Chapter 9

Rapid Creation, Monte Carlo Simulation, and Visualization of Realistic 3D Cell Models

Jacob Czech, Markus Dittrich, and Joel R. Stiles

Summary

Spatially realistic diffusion-reaction simulations supplement traditional experiments and provide testable hypotheses for complex physiological systems. To date, however, the creation of realistic 3D cell models has been difficult and time-consuming, typically involving hand reconstruction from electron microscopic images. Here, we present a complementary approach that is much simpler and faster, because the cell architecture (geometry) is created directly in silico using 3D modeling software like that used for commercial film animations. We show how a freely available open source program (Blender) can be used to create the model geometry, which then can be read by our Monte Carlo simulation and visualization software (MCell and DReAMM, respectively). This new workflow allows rapid prototyping and development of realistic computational models, and thus should dramatically accelerate their use by a wide variety of computational and experimental investigators. Using two self-contained examples based on synaptic transmission, we illustrate the creation of 3D cellular geometry with Blender, addition of molecules, reactions, and other run-time conditions using MCell's Model Description Language (MDL), and subsequent MCell simulations and DReAMM visualizations. In the first example, we simulate calcium influx through voltage-gated channels localized on a presynaptic bouton, with subsequent intracellular calcium diffusion and binding to sites on synaptic vesicles. In the second example, we simulate neurotransmitter release from synaptic vesicles as they fuse with the presynaptic membrane, subsequent transmitter diffusion into the synaptic cleft, and binding to postsynaptic receptors on a dendritic spine.

Key words: Blender, MCell, DReAMM, MDL, Cell modeling, Cell architecture, Cell geometry, Stochastic, Diffusion-reaction.

1. Introduction

A quantitative understanding of cell and tissue function requires detailed models through which hypotheses may be generated and tested. As models become more complex, computer simulations provide tests of important questions that are beyond simple intuition

Ivan V. Maly (ed.) *Methods in Molecular Biology, Systems Biology, vol. 500*

DOI: 10.1007/978-1-59745-525-1_9

and are inaccessible to current experimental methods. In the best case, a tight coupling between models, simulations, and experiments can lead to breakthroughs in understanding.

Model development and simulation involve a number of distinct steps carried out with different software tools, similar to the different steps, methods, and equipment used in an experimental protocol. Realistic physiological models are particularly challenging because they encompass complex biochemistry taking place in small complex 3D spaces, and the different software tools required at each step can present steep learning curves. Nevertheless, it is increasingly necessary to develop and use such models to understand the physiology of disease, drug effects, and phenotypic changes produced by genetic manipulations.

Creation of a spatially realistic model begins with the definition of its cellular architecture, or geometry, followed by the addition of biochemical species and interactions within the geometry. The methods chosen for these initial steps depend in large part on the simulation approach to follow. For example, while some models may represent an entire cell as a single well-mixed compartment, we focus on stochastic diffusion and reactions within arbitrarily complex intra- and extracellular spaces. Triangulated surface meshes are used to represent cell and organelle membranes, and thus the meshes also define different volumes of intervening solution. To build a model, one must somehow create the surfaces and then define how many molecules of what types are present in different spatial regions. One must also provide the diffusion coefficient for each molecular species and define the network of biochemical interactions and associated rate constants. Having done so, one can investigate such problems as diffusion of neuromodulators and neurotransmitters through tortuous extracellular space in brain *(1–3)*, or neurotransmitter release (exocytosis) and synaptic transmission *(4–11)*. Simulations of exocytosis involve voltage- and/or calcium-triggered release of signaling molecules from synaptic or endocrine vesicles. The released molecules subsequently diffuse through some extracellular space and are detected by receptor protein molecules on the downstream cell or cells. Our program MCell (Monte Carlo Cell; **refs.** *4, 6, 7, 12)* was designed for such simulations, although it now is very general and can be used for a wide variety of diffusion-reaction models (e.g., **refs.***13, 14)*. The companion program DReAMM is used to visualize and animate the simulation results *(9, 12, 15)*.

In this chapter we present a protocol for development of an MCell model and simulations, providing step-by-step instructions for an example based on signal transduction at synaptic boutons. Creation of the initial 3D geometry has been a longstanding bottleneck for all such projects, especially when synaptic geometries are extracted (segmented) from electron microscopy data and subsequently converted into surface meshes for use in

simulations *(6, 9, 10)*. Recently, however, we have developed a complementary and much faster approach based on the use of Blender *(16)*, open-source 3D modeling software (*see* **Note 1**), as well as plug-ins that we have developed to link Blender to MCell and DReAMM. As we illustrate here, arbitrary cell-like geometry can be designed directly *in silico* using Blender, and the resulting meshes can be exported to MCell and DReAMM for simulation and visualization. This time-saving approach makes detailed simulations and investigations available to virtually any laboratory in relatively short order.

