.3 can be expressed by the function DEFINE_REACTION according to the format defined in the Table 3.6-3. The MDL expression of this reaction is:

MCell Code

```

DEFINE_REACTION single_path {
  A[>AE {k1: +E, BOTH_POLE}]
  AE[>A {k2: -E, EITHER_POLE}]
}.

```

We express a multiple path chemical reaction path by path, and each path is enclosed in a separate set of square brackets. For the reaction shown in the Figure 3.4, we have four paths:

- First, state A binds with state R^0 as the reaction constant $k11$ is achieved; this reaction will produce the state AR^1 ;
- Second, state AR^1 will unbind and release both states R^0 and A as the reaction constant $k12$ is satisfied;
- Third, state A binds with state R^0 when the reaction constant $k22$ is reached, and state AR^2 will be the new state that is formed after the reaction;
- Fourth, state AR^2 unbind and release both states R^0 and A as the reaction constant $k21$ is satisfied.

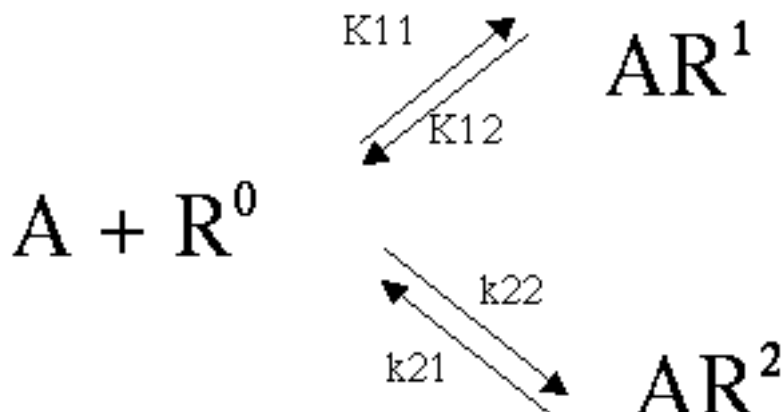


Figure 3.4 The Multiple path chemical reaction

The MDL expression of this reaction is given in the following example:

Example 2

The multi-path chemical reaction may be expressed path by path. When two or more reactions will happen between two states under different conditions (such as the first and the third paths in the Figure 3.4), we can list them in the same line as the codes listed below.

MCell Code

```
DEFINE_REACTION multi_path {
  A[>AR1 {k11: +R0, POSITIVE_POLE}][>AR2 {k22: +R0, NEGATIVE_POLE}]
  AR1[>A {k12: -R0, EITHER_POLE}]
  AR2[>A {k21: -R0, EITHER_POLE}]
}
```

Section 3.6.3.1 The Operators of Ligand Interactions

The ligand interaction operators are the symbols used to tell the MCell simulator what kind of interaction will happen between the defined effector state and the selected ligand molecule. The symbol % that appeared in the function DEFINE_REACTION represents these 6 ligand operators (six kinds of the ligand interactions):

Operator "+"

Operator "+" is the ligand binding operator; In the function DEFINE_REACTION, when this operator appears, the ligand behind it will bind with the defined effector, under the proper polarity conditions.

Operator "-"

Operator "-" is the ligand unbinding operator; In the function DEFINE_REACTION, when this operator appears, the ligand behind it will unbind with the defined effector, under the proper polarity conditions.

Operator "~"

Operator "~" is the ligand transport operator. The ligand which follows this operator will be transported through the medium. An example would be molecular movements through a transmembrane channel.

Operator "#"

Operator "#" is the ligand destruction operator; In the function DEFINE_REACTION, when this operator appears, the ligand behind it will be degraded by the defined effector when the chemical rate constant is satisfied.

Operator "*"

Operator "*" is the operator for the production of the diffusing ligand.

Operator "@"

Operator "@" is another ligand producing operator, the ligand production of this operator has a Poisson distribution, so we call it Producing Poisson.

Besides ligand interaction operators, you also need the ligand polarity values to define the chemical reaction. The MDL keyword **polarity** is used to define the binding and unbinding between the ligands and the effectors in the function DEFINE_REACTION. The value of *polarity* depends on the ligand interaction operators. Table 3.6-4 lists the *polarity* values for different ligand interaction operators.

Table 3.6-4 The *polarity* of A Ligand

Table3.6-4

Ligand Interaction	<i>polarity</i> Value
Binding (+ case)	<i>polarity</i> = POSITIVE_POLE , NEGATIVE_POLE , BOTH_POLE
Unbinding (- case)	<i>polarity</i> = POSITIVE_POLE, NEGATIVE_POLE, EITHER_POLE , BOTH_POLE
Transporting (~ case)	<i>polarity</i> = POSITIVE_POLE , NEGATIVE_POLE , BOTH_POLE
Destruction (# case)	<i>polarity</i> value is ignored in this case.
Production (* case)	<i>polarity</i> = POSITIVE_POLE, NEGATIVE_POLE, EITHER_POLE
Producing Poisson (@ case)	<i>polarity</i> = POSITIVE_POLE, NEGATIVE_POLE, EITHER_POLE

Section 3.6.3.2 The Function REFERENCE_STATE

The MDL function REFERENCE_STATE is an argument of the function DEFINE_REACTION. The function REFERENCE_STATE specifies the number of ligands (defined by the MDL command NUMBER_BOUND) that are bound to the effector state defined in the chemical reaction mechanism. Both the specified effector state name and the related ligand name should be given in this definition. If we denote the total number of ligands bound with an integer

number n , then the function REFERENCE_STATE has a format:

```
REFERENCE_STATE state_name {  
    ligand_name  
    NUMBER_BOUND = n  
}
```

Section 3.7 Templates of The Physical Object

Physical objects are objects with definite physical locations in 3D space. Physical objects may be objects created from 3D reconstruction, objects designed by computer graphics tools, or objects defined by the MCell physical object definition. They are designed to hold or locate the logical objects in MCell simulations. All simulation events occur either on the surface of physical objects or around them. Three basic physical objects defined in the MCell are:

- a. The ligand release site ;
- b. The box object;
- c. The polygon list object;

You may group the basic physical objects hierarchically into a meta object template, or instantiate the physical objects as components of the MCell simulation. You can also apply [geometrical transformations](#) to all physical objects, and modify surfaces of specified physical objects by removing certain surface elements, defining the surface permeability with respect to some ligand, or adding effectors on surface elements. All these applications are described by the term *surface_modifier*. If a logical object is located on or inside a particular physical object, we define it within the physical object template.

Section 3.7.1 The Ligand Release Site

A ligand release site is the physical location of a ligand source in 3D space. The MDL function SPHERICAL_RELEASE_SITE is used to define a ligand release site. This function provides the MCell simulator information about the location and size of the site, the number of the ligands that are released, the name of the released ligand, and the temporal pattern of the ligand release process. The ligand release source is modelled as a sphere by the MDL function SPHERICAL_RELEASE_SITE. You can create a point source site by assigning zero to the diameter of the ligand source.

The reason for this simplification is that the ligand source is usually very small in the physical world, and most of them are round shaped. So for computational convenience, we neglect the shape of the ligand site and regard it as a small sphere. ***But a sphere is not the only shape that may be used for the ligand source, you may define an arbitrarily shaped ligand source too.*** To define an arbitrarily shaped ligand source, you first define a point source using the MDL function SPHERICAL_RELEASE_SITE and put it inside the desired shaped physical object. This physical object will be the ligand source. The arbitrarily shaped physical object may be defined by the MDL physical object definitions and other 3D reconstruction and graphical tools.

Section 3.7.1.1 SPHERICAL_RELEASE_SITE

The MDL function SPHERICAL_RELEASE_SITE defines the ligand release site. Since we

regard it as a spherical source, we only need to provide the origin and the diameter of the sphere in the function SPHERICAL_RELEASE_SITE to locate this source. The ligand source site should have an uniform distribution of the ligand molecules. Geometric transformations, which include object translation, rotation, and object scaling, can be applied to the spherical release site as well as all the other physical objects. The Format of the function SPHERICAL_RELEASE_SITE is shown in Table 3.7-1.

Table 3.7-1 The MDL Function SPHERICAL_RELEASE_SITE

Table3.7-1

Format	<pre> <i>site_name</i> SPHERICAL_RELEASE_SITE { LOCATION = [x, y, z] MOLECULE = molecule_name <i>release_number_specs</i> RELEASE_PROBABILITY = c1 SITE_DIAMETER = c2 RELEASE_PATTERN = pattern_name <i>geometric_transformation</i> } </pre>
Annotation	<p>The numbers <i>c1</i> and <i>c2</i> are real numbers; <i>release_number_specs</i> includes three different specifications to define the number of molecules released: A fixed number; Volume dependent number; Gaussian distribution number. RELEASE_PATTERN is an optional statement in this function; <i>geometric_transformation</i> represents the geometrical transformation applied to the physical object, it is an optional statement.</p>
Unit	<p>LOCATION: μm SITE_DIAMETER: μm</p>
Default	<p>RELEASE_PROBABILITY = 1; If no spherical release site is specified, no ligand will be released.</p>

LOCATION

The x-y-z coordinates of the sphere origin defined in the Cartesian Coordinate system.

MOLECULE

Name of the defined molecule from the function [DEFINE MOLECULE](#), the ligand with the name assigned in this specification will be released from the defined source site.

release_number_specs

The total number of the selected ligand molecules that will be released from this source. There are three different ways to define the ligand release number.

1. NUMBER_TO_RELEASE = n
2. **Volume dependent release number**
VOLUME_DEPENDENT_RELEASE_NUMBER {
MEAN_DIAMETER = c

```
STANDARD_DEVIATION = c
CONCENTRATION = c
}
```

which will compute the number to release based upon a fictitious mean vesicle diameter, standard deviation of the diameter and filling concentration for the vesicle. The diameter will be normally distributed but the vesicle volume and therefore number in the vesicle will be non-normally distributed. *c* is a real number. Note that the MEAN_DIAMETER parameter refers to a fictitious diameter and has nothing to do with the SITE_DIAMETER parameter for the SPHERICAL_RELEASE_SITE.

3. **Gaussian release number**

```
GAUSSIAN_RELEASE_NUMBER {
MEAN_NUMBER = n
STANDARD_DEVIATION = n
}
```

which will cause the release of a normally distributed number of molecules; *n* is an integer number.

RELEASE_PROBABILITY

The probability of the ligand being released from the ligand source. The default value of this parameter is 1, which means that the ligand will be released by default from the defined ligand release site. You may use this value to limit the number of ligand to be released.

