.. include:: ../isonum.txt

.. include:: ../isogrk1.txt

.. |Ca2+| replace:: Ca\ :sup:`2+`

======================

Visualization 2: video

======================

Some users may want to further analyze simulation results in a spatio-temporal manner, e.g., 3D visualization. The script '61\_make\_video.py' introduces a method to make videos of molecular dynamics using the visualization software Mayavi and the video generation software ffmpeg. In this script, the ConnectAnalysis class offers the use of user-defined function to access all dimension of the simulation data, i.e., a time series of 4D images (3D space + 16 molecular slots; Lines 55-60). The time-developing 4D images are consecutively passed to the function 'event' (Lines 34-52).

The function 'event' has three variables: the lattice, sys\_param, and usr\_param. In this function, the function 'get\_loc\_molecules' provides the locations of a specified molecule in the specified lattice (Lines 10, 38, 40), and the molecular locations are plotted in the pre-defined 3D space (Lines 41-43). The visualized molecular distribution is saved as XXXX.png in 'output\_dir' (Lines 24, 47-48). This process is repeated in a time-development manner.

.. literalinclude:: ../../tutorial/1/61\_make\_video.py

:language: python

:linenos:

:caption: 61\_make\_video.py

|

Finally, the video generation software 'ffmpeg' assembles the sequential images into a single video (Lines 63-69). The generated videos are shown below. In the case of FRAP (Lines 9-13), we can observe the re-filling of YFP (yellow points) to the target spine after the photobleaching.

.. video:: \_static/YFP.mp4

|

In the case of |Ca2+| influx via NMDA receptors (Lines 16-20), we can observe the spread of |Ca2+| ions (blue points) from the stimulated spine.

.. video:: \_static/Ca.mp4

.. include:: ../isonum.txt

.. include:: ../isogrk1.txt

.. |Ca2+| replace:: Ca\ :sup:`2+`

===========================

Visualization 1: graph plot

===========================

Either single run or ConnectRun stores the results of simulation in the LM-format files. The LM-format files are indeed HDF container files, and users can directly analyze them; however, it is not so easy to capture `the data structure <https://github.com/Luthey-Schulten-Lab/Lattice\_Microbes/blob/master/docs/HDF5FileFormat.text>`\_. Also, similar demands on the analyses are shared by users.

LD thus provides functions for analyses, in particular, to connect data across multiple unit runs. The total numbers of molecules are saved in the LM files; therefore, it is easy to obtain the total concentration of target molecules. In the script '51\_plot\_conc.py', the 'ConnectTotalConcs' class handle this (Lines 25-27). The variable 'domain\_name' contains a domain name for the calculation of the volume, and the variable 'lm\_files' contains a list of target LM files. In this case, Line 10 specifies a target directory ('simulation\_dir'), in which all LM files are obtained in an ascending order of numbers (Line 22). The instance variable 't.timepoints' that shows observed timepoints, and the method 't.get\_concs(species)' shows the time development of concentration of the specified species (Lines 49-51).

Some users may want to obtain the concentration of molecules in the labeled regions. This is enabled by two classes: 'GetLabeledConcs' and 'ConnectLabeledConcs'. The GetLabeledConcs class calculates the numbers of all molecules in all labeled volumes (Lines 29-35). The calculated numbers are saved in the h5 files ('conc\_files'; Lines 23, 32, 35). The saved h5 files are loaded by the GetLabeledConcs class, and converted into a single time series. Users can plot the time development of concentration of target molecules within target regions (timepoints and get\_concs; Lines 45-47).

In the case of FRAP (Lines 7-11), we can observe the FRAP of YFP in the target spine.

In the case of |Ca2+| influx via NMDA receptors (Lines 14-18), we can observe the transient increase in |Ca2+| concentration in the target spine.

.. include:: ../isonum.txt

.. include:: ../isogrk1.txt

.. |Ca2+| replace:: Ca\ :sup:`2+`

==============

Run simulation

==============

Models stored in LM-format files can be executed by Lattice Microbes (LM). LM is not a Python module, but is executed by the command prompt:

.. code-block:: bash

$ mkdir results\_photobleach

$ cp models/photobleach.lm results\_photobleach/0000.lm

$ lm -r 1 -sp -sl lm::rdme::MpdRdmeSolver -f results\_photobleach/0000.lm

The first command makes the directory 'results\_photobleach', and the

second command makes a copy of 'models/photobleach.lm' as 'results\_photobleach/0000.lm', then it is executed by LM in the third command. This process can also be described using a Python script (41\_single\_run.py):