Specifically, we show how to do the following:

1. Create the model geometry using Blender. In **Subheading 3.1**, a synapse is created on one of several dendritic spines (**Fig. 1**), similar to actual dendrite and spine ultrastructure (**Fig. 1**, inset). A synaptic cleft space separates the presynaptic bouton from the spine head, and the bouton contains two docked synaptic vesicles (*see* **Fig. 4E**). Particular regions of the vesicle membrane and pre- and postsynaptic membranes (*see* **Fig. 4F, G**) are defined for subsequent addition of calcium

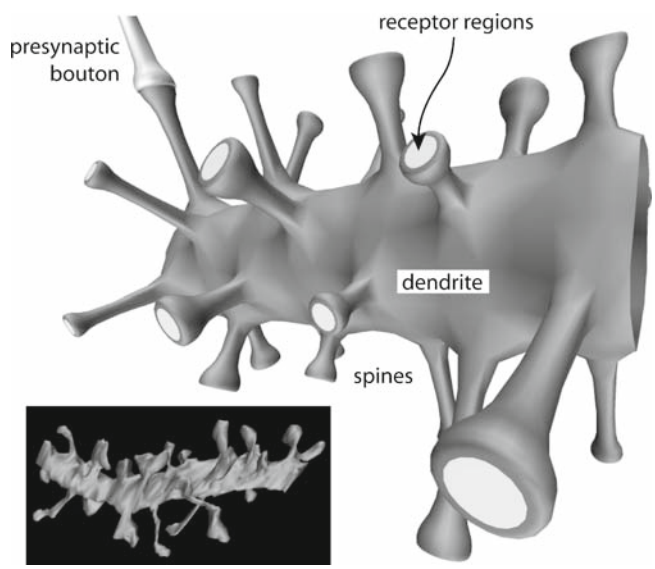

Fig. 1. Three-dimensional model of a spiny dendrite illustrates some of the structures to be created and used in simulations in this chapter. The model dendrite was created with Blender and visualized with DReAMM. Postsynaptic receptor regions (*light gray*) are located at the ends of the spine heads. One of the spine heads forms a synapse with a presynaptic bouton (*upper left*). For comparison, the inset shows a dendrite reconstructed from serial electron micrographs of rat brain. The data for the reconstruction were obtained as a publicly available VRML file from **ref.** *17*. The VRML data were subsequently imported into DReAMM and visualized as illustrated by the inset image.

binding sites, voltage-gated calcium channels (VGCCs), or ligand-gated neurotransmitter receptors, respectively, using MCell's MDL (**Subheadings 3.3** and **3.5**). In **Subheading 3.2**, the synaptic vesicles are modified with Blender to include an expanding fusion pore (*see* **Fig. 4H, I**), and the model is subsequently used for MCell simulations of neurotransmitter diffusion through the pore (**Subheading 3.5**).

2. Use Blender plug-ins to export the model geometry as triangulated surface meshes with region annotations for use with MCell and DReAMM.
3. Use MCell's MDL to specify (**Subheadings 3.3** and **3.5**) the following:
   (a) The types of molecules in the model (calcium ions, calcium binding sites on proteins, neurotransmitter molecules, postsynaptic receptor proteins), their diffusion constants, and their initial locations.
   (b) The stoichiometry, rates, and directionality of the stochastic reactions that can occur during simulations (conformational changes of calcium channels, calcium entry from open channels, calcium binding to protein sites on synaptic vesicles, neurotransmitter binding to postsynaptic receptors, channel opening and closing of bound receptors).
   (c) The reaction data to be saved when the simulations are run. This includes counts of molecules and reactions in specific compartments as a function of time, and the resulting text (ASCII) files can be analyzed or plotted using common graphing software and/or scripts.
4. Save MCell visualization data for DReAMM so that the model can be rendered (displayed) and checked. The importance of this step cannot be overemphasized, especially for complex models. For efficiency, the visualization files are written in a structured binary format that minimizes disk space and maximizes interactive speed and simplicity of use with DReAMM.
5. Run complete MCell simulations and visualize the results with DReAMM. In **Subheading 3.3**, we simulate a voltage-clamped presynaptic bouton in which calcium channels open stochastically, allowing calcium ions to enter. The ions then diffuse and bind to sites on the synaptic vesicles, simulating the presence of calcium-binding proteins (e.g., synaptotagmin) and some of the events leading to calcium-dependent neurotransmitter release (exocytosis). In **Subheading 3.5**, we simulate expansion of fusion pores between the vesicles and presynaptic membrane, with subsequent diffusion of neurotransmitter molecules into the synaptic cleft. Within the cleft,

neurotransmitter binds reversibly to the postsynaptic receptors. Receptors that reach a double-bound state can undergo a reversible conformational change to an open channel state (ligand-dependent channel gating).