SITE_DIAMETER

The diameter of the spherical ligand source.

RELEASE_PATTERN

The name of the defined ligand release pattern (it will be defined by the MDL function [DEFINE_RELEASE_PATTERN](#)). It is an optional argument in the function SPHERICAL_RELEASE_SITE.

[geometric transformation](#)

This MDL symbol represents one of the three geometrical transformations the user could apply to physical objects. The three geometrical transformations are translation, rotation, and scaling. All physical objects can be transformed into other objects. A Detailed description of geometric transformation will be given in section 3.9.

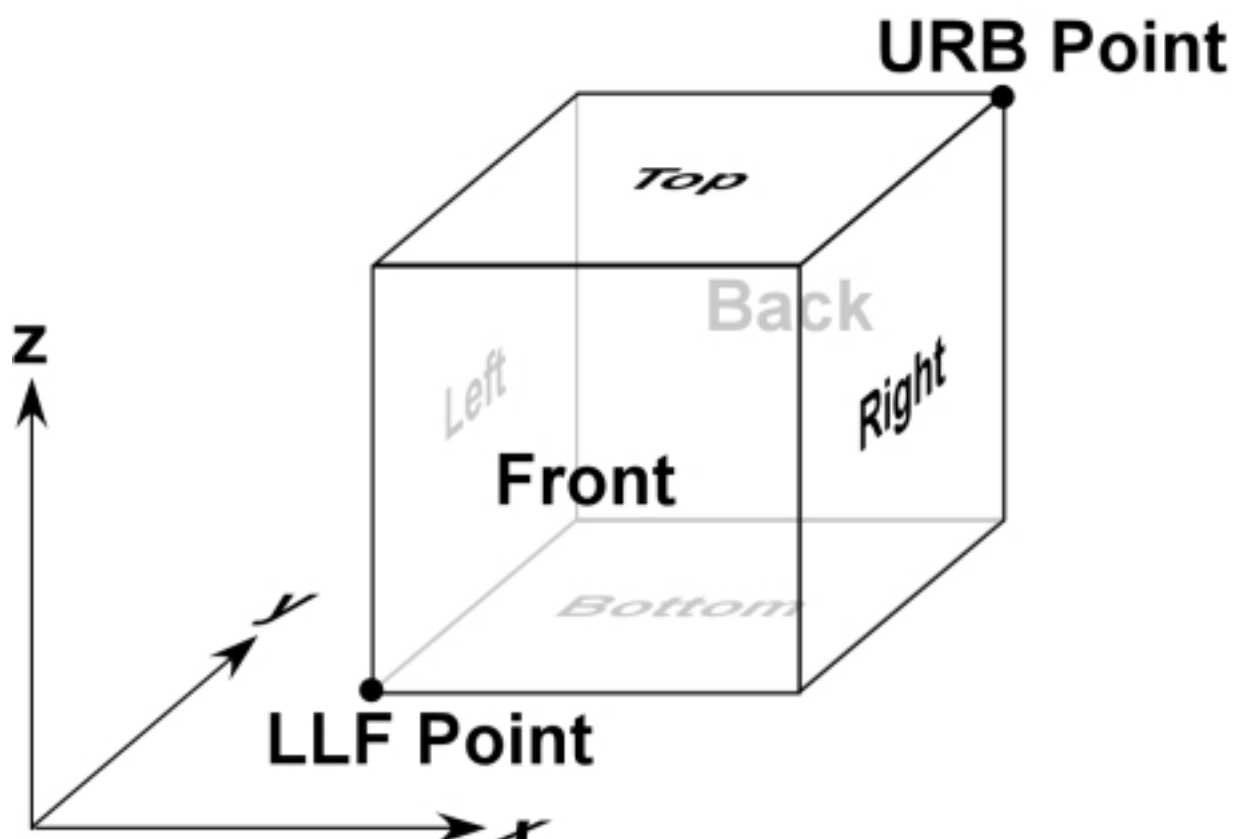
Section 3.7.1.2 Arbitrary Shaped Ligand Release Site

We have mentioned that the user can define a ligand release site with an arbitrary shape. You can follow the following steps to define an arbitrarily shaped ligand release site in your simulation.

1. First, construct a physical object with the desired shape. The physical object can be defined from 3D reconstruction, built by graphical tools, or built by the MDL physical object definitions.
2. The second step is to set a point ligand source using the function SPHERICAL_RELEASE_SITE, You can do this by setting the keyword SITE_DIAMETER in the function SPHERICAL_RELEASE_SITE equal to zero. **The most important step is to locate the point source inside the designed ligand release site--the physical object that you defined in the first step.**
3. The third step is to define all the surface elements of the ligand release site as reflective, so that the ligand can diffuse and mix well inside. After a certain number of iterations the distribution of ligands inside the site will be uniform.
4. Set a checkpoint in the simulation such that when the checkpoint is reached, the ligands inside the release site would have been distributed homogeneously. After the checkpoint, change the permeability of the release site surface elements to transparent, so when the simulation start again the ligands can escape from your designed ligand source.

Section 3.7.2 The Box Object

The box object is a hexahedron constructed with the constraint that each face of the hexahedron is perpendicular to one of the x-y-z axes. We define the box object by specifying two vital points: the lower-left-front (LLF) corner and the upper-right-back (URB) corner (as point P1: lower-left-front point (LLF) and P2: upper-right-back (URB) point in the Figure 3.5). The box object is defined by the right-hand rule in the Cartesian coordinate system. To define the physical objects, we indicate whether the object is a closed object or an opened object.



Right-handed coordinate system

Figure 3.5 The Box object defined in the MCell Simulation. The coordinates of the points LLF point and URB point should be given by user to define the box object in the MCell simulation.

Figure 3.5 illustrates a box object defined in MCell, the two corners P1 and P2 shown in the figure are the required points for box definition. We could find out the size and location of the box object by the symmetry of the box from the coordinates of P1 and P2. The box object is defined by the MDL function BOX. The format of the BOX function is given in Table 3.7-2.

Table 3.7-2 The MCell Box Object

Table3.7-2

Format	<i>box_name</i> BOX { CORNERS = [x1, y1, z1], [x2, y2, z2] FULLY_CLOSED = <i>boolean</i> <i>surface_modifiers</i> <i>geometric_transformation</i> }
Annotation	<i>boolean</i> value is: YES, NO, TRUE, FALSE; <i>surface_modifiers</i> is an optional element in this function; <i>geometric_transformations</i> is an optional element in this function; You can define the logical object inside this template when the logical object is located on or inside the defined box.
Unit	CORNERS: μm
Default	None

CORNERS

The lower-left-front point (p1 in the Figure 3.5) and the upper-right-back point (P2 in the Figure 3.5) are the corner points of a box object which we use to define the location of it in a MCell simulation. The coordinates of the points P1 and P2 are the values assigned to the MDL keyword CORNERS. The x-y-z coordinates of the points P1 and P2 should be contained within the square bracket [], and separated by a comma.

FULLY_CLOSED

The MDL keyword FULLY_CLOSED is used to describe the closed or open status of the defined object. Boolean values YES, NO, TRUE, and FALSE will be assigned to the keyword FULLY_CLOSED. The boolean value YES and TRUE have the same meaning, which is that the defined object is a closed object, boolean value NO and FALSE mean that the defined object is an open object.

surface_modifiers

surface_modifiers is a symbol used in the MDL to represent the surface modifications applied to physical objects. The surface modifications that you can apply to the simulation objects are: defining the *surface_permeability* of the surface element, removing the specified surface element (wall or face) from a box or polygon list object (defined by the MDL keyword REMOVE_ELEMENT), tiling a surface with the stationary effector molecules (defined by the MDL function ADD_EFFECTOR), adding active zones onto physical objects, or adding color to the surface elements. Please see [section 3.8](#) for a detailed description of the surface modifiers.

Example 1

Create a fully closed box with the name "box1". The center of the box is at the the origin (0, 0, 0), and the length, width, height of the box all equal to 0.2 μm .

MCell Code

```
/* The two vital points for the box object definition can be computed from the origin and the size
of the box. We can calculate the front-lower-left point as P1(-0.1, -0.1, -0.1) and
back-upper-right point as P2(0.1, 0.1, 0.1) from the known parameters.*/
box1 BOX {
  CORNERS = [-0.1, -0.1, -0.1], [0.1, 0.1, 0.1]
  FULLY_CLOSED = YES
}
```

Example 2

Create a fully closed box with the name as "box2". The location and size of "box2" are the same as the "box1" which we defined in the previous example. The effector AChR is added on the bottom wall of the box object, with a density of $8935 \mu\text{m}^{-2}$, and the orientation of the effector is on the back side of the surface element. We remove the top side of the box. The left and right surface elements of the box are defined as transparent surfaces to the ligand ACh.

MCell Code

```

box2 BOX {
  CORNERS = [-0.1, -0.1, -0.1], [0.1, 0.1, 0.1]
  FULLY_CLOSED = YES
  REMOVE_ELEMENT = TOP
  /* Add the effector AChR to the bottom of the box. */
  ADD_EFFECTOR {
    STATE = AChR
    DENSITY = 8935
    ELEMENT = BOTTOM
    POLE_ORIENTATION = POSITIVE_BACK
  }
  /* Define the surface permeability of the left and right surface elements to the ligand ACh.
  */
  TRANSPARENT {
    LIGAND = ACh
    ELEMENT = LEFT
  }
  TRANSPARENT {
    LIGAND = ACh
    ELEMENT = RIGHT
  }
}

```

Section 3.7.3 The Polygon List Object

Any physical object can be presented as a polygon list object. The polygon list object may be obtained from 3D reconstruction or from the MDL function `POLYGON_LIST`. Each surface element of the polygon list object is a planar polygon connected by several vertices. If you use the MDL function `POLYGON_LIST` to define the polygon list object you should specify all the unique vertices in each planar polygon and the connections between each vertex. The unique vertices of the polygon are defined by the MDL function `VERTEX_LIST`, which is a sub-function of the function `POLYGON_LIST`. The coordinates of all the unique vertices should be given in a specific order such that they are arranged to make the normal vector of the element match the right hand rule. The indices of the vertices are used to specify the connection between the them. The connection of the elements are defined by the function **ELEMENT_CONNECTIONS** inside the `POLYGON_LIST`.

The x-y-z vertex coordinates that we will use in the definition of the polygon list object are

defined in the right-hand Cartesian coordinates system. The sequential numbers appeared in the function VERTEX_LIST are the index number of each vertex, and we will use these numbers to define the connections between each vertex by the function ELEMENT_CONNECTIONS. The definition and format of the MDL function POLYGON_LIST are shown in Table 3.7-3.