.. literalinclude:: ../../tutorial/1/41\_single\_run.py

:language: python

:linenos:

:caption: 41\_single\_run.py

We can easily run simulation; however, in this simple run, we cannot handle any event functions such as photobleaching and synaptic input. The 'ConnectRun' class can handle those events by connecting multiple runs into a sequential run. Events are inserted just before the unit runs. In '42\_connect\_run.py', the class instance 'r' is created in Line 21, and 'r' accepts many variables such as a label volume file (label\_volume\_file; Line 22), template LM file (template\_lm\_file; Line 25), simulation result directory (output\_dir; Line 26). The variable 'exec\_periods' accepts the Python list that contains the simulation time of each unit run (Line 27), and 'exec\_events' accepts the list of functions, each of which is called just before the unit run (Lines 28). 'event\_params' sets parameters for the functions (Line 29). The class method 'exec' executes the defined connected run (Line 48).

.. literalinclude:: ../../tutorial/1/42\_connect\_run.py

:language: python

:linenos:

:caption: 42\_connect\_run.py

The method 'exec' first calls 'null\_event' (do nothing), and executes a 4-s pre-run. It then calls 'event\_replace', which is described in Lines 6-18, and executes a 4-s post-run. The simulation results are stored in '0000.lm' and '0001.lm' in the directory 'output\_dir'.

All event functions in the 'ConnectRun' class must accept three variables: lattice, sys\_param, and event\_param (Lines 6).

The variable 'lattice' is a numpy 4D array that has a dimension of X |times| Y |times| Z |times| 16, the last of which denotes the slots of molecules (16) at each lattice site. The variable 'sys\_param' conveys the system parameters such as the number of calls (sys\_param['i']; Line 7), current simulation time (sys\_param['time']; Line 8), a dict variable that specifies molecular species id (sys\_param['species']; Line 9), and the label volume (sys\_param['label volume']; Lines 14, 22). The variable 'event\_param' contains the variable that is described in 'event\_params' (Line 29). Here, we obtain the molecular ids of 'YFP' and 'bleached YFP' in the variables 'src' and 'dst', respectively.

Then, the molecules 'YFP' are replaced with the molecules 'bleached YFP' if they are located in the labeled area (Lines 15-17). All the event functions in the 'ConnectRun' must return the two variables: 'lattice' and 'event\_param' (Lines 18). Both of them can be altered, and used for the subsequent simulation and event.

In the case of |Ca2+| influx via NMDA receptors (Lines 32-37), the simulation time is decreased to 2 + 2 s, and 'event\_replace' replaces the molecules 'inactive NMDAR' with the molecules 'active NMDAR'.

============

Build models

============

.. |Ca2+| replace:: Ca\ :sup:`2+`

.. |um2| replace:: |mgr|m\ :sup:`2`

In this subsection, a LM model is built based on the voxelized shape of a spiny dendrite (31\_build\_model.py).

1. Some HDF containers are loaded to define the cytosolic region (vol\_cytosol; Line 21), PSD region (vol\_PSD; Line 22), and the region of cell boundary (vol\_bound; Line 23).

2. The dict variable 'domains' is set to designate the names of each integer in vol\_cytosol (8-bit unsigned int; Line 25), and the dict variable 'surfaces' is set to designate the pairs of a name and voxelized surface areas (float; Line 27).

3. The BuildAnyShape class is called to register the domains and surfaces in the instance variable 'cell' (Line 32).

4. The instance variable 'cell' further incorporates the following properties regarding molecules: names, initial locations, diffusion, and interactions (set\_molecules; Line 33). The contents of 'set\_molecules' are described in the files 'set\_molecule\_FRAP.py' and 'set\_molecule\_Ca.py'. The Python language utilizes triple quotes (|'''|) to comment out a block of code. Thus users can comment out either of them (Lines 7-15).

5. Simulation timers are set under the instance 'sim' (Lines 33-37). Descriptions of setTimestep, setWriteInterval, setLatticeWriteInterval, and setSimulationTime are provided in `the instruction guide of LM <http://faculty.scs.illinois.edu/schulten/software\_manuals/InstructionGuide.pdf>`\_. Indeed, the BuildAnyShape class is a wrapper of a class of LM 'pyLM.RDME.RDMESimulation', all methods of which are preserved in the instance 'sim'.

6. The overall setup is saved in the LM-format file 'models/photobleach\_yfp.lm' or 'models/Ca\_influx.lm' (Lines 7-11 and 40-43).