6. Use DReAMM to animate simulation results.

The projects illustrated in this chapter are completely self-contained and can be used without reference to additional material. Many modeling features are illustrated, although spatial and biochemical details have been simplified for the sake of space and readability. Thus, the chapter provides an efficient introduction to development and use of spatially realistic stochastic cellular models and simulations, and, where necessary, provides links and citations to further discussion and examples.

2. Materials

2.1. Preliminary Issues

Building and simulating the models described in this chapter requires a computer with the programs Blender, MCell, and DReAMM installed. Like many open source programs, Blender, MCell, and DReAMM are available in precompiled binary executable form for a limited set of computer architectures and operating systems. This can dramatically simplify installation for users who have access to those particular systems and do not have experience in compiling ("building"), installing, and administering (large) programs obtained as source code. However, it is not possible to "prebuild" for all possible systems, and for this and other reasons it may be necessary or preferable to obtain the source code and build it "from scratch". This is generally not a problem for a UNIX user with experience in code development or system administration, but it can be challenging for a novice. Part of the difficulty is simply dealing with a command line interface (typing commands at a prompt in a window), understanding where system files are located (in which directories or "folders"), and understanding how directory and file access is granted or denied to different users on a multiuser system. This is a very real issue, especially as more and more experimental biologists are becoming more and more interested in using computational models and simulations to complement their wet lab work. Although the basics are straightforward, entire books are dedicated to the use and care of UNIX systems. In this chapter our focus is on building and using spatially realistic models. Hence, we assume a basic knowledge of UNIX and a command line interface, just as an experimental protocol must assume a basic knowledge of solutions, gels, radiochemicals, etc.

2.2. Computer Hardware and Operating System

The computer to be used can have either a 32- or 64-bit architecture (single or multiple cores), and should have some form of a UNIX operating system that includes developer features (C/C++ compiler, OpenMotif libraries, standard image tool libraries, etc.). A fully configured Linux or similar system should work without problem. Mac OS X is in fact a form of UNIX, so it also works well, but it must be installed with the developer libraries (*see* **Note 2**). The difference between a 32- or 64-bit system is unimportant unless very large MCell simulations are to be run, in which case a 64-bit system with a very large amount of physical memory (RAM) may be required. At the upper extreme for large models, the simplest present solution is to run MCell on a large shared memory architecture (hundreds of GBytes), and if no such local computers are available one can use a shared memory computer available at the Pittsburgh Supercomputing Center or other national facility. In practice, however, many models and simulations (including those for this chapter) can be run quite easily on present-day desktop computers with typical amounts of memory (e.g., 2–8 GB).

Efficient and high-quality visualization is critically important to spatially realistic modeling, and so the computer should have at least one high-resolution display (1,600 × 1,200 or higher). We routinely use two such displays configured as one 3,200 × 1,200 workspace, driven by a professional-level OpenGL graphics card (e.g., NVidia Quadro 3xxx or 4xxx series at present). Disk storage is generally not a problem in current desktop systems, where capacities approaching a terabyte or more are common. In principle, many projects can also be run on a laptop, although limited display space can be an inconvenience.

2.3. Download and Install Blender and Blender Plug-Ins

1. Download and install Blender from http://www.blender.org. This is the Blender development site, so the most recent version will be available. As an alternative, you may download all the software required for this chapter from https://www.mcell.psc.edu/download.html, in order to obtain the specific versions that match those illustrated here. This chapter is based on Blender version 2.45, and more recent versions may have changes to hot key (and other) functions. In addition, different versions of UNIX may assign different functions to particular keys (such as the *Alt* key), so there may be some inevitable differences from what is described in the following sections.
2. After installation, make sure that your path environment variable is set so that Blender can be started from any command line (type "blender" and hit *Enter* at a command line).
3. Download the two Blender plug-ins from https://www.mcell.psc.edu/download.html if you have not already done so. One

plug-in will be used to generate MCell MDL files (.mdl) from meshes created with Blender. The other can be used to generate DX files (.dx) for use with DReAMM (*see* **Note 3**). The DX format is also the default visualization file format output by MCell for use with DReAMM.

4. To make the Blender plug-ins functional, create a directory and copy or move both plug-ins into it. Start Blender and mouse over the lower edge of the top menu bar (**Fig. 2A**). Left click and pull down to uncover several button fields including one labeled "File Paths". Click on it and then locate the "Python" text box (**Fig. 2B**), which has two buttons next to it. Click the rightmost button, navigate to the plug-in directory you just created, and then select it as the default script location. Then click on the left button to make the scripts available to Blender. If desired, hide the pull down button fields.
5. Hit *Ctrl-u* to save these settings for future use.
6. Verify that on the Blender menu bar, "File –> Export" now shows entries for MCell (.mdl) and DReAMM (.dx).