Table 3.7-3 The MDL Function POLYGON_LIST

Table3.7-3

Format	<pre> <i>polygon_name</i> POLYGON_LIST { VERTEX_LIST { [x₁, y₁, z₁] ... [x_n, y_n, z_n] } ELEMENT_CONNECTIONS { <i>connection_specifier</i> } FULLY_CLOSED = <i>boolean</i> <i>surface_modifiers</i> <i>geometric_transformation</i> } </pre>
Annotation	<p>The function VERTEX_LIST is defined by the x-y-z coordinates of the vertices by order;</p> <p>Array [x_n, y_n, z_n] is the coordinates of the <i>n</i>th vertex;</p> <p><i>connection_specifier</i> is a numerical array to define the connectivity of the vertex list. The value of this array is the 0-based indices of the vertices in the vertex list which comprise the defined element;</p> <p><i>boolean</i> is one of the following values: YES, NO, TRUE, FALSE; The FULLY_CLOSED function is reserved for future versions of MCell; It is currently ignored by the parser when the simulation initiates.</p> <p><i>surface_modifiers</i> and <i>geometric_transformation</i> are optional arguments for this function.</p>
Unit	µm
Default	None

Example 1

A fully closed polygon object "small_polygon" with 5 unique vertices, and 6 planar polygons is defined by the MDL function POLYGON_LIST. The coordinates of the five vertices are (0, 0, 0.2), (0.1, 0.1, -0.1), (0.1, -0.1, -0.1), (-0.1, -0.1, -0.1) and (-0.1, 0.1, -0.1). All the elements are transparent to the ligand Ach. The effector AchR is added on the 4th and 5th element of this polygon with the orientation on the face side of the surface element.

MCell Code

```

small_polygon POLYGON_LIST {
  /*The coordinates of the 5 vertices are listed sequentially. */
  VERTEX_LIST {
    [0, 0, 0.2] /* Coordinates of the vertex #0 */
    [0.1, 0.1, -0.1] /*Vertex #1 */
    [0.1, -0.1, -0.1] /*Vertex #2 */
    [-0.1, -0.1, -0.1] /*Vertex #3 */
    [-0.1, 0.1, -0.1] /*Vertex #4 */

```

```

}
/* Define the connection of each polygon element. */
ELEMENT_CONNECTIONS {
  /*Surface element # 0. This element is consisted with vertex 0, vertex 2, and vertex
  1. */
  [0, 2, 1]
  [0, 3, 2] /* Surface element #1. */
  [0, 4, 3] /* Surface element #2. */
  [0, 1, 4] /* Surface element #3. */
  [2, 3, 4] /* Surface element #4. */
  [1, 2, 4] /* Surface element #5. */
}
FULLY_CLOSED = YES
TRANSPARENT {
  LIGAND = ACh
  ELEMENT = ALL_ELEMENTS
}
ADD_EFFECTOR {
  STATE = AChR
  DENSITY = 8395
  ELEMENT = 4
  POLE_ORIENTATION = POSITIVE_FRONT
}
ADD_EFFECTOR {
  STATE = AChR
  DENSITY = 8395
  ELEMENT = 5
  POLE_ORIENTATION = POSITIVE_FRONT
}
}

```

Section 3.7.4 The Meta Object

A convenient way to aggregate the physical objects in MCell is using the meta object template. The meta object is defined by the MDL function OBJECT, it may include both on-site newly created physical objects and pre-defined physical objects. You may define physical objects (ligand release sites, box objects, and polygon list objects) inside meta objects as elements of meta objects, or copy or transform the physical objects (which are pre-defined outside of the meta object) into meta objects. Like all the other physical objects in the MCell, you could apply *geometric_transformation* to each elements inside the meta object.

The method to define a meta object in the MCell is given in Table 3.7-4. The function OBJECT has multiple applications in the MDL:

- The function OBJECT may be used to define meta objects;
- The function OBJECT may be used to copy a pre-defined physical object into meta object;
- The function OBJECT may work together with the MDL keyword INSTANTIATE to instantiate physical objects as simulation components.

Table 3.7-4 Definition of The Meta Object

Table3.7-4

Format	<pre> object_name OBJECT { new_object_name OBJECT predefined_object_name { geometric_transformation } </pre>
--------	--

	<i>new object_template</i> <i>geometric_transformation</i> }
Annotation	The templates of the meta OBJECT may be new physical object definitions or copies of pre-defined objects; <i>geometric_transformation</i> is an optional operation.
Unit	None
Default	None

Example 1

There is a box object named "origin_box". We will copy the object "origin_box" into the meta object "group1" as "box1". It will be translated 1.0 um along the X-axis and be saved as "box2". It will be also rotated 45° around the axis [1, 1, 1], and be saved as the object "box3" inside "group1". We will also create the spherical ligand release site "red_source" and the box "box4" inside this meta object. The spherical release site is located at the origin, the diameter of the site is 0.03 um, 1000 'red_ligands' will be released from this site. The object "box4" with two corners P1 at (-1.5, -1.5, -1.5) and P2 at (1.5, 1.5, 1.5) is fully closed, the front face of "box4" is absorptive to the "red_ligand".

MCell Code

```

/* Meta object group1 definition */
group1 OBJECT {
  /* copy pre-defined object "origin_box" into "box1" */
  box1 OBJECT origin_box {
    /*Translate "origin_box" 1.0um as "box2" */
    box2 OBJECT origin_box {
      TRANSLATE = [1.0, 0, 0]
    }
    /*Rotate "origin_box" 45° around axis [1, 1, 1] as "box3" */
    box3 OBJECT origin_box {
      ROTATE = [1, 1, 1], 45
    }
    /*Define new ligand release site "red_source" */
    red_source SPHERICAL_RELEASE_SITE {
      LOCATION = [0, 0, 0]
      LIGAND = red_ligand
      NUMBER_TO_RELEASE = 1000
      SITE_DIAMETER = 0.03
    }
    /* Define new box object "box4" */
    box4 BOX {
      CORNERS = [-1.5, -1.5, -1.5], [1.5, 1.5, 1.5]
      FULLY_CLOSED = YES
      ABSORPTIVE {
        LIGAND = red_ligand
        ELEMENT = FRONT
      }
    }
  }
}

```

Section 3.7.5 The Instantiated Object

In order to perform a MCell simulation, you must instantiate all the physical objects (include the meta objects) as simulation components. The instantiation can be done by the MDL function *INSTANTIATE ... OBJECT*. Only the instantiated physical objects will be

simulated by the MCell simulator, and you can only obtain simulation results from objects which are instantiated. To obtain results, put the name of the defined physical objects into the template of the instantiated object, or define them inside the instantiated object. The same method is used to define the template of the instantiated object and the template of the meta object: both the pre-defined physical objects and newly created objects could be components of the instantiated object. This is the last chance to create new physical objects in a MCell simulation. You may copy or transform the pre-defined physical objects into the instantiated object the way that you did in the meta object template. The format of the function INSTANTIATE...OBJECT is given in Table 3.7-5.

Table 3.7-5 Creating The Simulation Components

Table 3.7-5

Format	INSTANTIATE <i>object_name</i> OBJECT { <i>object_templates</i> <i>geometric_transformation</i> }
Annotation	<i>object_templates</i> are the copy or the transformation operation applied to the pre-defined physical object or the creation of new physical objects; <i>geometric_transformation</i> is an optional operation.
Unit	None
Default	None

The name of the instantiated object will be used as the domain name in the output specification. The simulation results of the physical objects can be specified by its domain name and its own object name which is defined in the instantiated object.

Example 1

We use the meta object "group1" defined in *the previous example* to define the instantiated object "actor". The meta object "group1" will be copied into "actor" as the "left_group", it will be translated 4.0um along the Y-axis, and be saved as the object "right_group". All the objects included inside the instantiated object "actor" will be simulated by the MCell simulator. The domain name of the physical objects are "actor", you can export the visualization results of the objects "left_group" and "right_group" by choosing "actor.left_group" and "actor.right_group" in you output specifications.

MCell Code

```
/*Create simulation components:"left_group" and "right_group" */
INSTANTIATE actor OBJECT {
  /*Object templates 1: to copy the meta object "group1" as "left_group" */
  left_group OBJECT group1 { }
  /*Object templates 2: to translate the "group1" 4um along the Y axis to be the
  "right_group" */
  right_group OBJECT group1 { TRANSLATE = [0, 4, 0] }
}
```

Section 3.8 Surface Modifiers Definition of the Surface Regions

In MCell, you are able to define a desired surface region on an object template, and place effectors on the defined regions.

The term **surface modifier** applies to all operations applicable to the surface elements of physical objects. *surface_modifier* includes defining and removing the surface elements, changing permeability of the surface elements, and adding effectors to the surface of the physical objects. **You can specify an exact number of effectors or you can specify density of effectors on a given surface region.**

Section 3.8.1 Operations Applied to the Surface Element

The surface elements of the MCell physical objects are defined by the MDL command **ELEMENT**. Each value that is assigned to the keyword ELEMENT is called an *element_specifier*. The *element_specifier* varies with different physical objects. Besides defining the surface elements, you may also remove some surface elements by the MDL command **REMOVE_ELEMENT**. The specified surface elements are assigned to REMOVE_ELEMENT by the *element_specifier*. One ELEMENT, or REMOVE_ELEMENT statement can only specify one surface element. The format of the element operations are:

ELEMENT = *element_specifier*

REMOVE_ELEMENT = *element_specifier*

Different cases of the *element_specifier* are listed in Table 3.8-1.

Table 3.8-1 ELEMENT and *element_specifier*

Table3.8-1

Object type	Value of <i>element_specifier</i>
Any object	ALL_ELEMENTS
Box object	TOP (BOTTOM, LEFT, RIGHT, FRONT, BACK)
Polygon list object	<i>n</i> The numerical value <i>n</i> is the 0-based index number of a planar polygonal element.

From the Table 3.8-1 you may find that the command ELEMENT has different selection when it is applied to the box object and the polygon list object:

- If all the surface elements of an object are selected, the MDL command ELEMENT will be *ALL_ELEMENTS* no matter what kind of physical object it is.
- For a box object, the *element_specifier* is the six surface walls of the box.
- For the polygon list object, the *element_specifier* is the 0-based index number of a planar polygonal element, the index number is the sequence number defined by the MDL command ELEMENT_CONNECTIONS in the function [POLYGON_LIST](#).