We here recall two simulation targets: fluorescence recovery after photobleaching (FRAP; left in Figure above) and |Ca2+| influx through NMDA receptors (right in Figure above). To realize the FRAP simulation, the script below (32\_FRAP.py) has a function 'set\_molecules' to introduce two types of molecules: 'YFP' and 'bleached YFP' (define\_species; Lines 3-6), set their initial concentrations (add\_molecule\_uM; Lines 8-11), and diffuse them with a diffusion coefficient (set\_diffusion; Lines 13-16). If we select to call the script '32\_FRAP.py' (Lines 7-11 in 31\_build\_model.py), the definitions of the molecules are incorporated in the instance variable 'cell'.

In the case of |Ca2+| influx via NMDA receptors, the script '33\_CaSignal.py' defines a variety of molecules and their properties. A cytosolic molecule 'Ca' is created to represent |Ca2+| ions. ‘Pump’ and ‘Pump-Ca’ are created for the uptake of |Ca2+| ions at the cell boundary.

To simulate |Ca2+| influx via NMDA receptors, the script below (set\_molecule\_Ca.py) defines a variety of molecules and their properties. First, a cytosolic molecule '\*Ca\*' is created to represent |Ca2+| ions (Lines 4, 9). The initial number of '\*Ca\*' is set to be zero, which is declared using the number representation (add\_molecule; Lines 13-15). Next, two PSD molecules '\*active NMDAR\*' and '\*inactive NMDAR\*' are created (Lines 5-6, 10), and 'inactive NMDAR' is set to have an initial number density of 300 /|um2| (add\_surface\_molecule; Lines 5-6, 16-17). Similary, the cell-boundary molecules '\*Pump\*' and '\*Pump-Ca\*' are created for |Ca2+| uptake (Lines 5-6, 10), and 'Pump' is distributed with a density of 100 /|um2| (add\_surface\_molecule; Lines 5-6, 16-17). Only Ca is set to be diffusible (Lines 19-21).

Then, molecular interactions are modeled (Figure below). The NMDA receptors are set to be rapidly activated at time 0 s, and this abrupt change will be introduced in the process of simulation. The '\*active NMDAR\*' mediates |Ca2+| influx. This is formalized by a first-order reaction, with a rate constant of '\*k\_channel\_Ca\*' (reac\_oneway\_uM; Lines 30, 33-34). The '\*active NMDAR\*' is gradually converted to its inactivated form '\*inactive NMDAR\*' (first-order rate constant: '\*k\_nmdar\_deact\*'; Lines 31, 33-34). Cytosolic '\*Ca\*' is uptaken by a pump to the extracellular space (Figure below). In this process, '\*Ca\*' binds to '\*Pump\*' in a reversible manner (reac\_twoway\_uM; Lines 36-39), and the |Ca2+|-bound pump '\*Pump-Ca\*' releases the |Ca2+| to the extracellular space (first-order rate constant: '\*k\_pump\_Ca\*'; Lines 32-34). These interactions summarize a life cycle of spine |Ca2+| that is a cause of synaptic plasticity.

We can confirm the successful distribution of molecules by executing '32\_show\_molecule.py' (Figure below). You can find this script in $LD\_DIRECTORY/tutorial/1 .

========

Overview

========

Lattice Microbes (LM) is a software package for efficiently sampling trajectories from the chemical and reaction-diffusion master equations (CME/RDME) [#LM1] . LM computes CME/RDME using Graphics Processing Units (GPU) and even multiple GPUs [#LM2] . LM is recently used for the simulation of mRNA splicing in a HeLa cell [#LM3]\_.

LM has been designed for building geometrically-shaped cells. However, LM essentially can incorporate any shapes of cells and cellular structure, as demonstrated in a previous study [#LM4]\_.

LM is designed to simulate cells with geometric shapes, because of limited availability of information on cellular shapes. However, LM originally can incorporate any shapes of cellular boundary or the other cellular structure, as demonstrated in a previous study [#LM4] .

We here developed an extension of LM, which is named Lattice Dendrites (LD), for the efficient incorporation of realistic cellular shapes. In particular, LD is designed to incorporate shapes of segmented images from electron microscopy (EM). LD provides a variety of utility classes/functions for:

#. Conversion of segmented shapes from a software for EM segmentation (reconstruct).

#. Object rotation to minimize their volume in a rectangular solid.

#. Manual annotation using UNI-EM annotator.

#. Surface molecules distribution proportional to surface area.