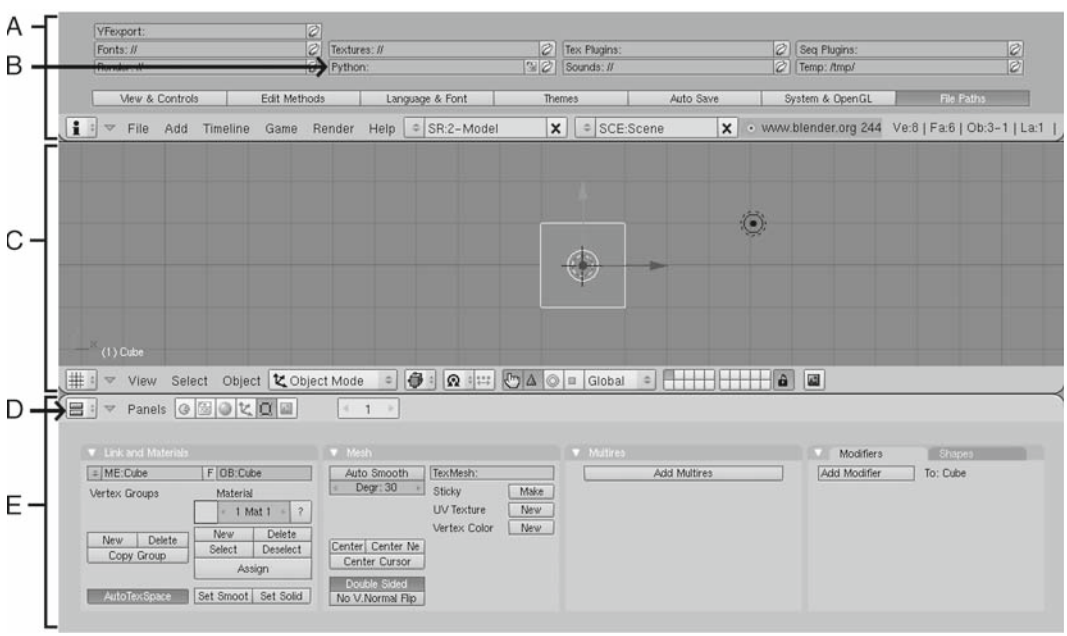

Fig. 2. Screen capture of Blender's main window. By default, the Blender interface consists of three areas (A, C, and E). At the top (A) is the "User Preferences" window, which is hidden initially. It allows the user to adjust Blender's appearance, performance, and file paths, including the location of the Python plug-ins directory (B). The "3D View" (C) is used for manipulation and visualization of mesh objects. The "Buttons Window" (E) offers a variety of operations that can be performed on the mesh objects. At the top left of the "Buttons Window" is a drop-down list (D) providing access to all of the available window areas.

2.4. Download and Install MCell

1. Download and install MCell from https://www.mcell.psc.edu/download.html. Throughout this chapter we assume the use of MCell version 3, or MCell3 *(12)*.
2. After installation, make sure that your path environment variable is set so that MCell can be started from any command line. To do so, enter the name of your executable for MCell at a command line (e.g., "mcell3"; the actual name may vary depending on your installation) and verify that MCell starts running and prints out a number of default start-up messages in the command window. In the absence of any command line arguments/options (as in this case), the start-up messages will include an error message stating that no MDL filename has been specified. This is normal and can be ignored.

2.5. Download and Install DReAMM

1. To use DReAMM you must first download and install PSC_DX, a visual programming, imaging, and data manipulation environment on which DReAMM is built. PSC_DX is a customized and improved version of OpenDX, or Open Data Explorer, originally developed by IBM. Note that DReAMM requires PSC_DX and will not run with OpenDX. Both PSC_DX and DReAMM can be downloaded from https://www.mcell.psc.edu/download.html. This chapter is based on PSC_DX and DReAMM version 4.1.
2. Compile (if necessary) and install PSC_DX.
3. To verify the installation, enter "dx" at a command line. A PSC_DX (Data Explorer) start-up menu and DReAMM splash image should appear.
4. Quit PSC_DX (click on the Quit button in the start-up menu).
5. Install DReAMM. Make sure that the start-up script ("dreamm") or a link to the start-up script is accessible in your path.
6. To verify the installation, enter "dreamm" at a command line. The DReAMM start-up menus (**Fig. 3**), splash image, and "DReAMM Image Window" should open. Click the Play button on the "Sequence Control" menu (**Fig. 3B**) to see a default time series in the "Image Window". Change the pop-up "Keyframe Mode" button (bottom of "Quick Controls" menu, labeled *8* in **Fig. 3A**) from "All Interactive" to "Keyframes" and then replay the time series to see it with animated camera positions. To quit DReAMM, go to the "DReAMM Image Window" menu bar and click on "File –> Quit". When prompted to save the project, click "No".