Section 3.8.2 Permeability of Physical Objects

The permeability is the selective characteristic of the physical object to some molecules, this value depends on how easily molecules can pass through the object. If a molecule can go through the object smoothly, the permeability of the object to this molecule is called transparent (MDL keyword: TRANSPARENT); If the molecule is absorbed by the object when it meets the

object, we say the object is absorptive (MDL keyword: ABSORPTIVE); If the molecule is reflected by the object when they meet each other, we say the permeability of the object to this specific molecule is reflective (MDL keyword: REFLECTIVE), *the default permeability of physical objects in MCell is reflective*. We use the symbol *surface_permeability* to represent the permeability definition.

Permeability is an optional sub-function in physical object definitions. In order to define permeability, you should provide the name of the permeable ligand and the specific surface elements of the physical object. The format of the permeability definition is:

```
surface_permeability {
    MOLECULE = molecule_name
    ELEMENT = element_specifier
}
```

TRANSPARENT

If a surface element is defined as transparent (MDL keyword: TRANSPARENT) to some specified molecule, then that molecule can go through the surface element as if there is nothing there.

ABSORPTIVE

An absorptive (MDL keyword ABSORPTIVE) surface will absorb the specified molecule. When the specified molecule hits the absorptive surface, it will disappear from the simulation forever.

REFLECTIVE

If a surface element is reflective (MDL keyword REFLECTIVE), the molecule's direction will be changed after it hits the surface.

Section 3.8.3 Defining Surface Regions on Object Templates

A new feature of MCell lets the user define the surface region on object templates; the user may place effectors on specific regions of the simulated object. This way the user can control the distribution of the effectors based on a realistic model.

Two general applications of this function are:

- Making a template object, defining a region on that object, and placing effectors on that region;
- Making two copies of the template, adding a second region to one copy, placing effectors on both regions;

Of course you can use it in other ways as well. The definition of the function

DEFINE_SURFACE_REGIONS is give in the Table 3.8-2.

Table 3.8-2 The MDL Function DEFINE_SURFACE_REGIONS

Table3.8-2

Format	DEFINE_SURFACE_REGIONS {
	OBJECT <i>existing_object_name</i> {
	REGION <i>region_name</i> {
	ELEMENT_LIST = [<i>element_specifier</i>]
	}
	...
	}

	... }
Annotation	You may define regions for as many objects as desired. For the usage of <i>element_specifier</i> , please see section 3.8.1.
Unit	None
Default	None

Section 3.8.4 How to Add Effectors

The effectors are large molecules used in MCell simulations, such as receptor proteins on the plasma membrane, transporter proteins, enzymes on an intra- or extra-cellular scaffold etc., it is also called the effector site. The effector site can bind and unbind with the surface of the physical objects.

There are three different ways to add effectors to the physical object:

- Adding effectors to the surface of the whole simulation object: the MCell simulator will distribute the effector over the surface randomly according to the EFFECTOR_GRID_DENSITY and the DENSITY of the effectors;
- Placing an exact amount effectors over the defined surface regions;
- Placing the effectors over the defined surface regions based on the specified effector density.

Section 3.8.4.1 Adding Effectors to Physical Objects

The MDL function ADD_EFFECTOR tiles a physical object surface with stationary effector molecules. In this way, effectors will be distributed over the entire surface of the simulation object randomly. **The function ADD_EFFECTOR must be defined inside the physical object templates.** The distribution of the effector molecules depends on the parameter EFFECTOR_GRID_DENSITY, the area of the surface mesh element and the effector density value on the mesh element. Table 3.8-3 shows the definition of the function ADD_EFFECTOR.

Table 3.8-3 The MDL Function ADD_EFFECTOR

Table3.8-3

Format	<pre>ADD_EFFECTOR { STATE = state_name DENSITY = c ELEMENT = <i>element_specifier</i> POLE_ORIENTATION = <i>orientation</i> }</pre>
Annotation	<p>All the elements in this function are necessary elements. <i>element_specifier</i> is used to specify the particular object element.</p> <p><i>orientation</i> has two values: POSITIVE_FRONT and POSITIVE_BACK.</p>
Unit	μm^{-2}
Default	None.

STATE

The state name of the added effector. The value of STATE is one of the *state_name*

defined in the chemical reaction specification.

DENSITY

The number of the effector molecules over an unit area. The effector molecules will be distributed on the effector tiles according to this value. The effector tiles are the triangular grids which divides the mesh elements on the physical object surface. The number of the effector tiles created depends on the area of the mesh element and the value of `EFFECTOR_GRID_DENSITY`. MCell divides the mesh element of the physical object into the effector tiles using the **barycentric subdivision method**. Each effector is an effector tile that has a chemical reaction mechanism associated with it, the effector sites are chosen from the effector tiles by a random number.

ELEMENT

The MDL command `ELEMENT` is the surface element specifier of the physical objects. The command `ELEMENT` has different values for different types of physical objects. All the surface elements assigned to `ELEMENT` are represented by the symbol *element_specifier* in this menu. A detailed description of the command `ELEMENT` and its application will be introduced later in this section .

POLE_ORIENTATION

The command `POLE_ORIENTATION` is the orientation of the added effector sites relative to the normal vector of the physical object surface. Figure 3.6 illustrates the normal vector of the physical object surface. The face of the object that the normal vector leaves is the front face of the surface mesh element. The face that the normal vector enters the back surface. The `POLE_ORIENTATION` specifies which side of the object surface that the added effector lands on.

- a. If the effector is added on the front face of the surface element, then we define:
`POLE_ORIENTATION = POSITIVE_FRONT`
- b. If the effector is added on the back face of the surface element, then we define:
`POLE_ORIENTATION = POSITIVE_BACK`

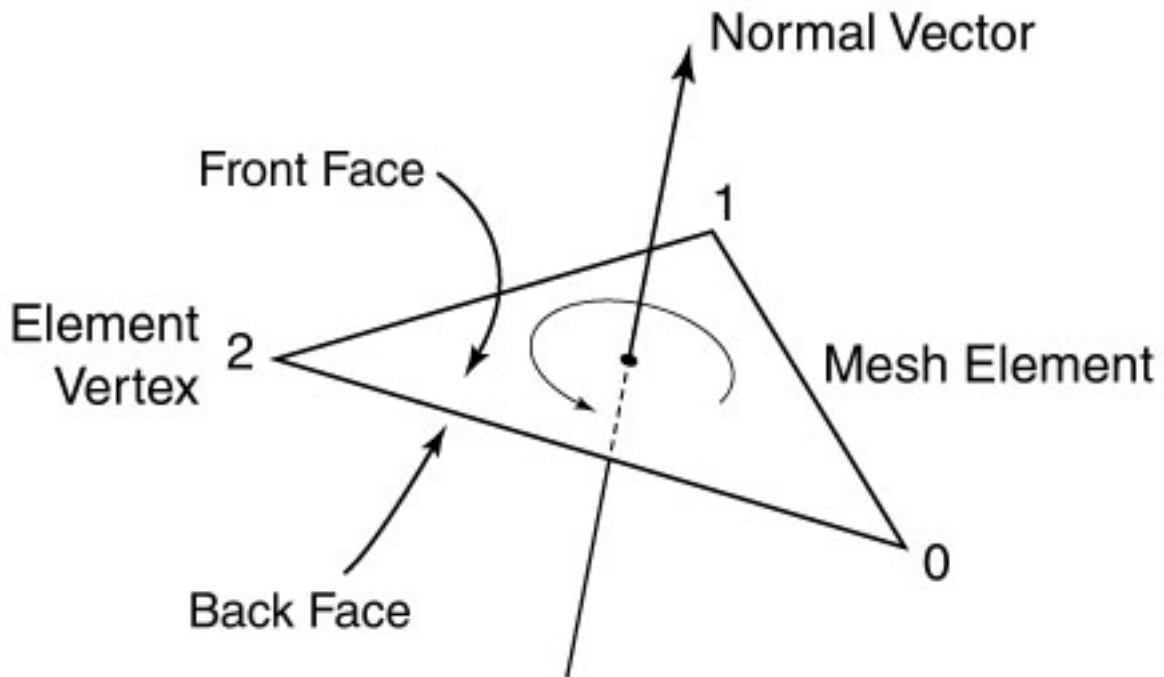


Figure 3.6 The triangle element shown in the figure is a surface mesh element of a physical object. Vector n is the normal vector of this surface mesh element. The front face and back face of the surface is decided by the direction of the normal vector n .

Example 1

Add the effector "AChR.R" to the bottom of a box object, the effector is added on the back side of the box surface. Another effector "AChR.AR" is added on a polygon list object, it is added on the front surface of the surface element number 5 and 6 of the polygon surface.

Notice: This is only a piece of MDL file. In the complete simulation input file, the function ADD_EFFECTOR should be defined inside the physical object template.

MCell Code

```

ADD_EFFECTOR {
  STATE = AChR.R
  DENSITY = 8935
  ELEMENT = BOTTOM
  POLE_ORIENTATION = POSITIVE_BACK
}
ADD_EFFECTOR {
  STATE = AChR.AR
  DENSITY = 1065
  ELEMENT = 5
  ELEMENT = 6
  POLE_ORIENTATION = POSITIVE_FRONT
}

```

Section 3.8.4.2 Adding Effectors to Defined Surface Regions

MCell provides new application for the distribution of the effectors. It can place effectors on defined surface regions, either by an exact number of effectors or by effector density. To add effectors to the defined surface regions with an exact number, please follow the definition given in Table 3.8-4.

Table 3.8-4 Adding an exact number effectors to the defined regions

Table3.8-4

Format	<pre> DEFINE_EFFECTOR_SITE_POSITIONS { REGION <i>existing_object_name</i>[<i>existing_region_name</i>] { EFFECTOR_STATE <i>effector_state_name</i> { NUMBER = <i>numerical_expression</i> POLE_ORIENTATION = <i>orientation</i> } ... } ... } </pre>
Annotation	You may add as many kinds of effectors to the same region as your desire, you may choose as many REGION as you want.
Unit	None
Default	None

To add effectors on the defined surface region by density is similar to what we did to add the exact number of effectors. What we need to do is to replace the NUMBER statement with the DENSITY statement. It is defined in Table 3.8-5.