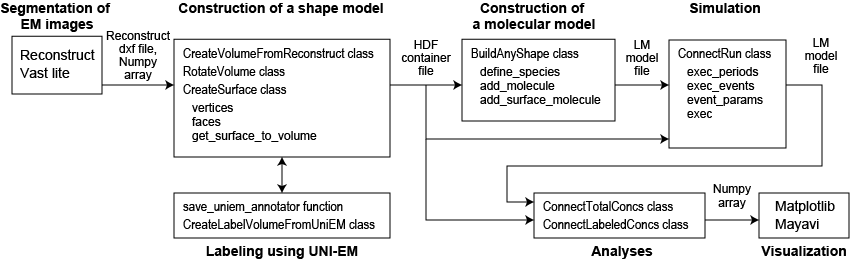
#. Simulation with events

#. Data analyses for visualization.

Users can use a variety of functions below for their model development, simulation, and analyses. Their usages are described in Tutorials 1 and 2. In tutorial 1, we develop a schematic model of dendritic spine, and simulate Ca2+ influx through NMDA receptors and fluorescence recovery after photobleaching (FRAP). In tutorial 2, we introduce a method for the import of realistic shapes of cells from the external software Reconstruct, and annotate it using UNI-EM annotator.

In tutorial 1, we would introduce how to build a geometrically-shaped dendritic spine, and how to simulate Ca2+ influx through NMDA receptors as well as fluorescence recovery after photobleaching (FRAP).

In tutorial 2, we would introduce how to import realistic shapes of cells from the external software Reconstruct, and how to annotate them using UNI-EM annotator.



.. [#LM1] Roberts E, Stone JE, and Luthey-Schulten Z (2013) Lattice Microbes: high-performance stochastic simulation method for the reaction-diffusion master equation, J. Comput. Chem. 34(3):245-255, http://faculty.scs.illinois.edu/schulten/lm/ , http://faculty.scs.illinois.edu/schulten/Software2.0.html#1

.. [#LM2] Hallock MJ, Stone JE, Roberts E, Fry C, Luthey-Schulten Z (2014) Simulation of reaction diffusion processes over biologically-relevant size and time scales using multi-GPU workstations, Parallel Comput. 40:86-99

.. [#LM3] Ghaemi Z, Peterson JR, Gruebele M, and Luthey-Schulten Z (2020) An in-silico human cell model reveals the influence of organization on RNA splicing, PLOS Comput. Biol. 16(3): e1007717, https://eukaryoticcellbuilder.github.io/HeLa\_Builder/

.. [#LM4] Earnest, TM, Watanabe, R, Stone, JE, Mahamid, J, Baumeister, W, Villa, E, & Luthey-Schulten, Z (2017) Challenges of integrating stochastic dynamics and cryo-electron tomograms in whole-cell simulations. J. Phys. Chem. B, 121(15):3871-3881

==============================

Tutorial 1: schematic dendrite

==============================

We would introduce how to use LD and LM through two tutorials. In this tutorial (tutorial 1), we first build a dendritic spine that is represented by a schematic shape (ball and stick), then two types of simulation are conducted: Ca2+ influx through NMDA receptors and fluorescence recovery after photobleaching (FRAP). Through these simulations, users can understand how to build a spiny dendrite, label the spine, run simulation, and visualize the simulation results.

.. toctree::

tut1/create\_spiny\_dendrite

tut1/label\_spine

tut1/set\_reactions

tut1/run\_simulation

tut1/visualization1\_matplotlib

tut1/visualization2\_mayavi

All tutorial programs are located in $LD\_DIRECTORY/tutorial. Find the following script in the subdirectory "1". It provides utility functions that are used in tutorial 1. The function "make\_cylinder" makes a cylinder with an indicated radius and length. The function "add\_shape" locates a geometric object into a volume space. The function "show\_dendrite" loads and displays the surface objects of cellular boundary, postsynaptic density (PSD), mitochondrion, and endoplasmic reticulum (ER).

All tutorial programs are located in $LD\_DIRECTORY/tutorial. Find the following script in the subdirectory "1". It includes utility functions for tutorial 1. The function "make\_cylinder" makes a cylinder with a indicated radius and length. The function "add\_shape" locates such a geometric object in a volume space. The function "show\_dendrite" loads and displays the surface objects of cellular boundary, postsynaptic density (PSD), mitochondrion, and endoplasmic reticulum (ER).