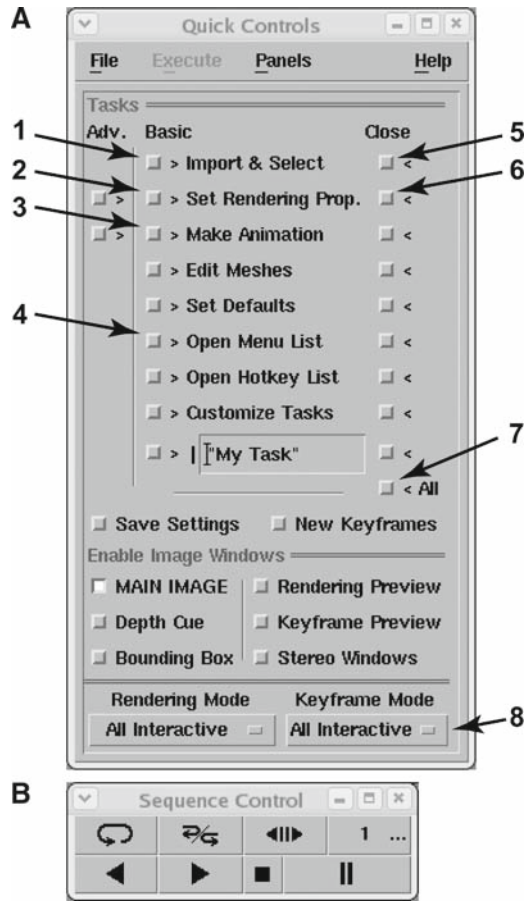

Fig. 3. Screen captures of start-up DReAMM controls. (A) The "Quick Controls" window organizes DReAMM's menus into task-based subsets that can be customized to suit user preferences. (1) Open menus to import data and select objects. (2) Open menus to assign rendering properties to objects. (3) Open menus to make an animation using keyframes. (4) Open a list of all menus that can be opened one-by-one. (5) Close the menus opened by (1). (6) Close the menus opened by (2). (7) Close all previously opened menus. (8) By default, the "Keyframe Mode" is set to "All Interactive", meaning that the camera can move about the 3D space freely. When set to "Keyframes", however, the camera will be locked into positions (keyframes) that were set using "Make Animation" menus. (B) "Sequence Control" for animations. Bottom row of buttons, left to right: Play in reverse, Play forward, Stop, Pause. Top row, left to right: Loop while either Play button is pressed, Palindrome (play to end or beginning and then reverse direction while either Play button is pressed), Single-step play mode, Frame number (pressing this button opens a "Frame Control" window that allows particular frames to be played). Note that Loop, Palindrome, and Single-step may all be used in combination.

3. Methods

3.1. Create Pre- and Postsynaptic Geometry with Blender

3.1.1. Create a Spine Head

In this section we will create a spine head by removing the upper portion of a sphere and then closing the opening (**Fig. 4A, B**). In later sections, the spine head will be copied and modified to create additional objects.

1. *Start Blender*: Enter "blender" at a command line. You will see two view ports, a large central "3D View" and a "Buttons Window"