Table 3.8-5 Adding effectors to the defined regions by density

Table3.8-5

Format	<pre> DEFINE_EFFECTOR_SITE_POSITIONS { REGION <i>existing_object_name</i>[<i>existing_region_name</i>] { EFFECTOR_STATE <i>effector_state_name</i> { DENSITY = <i>numerical_expression</i> POLE_ORIENTATION = <i>orientation</i> } ... } ... } </pre>
Annotation	You may add as many kinds of effectors to the same region as you desire and you may add effectors to any region.
Unit	None
Default	None

Section 3.9 Geometrical transformation (*geometric_transformation*)

To achieve a realistic simulation, the MCell simulator lets you apply geometrical transformations to physical objects to change their location or shape. The physical objects are the spherical ligand release site, the box object, the polygon list object, the meta object, and the instantiated

object. There are three geometrical transformations (represented in MDL by the symbol *geometric_transformation*): translation, rotation, and scaling, which are defined by the MDL function TRANSLATE, ROTATE, SCALE, respectively. We will introduce the method for applying these geometric transformations in MCell, and the basic theory behind each geometric transformation.

Section 3.9.1 TRANSLATE

The MDL function TRANSLATE is the operation used to move the physical object by a distance d_x along the X axis, a distance d_y along the Y axis, and a distance d_z along the Z axis. The object is moved by translating each point of the object to fulfill this operation. If point $P(x, y, z)$ is a point on the physical object, after translation it will become to the new point $P'(x', y', z')$. The coordinates of the new point after the movement can be calculated from:

$$x' = x + d_x, y' = y + d_y, z' = z + d_z \quad (3.7)$$

For computational convenience, we express the points P and P' as column vectors, and the transformation matrix as $T(d_x, d_y, d_z)$ using the homogenous coordinates:

$$P(x, y, z) = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, P'(x, y, z) = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}, T(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The matrix expression of the Equation (3.7) will be $P' = P + T$:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (3.8)$$

Using Equation (3.8), the location of the physical objects after any translation can be calculated easily. The format of the MDL function TRANSLATE is defined in the Table 3.9-1.

Table 3.9-1 Definition of The Function TRANSLATE

Table3.9-1

Format	TRANSLATE = [d_x, d_y, d_z]
Annotation	The array elements $d_x, d_y,$ and d_z are the translated distances of the defined object along the X, Y, Z directions.

Unit	μm
Default	No translation. $d_x=d_y=d_z=0$

Example 1

The ligand "BlueLigand" is released from the spherical release site "BlueSource". The diffusion constant of "BlueLigand" is $6.0 \cdot 10^{-6} \text{ cm}^2 \text{ s}^{-1}$. The origin of this site is at $[0, 0, 0]$, and the diameter of the sphere is $0.03 \mu\text{m}$, a total of 100 ligands will be released from this source site. The ligand source will be translated $0.2 \mu\text{m}$ along the X axis, and $0.3 \mu\text{m}$ along the Y axis.

MCell Code

```

/* Define ligand "Blueligand" */
DEFINE_MOLECULE BlueLigand {DIFFUSION_CONSTANT = 6.0e-6 }
/*Define the ligand source "BlueSource" */
BlueSource SPHERICAL_RELEASE_SITE {
  LOCATION = [0, 0, 0]
  LIGAND = BlueLigand
  NUMBER_TO_RELEASE = 100
  SITE_DIAMETER = 0.03
  /* Define the translation of the site */
  TRANSLATE = [0.2, 0.3, 0]
}

```

Section 3.9.2 ROTATE

In a MCell simulation, the physical object may be rotated through an angle ϕ around any arbitrary axis (a, b, c) . The axis (a, b, c) is the line extended from the world origin $(0, 0, 0)$ to a given point (a, b, c) . Positive angles of rotation are measured counter clockwise from the X axis toward the Y axis (right-hand rule). Derivation of the rotation matrix $R(\phi)$ of the rotation around the X, or Y, or Z axis can be found in computer graphics books and mathematical handbooks. Rotations around any arbitrary axis may be computed from the composition of the rotations around the X, Y, Z axis. The following is a short introduction about rotating around any arbitrary axis.

Let's denote point $P(x, y, z)$ as a point on the selected physical object. After rotation through angle ϕ , it will become point $P'(x', y', z')$. If the rotation is around the X axis, the rotation matrix is $R_x(\phi)$:

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) & 0 \\ 0 & \sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.9)$$

similarly the rotation around the Y axis is expressed by the matrix $R_y(\phi)$, which is:

$$R_y(\phi) = \begin{bmatrix} \cos(\phi) & 0 & \sin(\phi) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\phi) & 0 & \cos(\phi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.10)$$

The rotation around the Z axis is defined by the matrix $R_z(\phi)$:

$$R_z(\phi) = \begin{bmatrix} \cos(\phi) & -\sin(\phi) & 0 & 0 \\ \sin(\phi) & \cos(\phi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.11)$$

A rotation around any axis is computed from the combination of the matrices R_x , R_y , and R_z through five steps in the simulation. For a rotation of angle ϕ through an axis $[a, b, c]$, we define a point A which is located at $A(a, b, c)$, then the line segment OA is the rotation axis. The five steps of the computation about this arbitrary rotation are:

1. To rotate OA around the Z axis into YOZ plane to be line OA'
 To rotate the line OA, which is in 3D space, into the 2D y-z plane, we first project the line OA into the x-y plane, to get the projection OM_{xy} . The end point of the line OM_{xy} is the point $M_{xy}(a, b, 0)$. The projection is shown in Figure 3.7.

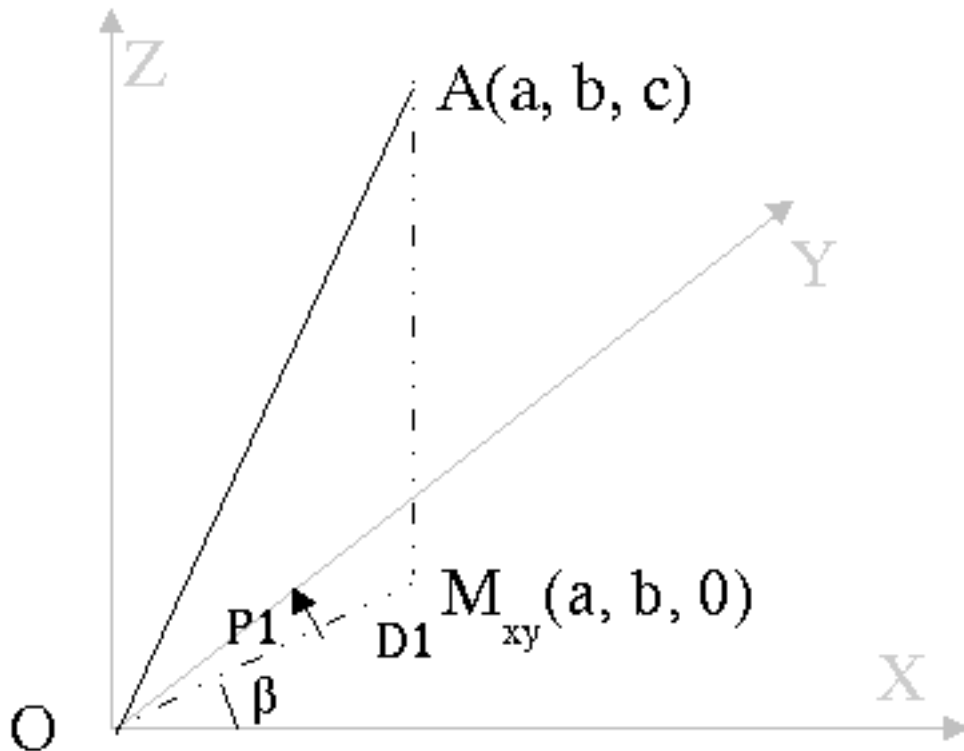


Figure 3.7 The line OA is projected into the 2D x-y plane, the projection is OM_{xy} . To rotate OA into the y-z plane, we need to rotate through angle $p1$, which is the angle between the Y axis and OM_{xy} . We have the relation that $p1=90-\beta$. After the rotation, the line OA becomes the line OA' on the YZ plane.

To rotate the line OA into the 2D plane y-z plane, we need to rotate OA around the Z axis by angle $p1$. Angle $p1$ is the angle between the Y axis and the projection OM_{xy} in the x-y plane, it may be computed from the equation:

$$p1 = 90 - \beta \quad (3.12)$$

The angle β in Equation (3.12) can be computed from the triangular relations of the line OM_{xy} inside the x-y plane, as:

$$\sin(\beta) = \frac{b}{|OM_{xy}|} = \frac{b}{\sqrt{a^2 + b^2}} \quad (3.13)$$

The rotation angle $p1$ can be solved by substituting the Equation (3.13) into Equation (3.12):

$$p1 = 90 - \arcsin\left(\frac{b}{\sqrt{a^2 + b^2}}\right) \quad (3.14)$$

After the rotation around the Z axis through the angle $p1$, the axis OA lies on the y-z plane as shown in Figure 3.8 to be the line OA'.

2. To rotate OA' around the X axis into Y axis to be line OA''
 As shown in Figure 3.8, after the rotation around the Z axis, we have the line OA', which lies on the y-z plane, if we denote the length of the line OM_{xy} in Figure 3.7 as D1, the coordinates of the point A' will be $A'(0, D1, c)$, we also denote the length of the line OA' as D2. In order to rotate the line OA' from the y-z plane to the Y axis, we need rotate through an angle $p2$ counter clockwise, which is the opposite value of the angle $p2$.

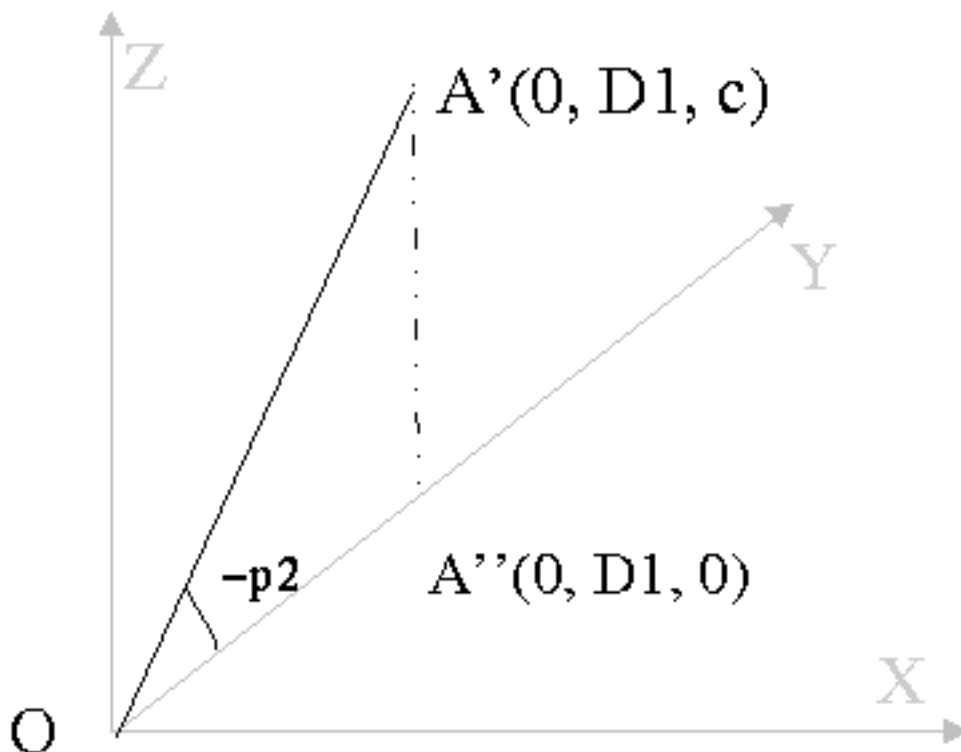


Figure 3.8. The line OA' lies on the y-z plane, it will rotate around the X axis through an angle $p2$ to lie

on the Y axis.