.. literalinclude:: ../tutorial/1/tut1\_functions.py

:language: python

:linenos:

:caption: tut1\_functions.py

.. include:: ../isonum.txt

.. include:: ../isogrk1.txt

==============

Create a shape

==============

1. We first draw a blueprint of a spiny dendrite as a combination of geometric shapes (Figure below). A sphere (radius: 0.25 |mgr| m) represents a spine head, which is partially labeled as a postsynaptic density (PSD; red colored area). The spine head has a cylindrical spike neck (radius: 0.1 |mgr| m, length 1.0 |mgr| m), and it is further connected to a parent dendrite that also has a cylindrical shape (radius: 0.5 |mgr| m, length 1.8 |mgr| m). Not only the contour of spiny dendrite, we also introduce two types of intracellular organelles. Endoplasmic reticulum (ER) is set as thin cylinders (radius: 0.08 |mgr| m, length 1.8 |mgr| m; blue colored objects), and a mitochondrion is set as a thich cylinder (radius: 0.2 |mgr| m, length 1.8 |mgr| m; yellow colored objects).

|

.. image:: imgs/Scheme.jpg

:scale: 60%

:align: center

|

2. The designed shape is embedded in a voxel space for simulation (Lines 1-35, 11\_create\_dend.py). First, a voxel space of 96 |times| 60 |times| 96 voxels (20 nm/voxel) is set as a numpy 3D array 'vol\_dend' (Line 14). Then, the spine head is created using a Python module function morphology.ball (skimage; Line 10; 0.25 |mgr| m, 12 voxels), which is added to 'vol\_dend' (add\_shape; Line 15). Similarly, we make the spine neck and dendrite as cylinders (Lines 11 and 12, respectively), and add them to 'vol\_dend' (Lines 16 and 17, respectively). The in-house function 'add\_shape,' simply overlays overwrapped regions, and represents the filled areas as 1 (cytosolic region) and the void areas as 0 (extracellular region). Thus, we have already built the contour of the spiny dendrite in 'vol\_dend.' Similarly, the PSD, Mitochondrion, and ER are embedded in vol\_psd, vol\_mito, and vol\_er, respectively.

|

3. Here, the volume sizes need to be corrected. LM can only simulate the volume with a size of a multiple of 32 |times| 32 |times| 32 voxels, whereas the current voxel size is 96 |times| 60 |times| 96 voxels. We thus execute a utility function 'lmpad' (Lines 39-42). The voxel sizes are automatically expanded to the multiple of 32 (96 |times| 64 |times| 96 voxels in this case). Users can of course set a multiple of 32 voxels from the beginning.

.. literalinclude:: ../../tutorial/1/11\_create\_dend.py

:language: python

:linenos:

:caption: 11\_create\_dend.py

4. Then, we use the CreateSurface class to generate smoothed surfaces of each object (Lines 53, 63, and 67). The generated surfaces are required for locating surface molecules as well as visualizing simulation results. Each surface is composed of triangles that are specified by three vertices (Lines 55 and 56, respectively). Also, the CreateSurface class can generate the surface areas per volume. LD can distribute surface molecules in the voxel space, depending on the surface areas per volume (Line 56). We can further select the surface triangles within the areas of PSD (face\_id\_psd, Lines 58, 59) to distribute molecules only in this area.

|

5. Finally, the generated variables are assembled in a Python dictionary 'm' (Lines 51-72), and 'm' is saved into the HDF container file 'models/ball\_and\_stick.h5' (Lines 75-80).

.. literalinclude:: ../../tutorial/1/12\_show\_dend.py

:language: python

:linenos:

:caption: 12\_show\_dend.py

.. image:: imgs/ball\_and\_stick.png

:scale: 50%

:align: center

6. Execute 'python3 11\_create\_dend.py'. If users have saved the volumes and surfaces successfully, the execution of the subsequent script 'python3 12\_show\_dend.py' will show its 3D shape (Figure above). We use this spiny dendrite for simulation.

For analyses, we often need to obtain molecular concentration of a specific region, such as a spine. To enable this, we label a spine volume. It is easy to do it in the case of the geometrically shaped cells, because users can label a specific area by re-defining the same sphere in the same space. In the sample script (‘12\_label\_head.py’), the total cytosolic volume ‘vol\_dend\_not\_mito\_not\_er’ is loaded from ‘models/ball\_and\_stick.h5’, and saved as the container ‘ref volume’ in the label file ‘models/labels\_ball\_and\_stick.h5’. Similarly, the label id is saved as the container ‘label ids’.

If users can successfully save the spine label, they can see it using the sample script ‘22\_show\_label.py’. The labeled area is shown as a colored part.

It is not easy to label specific regions in the case of a morphologically realistic model. We will try it in tutorial 2.

to obtain molecular concentration of this region. Because the spine has a geometric shape, we can programmably label the spine volume, not using the UNI-EM annotator, as follows.