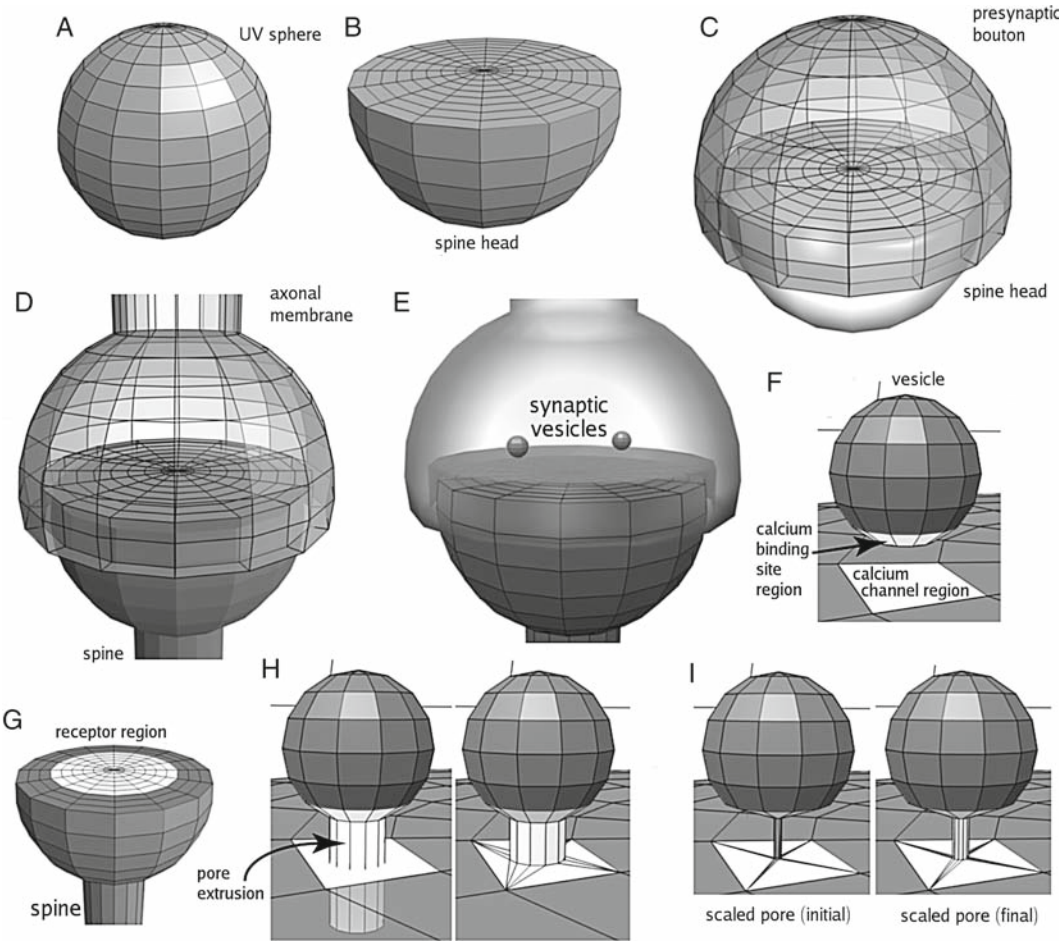

Fig. 4. Blender screen captures at key stages during mesh creation. A UV sphere (**A**, *see* **Note 6**) is cut nearly in half and closed off to create a dendritic spine head (**B**). The spine head then is duplicated, rotated, and manipulated to create a presynaptic bouton (**C**). Portions of the bouton and spine head are extruded to create axonal membrane and the spine neck (**D**). Two small spheres are added within the bouton and serve as synaptic vesicles (**E**). Regions on the presynaptic bouton and vesicle meshes are defined for voltage-gated calcium channels and calcium binding sites, respectively (**F**). A region on the spine head is defined for postsynaptic receptors (**G**). A fusion pore is extruded from each synaptic vesicle (**H**, *left*) and joined to the presynaptic bouton mesh (**H**, *right*). Finally, the fusion pore is scaled to the desired initial (**I**, *left*) and final (**I**, *right*) diameters.

widget at the bottom (**Fig. 2**). By default, the start-up 3D view shows the XY-plane parallel to the screen (*see* **Note 4**).

2. *Delete the default Blender start-up object*: At start-up, Blender creates a simple cube object. Delete this by hitting *x* (*see* **Note 5**), which will bring up a dialog box prompting to "Erase selected Object(s)". Click to confirm.

3. *Create a sphere*: Hit the *spacebar*. In the pop-up menu, mouse over "Add", then "Mesh", and click on "UVSphere" (**Fig. 4A**, **Note 6**). In the new pop-up menu titled "Add UV Sphere", make sure that "Segments" and "Rings" are set to "16", "Radius" is set to "0.25" (*see* **Note 7**), and then click "OK".

4. *Name the sphere*: Under the "Link and Materials" tab in the "Buttons Window", click in the text box that says "OB:Sphere" and change it so that it reads "OB:SpineHead".
5. *Change view*: Hit *1* (*on the number pad*) to switch to the XZ-view (*see* **Notes 8** and **9**).
6. *Deselect the sphere and make it semitransparent*: Hit *a* then *z*.
7. *Select the vertices to be removed*: Hit *b*, which will bring up a pair of cross-hairs used for selecting vertices. Clicking and dragging will create a rectangular area that follows the cross-hairs. Any vertices within this rectangle will be selected. Click and drag over all the vertices *above* (but not including) the equator of the sphere (the XY-plane in this view).
8. *Remove the faces that make up the top of the sphere*: Hit *x* and click on "Faces" in the "Erase" menu. The "3D View" should now show only the remaining lower portion of the sphere.
9. *Close the opening*: As shown in **Fig. 4B**, the desired result includes a set of new adjoining faces that will meet at a central vertex and close the opening (*see* **Note 10**). Begin by selecting the topmost vertices. Hit *b* and select the vertices by clicking and dragging the selector rectangle over the top edge. Next, hit *e* to extrude and click "Only Edges" in the "Extrude" pop-up menu. Hit *0* as the distance to extrude and hit *Enter* to confirm. Hit *s* to scale the extrusion, next hitting *0* to obtain the desired new radius, and then hit *Enter*. In the "Buttons Window" under "Mesh Tools", hit "Rem Double". This will remove all but one of the duplicated vertices and reconnect the triangles. Blender should inform you that 15 vertices have been removed, and the object should now be closed by a flat top.
10. *Subdivide the triangles that close the top*: At this point we need to create a set of concentric rings to be used in several subsequent operations (defining a region and creating an invagination). Hit *b* and once again select the topmost vertices, which now will also include the new central vertex. Next hit *7* (*on the number pad*) to change back to the XY-view (overhead view of the spine head). Hit *k* and, in the pop-up "Loop/Cut Menu", click "Knife (Multicut)". In the "Number of Cuts:" pop-up menu, click the right arrow until it reads "8" and then click "OK". In a clockwise motion, click and drag the knife-shaped cursor over all of the spoke-like edges radiating from the center point. All of the edges (spokes) need to be crossed; it does not matter where or in which order. After crossing all the edges, hit *Enter*. Blender will subdivide each edge into eight segments of equal length and create new coplanar faces arranged in concentric rings (**Fig. 4B**).