The rotation angle $p2$ (which should be counter-clockwise) is computed by the equations:

$$\sin(-p2) = \frac{c}{|OA'|} = \frac{c}{\sqrt{c^2 + DI^2}} = \frac{c}{\sqrt{a^2 + b^2 + c^2}} \quad (3.15)$$

$$p2 = -\arcsin\left(\frac{b}{\sqrt{a^2 + b^2 + c^2}}\right) \quad (3.16)$$

3. To rotate by angle ϕ around the Y axis

The line OA'' is the rotation axis; since it is on the Y axis, the rotation of the angle ϕ is around the Y axis, and the rotation matrix will be $R_Y(\phi)$. By now, the operations we have to perform this rotation are: $R_Z(p1)R_X(p2)R_Y(\phi)$. But we didn't finish yet, since the rotation axis is not on its original position, we need to rotate it back by the following two steps to finish this rotation.

4. To rotate line OA'' back to OA'

To rotate line OA'' back to the line OA' , a rotation of angle $-p2$ around the X axis is needed. The rotation matrix will be $R_X(-p2)$. This operation is the reverse operation of step 2.

5. To rotate line OA' back to line OA

To rotate OA' around Z axis with an angle $-p1$, it will go back to line OA (the original rotation axis). This rotation is the reverse operation of the step 1. The rotation matrix will be $R_Z(-p1)$.

After the sequence of rotations, the point $P(x, y, z)$ in the selected object is transformed to the point $P'(x', y', z')$, the coordinates of the point $P'(x', y', z')$ can be computed from the equation as:

$$P'(x', y', z') = R_Z(p1)R_X(p2)R_Y(\phi)R_X(-p2)R_Z(-p1) \cdot P(x, y, z) \quad (3.17)$$

Any rotation around an axis $[a, b, c]$ through an angle ϕ is defined by the MDL function ROTATE, the format of this function is shown in Table 3.9-2.

Table 3.9-2 Definition of the Function ROTATE

Table3.9-2

Format	ROTATE = [a, b, c], ϕ
Annotation	The array elements a, b, and c are the coordinates of the end point of the rotate axis, ϕ is the angle that the object will be rotated around the axis [a, b, c].
Unit	The array elements a, b, c have units of μm ; The parameter ϕ has a unit of <i>degree</i> .
Default	No rotation.

Example 1

The ligand "BlueLigand" with a diffusion constant of $6.0 \cdot 10^{-6}$ is released from the spherical release site "BlueSource". The origin of this site is at $[0, 0, 0]$, the diameter of the sphere is $0.03 \mu m$, a total of 100 ligands will be released from this source site. The ligand source will be rotated around the axis $[1, -1, 0]$ around a 45° angle.

MCell Code

```

/* Define ligand "Blueligand" */
DEFINE_MOLECULE BlueLigand {DIFFUSION_CONSTANT = 6.0e-6 }
/*Define the ligand source "BlueSource" */
BlueSource SPHERICAL_RELEASE_SITE {
  LOCATION = [0, 0, 0]
  LIGAND = BlueLigand
  NUMBER_TO_RELEASE = 100
  SITE_DIAMETER = 0.03
  /* Define the rotation of the site */
  ROTATE = [1, -1, 0], 45
}

```

Section 3.9.3 SCALE

Beside translation and rotation, you can also scale the physical object in three dimensions. If the point $P(x, y, z)$ is a point on the selected object, then after the object is scaled by a factor s_x along the X direction, a factor s_y along the Y direction, and a factor s_z along the Z direction, point P will become the point $P'(x', y', z')$. The scaling operation helps you to enlarge or shrink the object in the desired direction. Scaling is expressed by the equation $P' = S(s_x, s_y, s_z)P$. Where S is the scaling matrix with the format:

$$S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

To have a matrix form of the scaling equation, we express the point $P(x, y, z)$ and $P'(x', y', z')$ in a vector format using the homogenous coordinates:

$$P(x, y, z) = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, P'(x', y', z') = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}$$

Then the matrix form of the scaling equation is:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (3.18)$$

The scaling shown in equation 3.18 is an operation about the origin of the Cartesian coordinate system. The physical object may be scaled by applying equation 3.16 to each point of the object, the proportion of the scaled object depends on the scaling factors in each direction, with a uniform scaling ($s_x = s_y = s_z$), the proportions of the scaled object will not be changed. Successive scaling are multiplicative, if you scale an object twice by the scaling matrices $S1(s1_x, s1_y, s1_z)$, $S2(s2_x, s2_y, s2_z)$, when the scaling operations are finished, the point P on the selected object will be turned to the point $P''(x'', y'', z'')$, the point P'' can be computed from the equation:

$$P'' = S(s_{x1} s_{x2}, s_{y1} s_{y2}, s_{z1} s_{z2}) \cdot P \quad (3.19)$$

The scaling operation is defined by the MDL command SCALE, the format of the command is given in Table 3.9-3.

Table 3.9-3 Definition of The Function SCALE

Table3.9-3

Format	SCALE = [s_x, s_y, s_z]
Annotation	The array elements s_x, s_y, s_z are the scaling factors along the X, Y, and Z directions.
Unit	None.
Default	No scaling.

Example 1

The ligand "BlueLigand" with a diffusion constant of 6.0×10^{-6} is released from the spherical release site "BlueSource". The origin of this site is [0, 0, 0], and the diameter of the sphere is 0.03 μm . A total of 100 ligands will be released from this source site. The ligand source will be scaled by factors of 1 along the X axis, 1.5 along the Y axis and 2 along the Z axis.

MCell Code

```
/* Define ligand "Blueligand" */
DEFINE_MOLECULE BlueLigand {DIFFUSION_CONSTANT = 6.0e-6 }
/*Define the ligand source "BlueSource" */
BlueSource SPHERICAL_RELEASE_SITE {
  LOCATION = [0, 0, 0]
  LIGAND = BlueLigand
  NUMBER_TO_RELEASE = 100
  SITE_DIAMETER = 0.03
  /* Define the scaling of the site */
  SCALE = [1, 1.5, 2]
}
```

Section 3.10 MCell Output

In order to get simulation results, you must specify the output format of the simulation in your input file. MCell can output the simulation results for a variety of analytical purposes, visualization data for graphical animation, numerical data for chemical reaction analysis, or the C language style output for the data analysis in a customized way.

The user may trace the statistics of the simulation event in a space or time dependent way.

The amount of the simulation results are enormous, but you can specify the directory and the prefix of the output files in the MDL output specification.

For visualization convenience, MCell provides users with several different formats of output data to fit visualization tools that you may use for animation. You may also use the unique MCell facilities for the selective filtering, formatting, and timing of the output data. The MDL function VIZ_DATA_OUTPUT helps you to define the visualization output type of the MCell simulation, the function REACTION_DATA_OUTPUT is used to define the chemical reaction results as the output of the simulation. In this section, we will tell you how to define different simulation output in the MCell simulation.

Section 3.10.1 The Visualization Output

MCell is able to export simulation results to fit the native data formats of a variety of graphic tools: Data Explore, IRIT, Rayshade, Renderman and Povray. To define the visualization type of results from the MCell simulation, you should choose your preferred visualization mode first, then specify when you want to save your results, and which object you want to observe. The MDL function VIZ_DATA_OUTPUT is used to define the output of the visualization results from the simulation. The format of the function VIZ_DATA_OUTPUT is defined in Table 3.10-1.

Table 3.10-1 The MCell Specification of The Visualization Output

Table3.10-1

<p>Format</p>	<pre>VIZ_DATA_OUTPUT { MODE = <i>visualization_mode</i> <i>output_specifier</i> MOLECULE_FILE_PREFIX = "<i>file_prefix</i>" OBJECT_FILE_PREFIXES { surface_object_name = <i>text_expression</i> ... } STATE_VALUES { object_name = <i>numerical_expression</i> ... } }</pre>
<p>Annotation</p>	<p><i>visualization_mode</i> represents the MDL keywords: DX (Data Explore), IRIT (IRIT), RAYSHADE (Rayshade), POVRAY (Povray), RENDERMAN (Renderman), VOXEL_IMAGE_MODE, VOXEL_VOLUME_MODE.</p> <p><i>output_specifier</i> could be either <i>iteration_specifier</i>, ITERATION_FRAME_DATA or <i>time_specifier</i>. <i>iteration_specifier</i> uses the iteration number to choose the output data, and the <i>time_specifier</i> use the real time to define the output.</p> <p>Statement MOLECULE_FILE_PREFIX is used to define the prefix of the output file for molecule positions and states at each selected time step. The prefix must be contained inside double quotes.</p> <p>Function OBJECT_FILE_PREFIXES is used to select the output physical objects, and assign directories or file names for the output data.</p> <p>The function STATE_VALUES assign a state value to each output object (logical or physical object), this state value is</p>

	used to identify each separate object. The output mode VOXEL_IMAGE_MODE outputs pgm (portable graymap format) images right-side-up, and voxelize the effector states as well as the ligand states. The output mode VOXEL_VOLUME_MODE outputs counts of molecule states contained within a voxelized volume model space
Unit	None
Default	No visualization output.

Section 3.10.1.1 The Output Specifier

You may use either the iteration number or the real simulation time to specify the simulation output of a MCell simulation. The two different output list formats used in MCell are: *iteration_specifier*, which uses iteration numbers to select the output range and *time_specifier*, which uses real time to define the output range. By using *time_specifier* you do not need to recalculate the iteration numbers for output when the time step of simulation changes. Inside a VIZ_DATA_OUTPUT function, you can only use one of the output_specifier, either the *iteration_specifier* or the *time_specifier*, you are not allowed use them both at the same time.