11. *Save your current mesh object*: Hit *F2* and save the mesh (*see* **Note 11**).

3.1.2. Create a Presynaptic Bouton

In this section we will duplicate the existing spine head and morph the copy into an invaginated presynaptic bouton.

1. *Reset the view*: Hit *1* (*on the number pad*) to switch again to the *XZ*-view. If you are continuing from the previous section and some vertices are currently selected, hit *a* once to deselect everything. Then hit *a* (again) to select all the vertices.
2. *Duplicate the spine head and rotate the copy*: Hit *Shift-d* to duplicate the existing spine head. After it is duplicated, Blender will automatically select only the new portion, which at this point is still considered part of the original object. Hit *r*, then type *180*, and hit *Enter* to rotate the new portion by 180° around the Y-axis (the current view axis). Hit *p* and click on "Selected" in the "Separate" menu to make the new rotated portion a separate object.
3. *Select and name the new object*: Hit *Tab* to switch to "Object Mode" and right click on the new object so that it is highlighted in pink (*see* **Note 12**). In the "Link and Materials" tab in the "Buttons Window", name the object by clicking in the text box containing the string "OB:SpineHead.001" and changing it to "OB:PresynapticBouton".
4. *Shift and scale the bouton*: Hit *g* to grab the object, *z* to constrain the shift (move) operation to the *Z*-axis, then type *0.15*, and hit *Enter* to confirm. This will move the bouton 0.15 units along the positive *Z*-axis. Hit *s* to scale the bouton, type *1.2*, and hit *Enter* to confirm. This will increase the size of the bouton by 20%.
5. *Create an invagination in the bouton*: Hit *Tab* again to switch to "Edit Mode". Hit *a* to deselect everything. Select the lowermost vertices of the bouton by hitting *b* and then clicking and dragging around them. Next, hit *Ctrl-Minus* (*on the number pad*) to perform a "Select Less" operation. In this case the outermost ring of vertices will be deselected. Hit *e* to extrude, click "Region" in the "Extrude" menu, *z* to constrain it to the *Z*-axis, type *0.075*, and hit *Enter*. This will move the remaining selected vertices 0.075 units in the positive *Z* direction, producing an invagination into which the postsynaptic spine head will fit (**Fig. 4C**).
6. *Save the current mesh objects*: Hit *F2* to save the current meshes (*see* **Note 11**).

3.1.3. Add Axonal and Dendritic Extensions

We now add cylindrical extensions to the rounded ends of both objects, to create a length of axonal membrane for the presynaptic bouton and a dendritic spine for the spine head.

1. *Confirm that you are in "Edit Mode"*: If you are continuing from the preceding step, you should already be in "Edit Mode". If necessary, hit *Tab* to switch from "Object Mode" to "Edit Mode".
2. *Confirm the view*: The operations to be performed in this step are most safely accomplished using a "side view", so if necessary hit *1* (*on the number pad*) for the *XZ*-view.
3. *Create a cylindrical axon segment on the presynaptic bouton*: If necessary, hit *a* to deselect all objects, then zoom in (*see* **Note 8**) on the upper pole of the presynaptic bouton until the top vertex is clearly visible. Select the vertex by right clicking on it. Now perform two "Select More" operations by hitting *Ctrl-Plus* (*on the number pad*) twice, until two concentric rings of vertices are highlighted. Hit *x* and click "Faces" on the "Erase" menu. This will remove the mesh faces selected at the pole of the presynaptic bouton, leaving an opening. Hit *b* and then select the vertices that line the opening by clicking and dragging around them. Next hit *e*, click "Only Edges" on the "Extrude" menu, then hit *z*, type *3.0*, and hit *Enter*. A cylindrical axon segment should now extend from the top of the bouton (**Fig. 4D**).
4. *Select the spine head*: Hit *Tab* to go into "Object Mode". Right click on the spine head (bottom) mesh to select it. Hit *Tab* to go back into "Edit Mode".
5. *Create a cylindrical spine on the spine head*: As in earlier **step 2**, zoom in and select the bottommost vertex (at the pole) by right clicking on it. Hit *Ctrl-Plus* (*on the number pad*) two times. Hit *x* and click "Faces" on the "Erase" menu. Hit *b* and then click and drag around the vertices that line the hole in the bottom of the mesh. Hit *e*, click "Only Edges" on the "Extrude" menu, hit *z*, type *–2.0*, and hit *Enter*. A cylindrical spine should now extend from the bottom of the spine head (**Fig. 4D**).
6. *Save the current mesh objects*: Hit *F2* to save the current meshes (*see* **Note 11**).