Iteration specifier (*iteration_specifier*) includes two MDL specifications that are used to select the iteration steps for output. The two *iteration_specifier* are ITERATION_LIST and ITERATION_FRAME_DATA. When you choose ITERATIONS_LIST as the *iteration_specifier*, all the information (ligand, effector, surface elements, positions, states etc.) of the selected objects will be saved at all the selected steps. On the contrary, the function ITERATION_FRAME_DATA is more flexible, it helps you to limit your output by specifying the interested properties of the objects at the specified iteration steps, so you will only get the results that you defined within the *frame_data_specifier*.

ITERATION_LIST = *iteration_specifier*

iteration_specifier lists the iteration numbers specified by the user for output. If we let the interger number N represent the iteration number, then the *iteration_specifier* has the format:

```
[N1, N2, N3, ..., Nn]  
[[N1 TO Nn STEP dt]]  
[list_of_iterations,[N1 TO Nn STEP dt]]  
[[N1 TO Nn STEP dt], list_of_iterations]
```

The value of the MDL command ITERATIONS_LIST is a numerical array of the iteration times or an *iteration_range_specifier*, or the mixture of the iteration number lists and *iteration_range_specifier*. An *iteration_range_specifier* has the form as:
[start_iteration_time TO end_iteration_time **STEP** time_step]

TIME_LIST = *time_specifier*

The TIME_LIST has exactly the same format as ITERATION_LIST, but instead of the iteration numbers it uses time to define the output. You may use a list of times, or a time range, or the combination of list and range of time to choose the simulation output.

time_specifier has the following possible values:

```
[t1, t2, t3, ..., tn]  
[[t1 TO tn STEP dt]]
```

[list_of_times,[t₁ TO t_n STEP dt]]
 [[t₁ TO t_n STEP dt], list_of_times]
 Where t₁, t₂, t₃, t_n and dt are real numbers, they represent the time in the real world with a unit of **second**.

ITERATION_FRAME_DATA

ITERATION_FRAME_DATA {

frame_data_specifier = iteration_specifier

... }

The MDL function ITERATION_FRAME_DATA specifies the type of the visualization data to be output and the simulation iteration numbers after which visualization data will be output. By associating the specified visualization data with indicated iteration numbers, the user can specify as many frame data associations as needed. The **frame_data_specifier** includes:

SURFACE_POSITIONS
 SURFACE_STATES
 EFFECTOR_POSITIONS
 EFFECTOR_STATES
 LIGAND_POSITIONS
 LIGAND_STATES

Section 3.10.1.2 The Function OBJECT_FILE_PREFIXES

The function OBJECT_FILE_PREFIXES is the file designator that is associated with the selected physical objects. This function selects the named surface objects for visualization output. The format of the function OBJECT_FILE_PREFIXES is:

OBJECT_FILE_PREFIXES {

surface_object_name = "output_domain_name"

...

}

In the definition of the function OBJECT_FILE_PREFIXES, *output_domain_name* is the user defined domain name that will be used to save the output files. You may define as many physical objects as you need in this function by giving them a different *output_domain_name*. The surface object name should include both the instantiated object name and the object name with a format like "**instantiated_object_name.surface_object_name**". Large amounts of data will be output to different files under the indicated directory or the current directory. If we use variable *n* to represent the selected iteration numbers in visualization output, then we will have the following output files:

n_ME-n_SSV.dat
output_domain_name.effectors_site_positions.n.data
output_domain_name.effectors_site_positions.n.dx
output_domain_name.effectors_site_states.n.data
output_domain_name.effectors_site_states.n.dx
output_domain_name.mesh_elements.n.data
output_domain_name.mesh_elements.n.dx
output_domain_name.mesh_element_states.n.data

output_domain_name.mesh_element_states.n.dx
file_prefix.molecule_positions.n.data
file_prefix.molecule_positions.n.dx
file_prefix.molecule_states.n.dx

For each iteration MCell generates a set of files like the one listed above.

Section 3.10.2 The Chemical Reaction Output

The numerical results for chemical reactions from a MCell simulation may be obtained by using the function REACTION_DATA_OUTPUT in the simulation input file. The function REACTION_DATA_OUTPUT is defined on a time dependent and/or location dependent basis. The desired output steps are selected either by the iteration number or the real time value - there are three different ways for a user to choose. The MDL command COUNT provides a powerful and flexible expression (COUNT items include what to count, when to count, where to count, and how to count in the output specifications) to specify the chemical reactions results as output. Table 3.10-2 shows the format of the function REACTION_DATA_OUTPUT. Detail description about the command COUNT will follow Table 3.10-2.

Table 3.10-2 The MCell Chemical Reaction Output

Table3.10-2

Format	<pre> REACTION_DATA_OUTPUT { <i>reaction_specifier</i> {COUNT [<i>count_what</i>, <i>count_where</i>, <i>count_how</i>, ...] ...} => <i>file_name</i> ... }</pre>
Annotation	<p>reaction_specifier is the time expression for the chemical reaction output. It could be either <i>iteration_specifier</i>, STEP expression or <i>time_specifier</i>. The three different formats are defined by three different keywords: ITERATION_LIST TIME_LIST STEP = c .</p> <p>COUNT statements are used to select the desired reaction results as output to the file defined <i>file_name</i>.</p>
Unit	None
Default	None

The MDL command COUNT can be added as needed, you can combine the counted items to freely define your chemical reaction output. Now, let us see what we can count, where can we count and how can we count in the COUNT statements.

reaction_specifier

The **reaction_specifier** has three different expressions. It could be either *iteration_specifier*, *time_specifier* or STEP expression. We introduced *iteration_specifier* and *time_specifier* in the last section with *output_specifier*. Both the STEP and TIME_LIST definitions use time intervals (**seconds**) to count the points at which data is saved during the simulation iterations.

STEP = t (seconds)

t, a real number, represents the time interval between each data output. This function lets the user save the reaction output with a fixed frequency, counting from the beginning of the simulation.

count_what

specifies the name of the reaction intermediate (ligand or effector state) or reaction state transition (binding, unbinding, etc.), or the collision between ligand and surface to be output.

count_where

specifies the spatial location of the item indicated by **count_what**, it might be WORLD (the whole simulation world), existing_object or existing_region.

count_how

can be a list of specialized options for additional control over the manner of counting.

The allowed values for *count_where* and *count_how* depend upon the item specified by *count_what*. Note that mathematical expressions involving multiple COUNT statements may be specified within the braces {} surrounding the COUNT statement block. Please see the presently implemented COUNT statement blocks (subject to change and expansion):

- a. **{COUNT[*ligand_name*, WORLD, FOR_EACH_TIME_STEP]} =>file_name**
Count number of ligands "*ligand_name*" in the entire simulation environment at each time step.
- b. **{COUNT[*ligand_name*, existing_region, FOR_EACH_TIME_STEP, ALL_HITS]} =>file_name**
Count all hits of ligands "*ligand_name*" with the selected surface region at each time step.
- c. **{COUNT[*ligand_name*, existing_region, FOR_EACH_TIME_STEP, ALL_HITS_FRONT]} =>file_name**
Count ligand "*ligand_name*" hits only with the FRONT side of the surface region at each time step.
- d. **{COUNT[*ligand_name*, existing_region, FOR_EACH_TIME_STEP, ALL_HITS_BACK]} =>file_name**
Count ligand "*ligand_name*" hits with the BACK side of the surface region at each time step.
- e. **{COUNT[*ligand_name*, object_name, FOR_EACH_TIME_STEP]} =>file_name**
Count number of ligands "*ligand_name*" inside the fully closed physical object comprising the instantiated physical or meta object named *object_name* at each time step.
- f. **{COUNT[*state*, WORLD, FOR_EACH_TIME_STEP]} => file_name**
Count all the effector sites in the specified state in the entire simulation environment.
- g. **{COUNT[*state*[>*adjacent_state*], WORLD, SUM_OVER_ALL_EFFECTORS, FOR_EACH_TIME_STEP, ALL_EVENTS]} =>file_name**
Count all transitions from *state* to *adjacent_state*, summing over all effectors in the entire simulation environment, and output the number of transitions that occur during each time step.
- h. **{COUNT[*state*[>*adjacent_state*], WORLD, SUM_OVER_ALL_EFFECTORS, FOR_EACH_TIME_STEP, INITIAL_EVENTS]} =>file_name**
Count the subset of transitions from *state* to *adjacent_state* in which the previous state was not *adjacent_state*. Sum over all effectors in the entire simulation environment, and output the number of transitions that occur during each time step.
- i. **{COUNT[*state*[>*adjacent_state*], WORLD, FOR_EACH_LIGAND, SUM_OVER_ALL_TIME_STEPS]} =>file_name**
Count the subset of transitions from *state* to *adjacent_state* for each individual ligand molecule of type *ligand_name* over all time steps.

-
- j. `{COUNT[state>adjacent_state], WORLD, SUM_OVER_ALL_LIGANDS, FOR_EACH_TIME_STEP}}`
`=>file_name`
Count the subset of transitions from *state* to *adjacent_state*. Sum over all ligand molecules in the entire simulation environment, and output the number of transitions that occur during each time step.
- k. `{COUNT[state>adjacent_state], WORLD, SUM_OVER_ALL_LIGANDS, CUMULATE_FOR_EACH_TIME_STEP}}`
`=>file_name`
Count the subset of transitions from *state* to *adjacent_state*. Sum over all ligand molecules in the entire simulation environment, and output the number of transitions that cumulate at each time step.
- l. `{COUNT[state>adjacent_state], existing_region, SUM_OVER_ALL_LIGANDS, FOR_EACH_TIME_STEP}}`
`=>file_name`
Count the subset of transitions from *state* to *adjacent_state*. Sum all ligand molecules over the defined surface region at each time step.
- m. `{COUNT[state>adjacent_state], existing_region, SUM_OVER_ALL_LIGANDS, CUMULATE_FOR_EACH_TIME_STEP}}`
`=>file_name`
Count the subset of transitions from *state* to *adjacent_state*. Sum all ligand molecules over the defined surface region, and output the number of transitions that cumulate at each time step.
- n. `{COUNT[state>adjacent_state], existing_region, FOR_EACH_LIGAND, SUM_OVER_ALL_TIME_STEPS}}`
`=>file_name`
Count the subset of transitions from *state* to *adjacent_state* for each ligand molecule. Sum all transitions over the simulation on the defined surface region.
- o. `{COUNT[state>adjacent_state], existing_region, SUM_OVER_ALL_EFFECTORS, CUMULATE_FOR_EACH_TIME_STEP, ALL_EVENTS}}`
`=>file_name`
Count the subset of transitions from *state* to *adjacent_state*. Sum over all effectors on the defined surface region, and output the number of transitions that cumulate at each time step.
- p. `{COUNT[state>adjacent_state], existing_region, SUM_OVER_ALL_EFFECTORS, CUMULATE_FOR_EACH_TIME_STEP, INITIAL_EVENTS}}`
`=>file_name`
Count the subset of transitions from *state* to *adjacent_state* in which the previous state was not *adjacent_state*. Sum over all effectors on the defined surface region, and output the number of transitions that cumulate at each time step.
- q. `{COUNT[state>adjacent_state], existing_region, SUM_OVER_ALL_EFFECTORS, FOR_EACH_TIME_STEP, ALL_EVENTS}}`
`=>file_name`
Count the subset of transitions from *state* to *adjacent_state*. Sum over all effectors on the defined surface region, and output the number of transitions that occurs during each time step.
- r. `{COUNT[state>adjacent_state], existing_region, SUM_OVER_ALL_EFFECTORS, FOR_EACH_TIME_STEP, INITIAL_EVENTS}}`
`=>file_name`
Count the subset of transitions from *state* to *adjacent_state* in which the previous state was not *adjacent_state*. Sum over all effectors on the defined surface region, and output the number of transitions that occurs during each time step.
- s. `{COUNT[state, existing_region, FOR_EACH_TIME_STEP]}`
`=>file_name`
Count the reaction state of *state* on the defined surface region at each time step.
- t. `{COUNT expression1 + COUNT expression2}`
`=> file_name`
Addition of the count expression
- u. `{COUNT expression1 - COUNT expression2}`
`=> file_name`
Subtraction of the count expression
- v. `{COUNT expression1 * COUNT expression2}`
`=> file_name`
Times of the count expression
-

- w. `{COUNT expression1 / COUNT expression2} => file_name`
Division of the count expression
- x. `{EXPRESSION [numerical_expression]} =>file_name`
Evaluate *numerical_expression*. This can be useful in mathematical expressions involving one or more COUNT statements.

Section 3.10.3 The Generic "C" Language Output

The MCell simulator supports several generic file output commands which conform to the normal C language syntax. Please notice that only the file output statements are supported by MCell. We only list some commonly used output file functions in the following list. For more detailed description about the C language output functions please see the C language documentation.

- a. **fopen()**: Opens a file. If the handle of the file is **fp**, then the format of this function is ***fp = fopen(file_name, "mode")***. The legal values for mode are:
 - r
open a text file for reading;
 - w
create a text file for reading;
 - a
append to a text file;
 - rb
open a binary file for reading;
 - wb
create a binary file for reading;
 - ab
append to a binary file;
 - r+
open a text file for read/write;
 - w+
create a text file for read/write;
 - a+
append or create a text file for read/write;
 - r+b
open a binary file for read/write;
 - w+b
create a binary file for read/write;
 - a+b
append or create a binary file for read/write.
- b. **fclose()**: Closes a file. The format of this function is ***fclose(fp)***.
- c. **printf()**: Print out the formatted text and data to the window where MCell started. The format of this function is ***printf("text expression\n", 10)***.
- d. **fprintf()**: Write the formatted text and data to the file associated with the defined name *file_name*. The format of this function is ***fprintf(fp, "scale = [%f, %f, %f]\n", 1.0, 1.0, 2.0)***.

- e. **print_time()**: Print out formatted date and time information to the window where MCell started. The format of this function is **print_time("%m%d%y\n")**.
- f. **fprint_time()**: Write the formatted date and time information to the file associated with the defined name *file_name*. The format of this function is **fprint_time(fp, "%m%d%y\n")**.

Example 1

In this example, let us assume that we have already defined the following items in the previous part of the simulation input file: the ligand "ACh", the effectors "AChR.R" and "AChR.AR", the box objects named "box1" and "box2". Then we continue from the instantiated object "microdomain", and define both the visualization output and chemical reaction output of the simulation. In order to separate the different objects in the animation, we set the state values for ligand "ACh" as 1, effector "AChR.R" as 5, effector "AChR.AR" as 10. We would like to save the results related with the box object "box1" into files begin with the domain name "box_1", the results about the object "box2" into the files with a domain name as "box_2". The visualization results will be exported every 100 iterations from the beginning of the simulation to the 2000 iterations. The chemical reaction will be tracked every iteration, both the ligand and effectors will be recorded through the whole simulation world, the results of the ligand will be saved into the file named "visualize.mdl.ACh", the effector results can be found from the files named "effector.R" and "effector.AR". After the simulation, you will have the following type of output files in the current simulation directory:

```
n_ME-n_SSV.dat
box_1.effectors_site_positions.n.data
box_1.effectors_site_positions.n.dx
box_1.effectors_site_states.n.data
box_1.effectors_site_states.n.dx
box_1.mesh_elements.n.data
box_1.mesh_elements.n.dx
box_1.mesh_element_states.n.data
box_1.mesh_element_states.n.dx
box_2.effectors_site_positions.n.data
box_2.effectors_site_positions.n.dx
box_2.effectors_site_states.n.data
box_2.effectors_site_states.n.dx
box_2.mesh_elements.n.data
box_2.mesh_elements.n.dx
box_2.mesh_element_states.n.data
box_2.mesh_element_states.n.dx
molecule_positions.n.data
molecule_positions.n.dx
molecule_states.n.dx
effector.R
effector.AR
visualize.mdl.ACh
```

MCell Code

```
/* Define the instantiated object "microdomain" */
INSTANTIATE microdomain OBJECT {
    m1 OBJECT ACh_release_site {}
    m2 OBJECT box1 {}
    m3 OBJECT box2 {}
}
/*Define the visualization output */
VIZ_DATA_OUTPUT {
    MODE = DX
    STATE_VALUES {
        ACh = 1
        AChR.R = 5
```

```
    AChR.AR = 10
  }
  /* Designate output file name for object "m2" and "m3" */
  OBJECT_FILE_PREFIXES {
    microdomain.m2 = "box_1"
    microdomain.m3 = "box_2"
  }
  ITERATION_LIST = [[ 0 TO 2000 STEP 100]]
}
/*Define the chemical reaction output */
REACTION_DATA_OUTPUT {
  STEP = dt
  { COUNT [AChR.R, WORLD, FOR_EACH_TIME_STEP]} =>"effector.R"
  { COUNT [AChR.AR, WORLD, FOR_EACH_TIME_STEP]} =>"effector.AR"
  /* count ligand "ACh" at each time step in the through the whole simulation and put file to
  the file "visualize.mdl.ACh" */
  { COUNT [ACh, WORLD, FOR_EACH_TIME_STEP]} => INPUT_FILE & ".ACh"
}
```

Index of the MCell Keywords

ABS 13
ABSORPTIVE 43
ACOS 13
ADD_EFFECTOR 44
ALL_ELEMENTS 42
ALL_EVENTS 59
ALL_HITS 59
ALL_HITS_BACK 59
ALL_HITS_FRONT 59
ASIN 13
ATAN 13
BACK 42
BOTH_POLE 31
BOTTOM 42
BOX 35
CEIL 13
CHARGE
CORNERS 36
COS 13
COUNT 58
CUMULATE_FOR_EACH_TIME_STEP 60
DEFINE_EFFECTOR_SITE_POSITIONS 47
DEFINE_MOLECULE 25
DEFINE_REACTION 28
DEFINE_RELEASE_PATTERN 25
DEFINE_SURFACE_REGIONS 43
DELAY 28
DENSITY 45
DIFFUSION_CONSTANT 26
DX 55

MCell Appendix

EFFECTOR_GRID_DENSITY 19

EFFECTOR_POSITIONS 57

EFFECTOR_STATES 57

EITHER_POLE 31

ELEMENT 45

ELEMENT_CONNECTIONS 42

EXP 13

EXPRESSION 61

FALSE 36

FLOOR 13

FOR_EACH_LIGAND 59

FOR_EACH_TIME_STEP 59

FOR_EACH_TIME_STEP 59

FOR_EACH_TIME_STEP 59

FOR_EACH_TIME_STEP 59

FRONT 42

FULLY_CLOSED 36

FULLY_RANDOM 23

INCLUDE_FILE 15

INITIAL_EVENTS 59

INSTANTIATE...OBJECT 41

IRIT 55

ITERATION_FRAME_DATA 55

ITERATION_LIST

LEFT 42

LIGAND_POSITIONS 57

LIGAND_STATES 57

LOCATION 33

LOG 13

LOG10 13

MAX 13

MCell Appendix

MIN 13
MOD 13
MODE
MOLECULE 33
MOLECULE_FILE_PREFIX
NEGATIVE_POLE 31
NO 36
NUMBER_BOUND 31
NUMBER_OF_TRAINS 28
NUMBER_TO_RELEASE = n 33
OBJECT 39
OBJECT_FILE_PREFIXES 55
PARTITION_X 24
PARTITION_Y 24
PARTITION_Z 24
POLE_ORIENTATION 45
POSITIVE_BACK 45
POSITIVE_FRONT 45
POSITIVE_POLE 31
POVRAY 55
RADIAL_DIRECTIONS 19
RADIAL_SUBDIVISIONS 19
RAYSHADE 55
REACTION_DATA_OUTPUT 58
REFERENCE_DIFFUSION_CONSTANT 26
REFERENCE_STATE 31
REFLECTIVE 43
REGION 43
RELEASE_INTERVAL 28
RELEASE_PATTERN 34
RELEASE_PROBABILITY 34

MCell Appendix

REMOVE_ELEMENT 36
RENDERMAN 55
RIGHT 42
ROTATE 48
ROUND_OFF 13
SCALE 48
SIN 13
SITE_DIAMETER 34
SPHERICAL_RELEASE_SITE 25
SQRT 13
STATE 44
STATE_VALUES 55
STEP 58
SUM_OVER_ALL_EFFECTORS 59
SUM_OVER_ALL_LIGANDS 60
SUM_OVER_ALL_TIME_STEPS 59
SURFACE_POSITIONS 57
SURFACE_STATES 57
TAN 13
TIME_LIST
TO 56
TOP 42
TRAIN_DURATION 28
TRAIN_INTERVAL 28
TRANSLATE 48
TRANSPARENT 42
TRUE 36
UNLIMITED 28
VERTEX_LIST 37
VIZ_DATA_OUTPUT 55
VOXEL_IMAGE_MODE 55

MCell Appendix

VOXEL_VOLUME_MODE 55

WORLD 59

YES 36