3.1.4. Add Synaptic Vesicles with Regions for Calcium Binding Sites

We will now create two small spheres to represent synaptic vesicles inside the presynaptic bouton. We will also define a region on each vesicle to contain calcium binding sites (**Fig. 4E, F**). In Blender, we define regions simply by assigning each one a uniquely named material. If desired, the regions can then be visualized by giving them separate colors. The regions assigned with Blender will be accessible automatically with MCell and DReAMM.

1. *Create the first synaptic vesicle*: New objects always appear at the position of the 3D cursor. Make sure that the cursor is now positioned at the origin by hitting *Shift-c*. If you are continuing from the previous section, hit *Tab* to enter "Object Mode". Hit the

spacebar, mouse over "Add", then "Mesh", and click on "UVSphere". In the "Add UV Sphere" menu, change "Segments" to "12", "Rings" to "8", "Radius" to "0.02", and then click "OK". This creates a small sphere centered at the origin.

2. *Rotate the vesicle into a more convenient orientation*: Hit *r,* then *x*, type *90*, and hit *Enter* to rotate the sphere 90° around the *X*-axis.
3. *Define the region for calcium binding sites*: Under "Link and Materials", click "New" (the rightmost option) to assign material properties to the whole vesicle. Default settings and a default name will be used automatically. Next hit *a* to deselect everything and then right click on the bottommost vertex of the vesicle to select only that vertex. Hit *Ctrl-Plus* (*on the number pad*) two times to select two additional concentric rings of vertices. Again click "New" under "Links and Materials" to create a second material. The words "2 Mat 2" appear in a box directly above. Click in the gray square to the left of these words, and a color selector will appear. The color selector includes a horizontal color bar and a larger saturation-value gradient box. Click in the upper right-hand corner of the gradient box so that a reddish shade is selected. Now click "Assign" to apply this red material to the selected faces.
4. *Name the new region*: Hit *F5* to change to the "Shading" section of the "Buttons Window", and, under "Links and Pipeline" in the text box titled "Link to Object", change the field that reads "MA:Material.002" (*see* **Note 13**) to "MA:CaBS_Reg" for the calcium binding sites region. Hit *F9* to return to the "Editing" section of the "Buttons Window".
5. *Move the vesicle*: Hit *Tab* to change into "Object Mode". Hit *g* to grab the entire vesicle, then *x*, type *–0.108*, and hit *Enter* to move the vesicle –0.108 units along the *X*-axis. Hit *g,* then *z*, type *0.105*, and hit *Enter* to move it 0.105 units along the *Z*-axis.
6. *Duplicate the vesicle and move the copy*: Hit *Shift-d* to duplicate the vesicle. Hit *g*, then *x*, type *0.216*, and hit *Enter* to move the copy 0.216 units along the *X*-axis away from the original.
7. *Rotate the vesicles into their final positions*: We want each vesicle situated above a single quadrangular face (quad) of the presynaptic mesh, so that the two quads can be defined as a region for VGCCs (*see* **Subheading 3.1.5** later). Thus, we now rotate the vesicles around the *Z*-axis. With the vesicle copy still selected, hold *Shift* and right click on the other vesicle to select it. Hit *r*, then *z*, type *11.25*, and hit *Enter* to confirm.
8. *Name the two vesicles*: Select the left vesicle by right clicking on it. To name it, look under "Link and Materials", click in the text box that reads "OB:Sphere", and change it to "OB:Vesicle1". To name the vesicle on the right, right click

on it and change the text box from "OB:Sphere.001" (*see* **Note 13**) to "OB:Vesicle2".

9. *Save the current mesh objects*: Hit *F2* to save the current meshes (*see* **Note 11**).

3.1.5. Define Region for VGCCs

We now define a presynaptic membrane region in which VGCCs will be located. This region will include two noncontiguous areas that underlie the synaptic vesicles. To expedite this operation, we will temporarily "clip" the existing meshes, i.e., make portions of them invisible. This will make it easier to see the particular remaining mesh faces that we want to include in the new regions.

1. *Clip the existing mesh objects*: Select the mesh for the presynaptic bouton by right clicking on it, and then hit *Tab* to change into "Edit Mode". Hit *Alt-b* to bring up cross-hairs for defining a clipping box. To define the box, click and drag a rectangle around both vesicles and the vertices on the presynaptic membrane directly beneath them. The rectangle will project through the plane of the screen to create a clipping box. Everything outside of the box will be invisible until *Alt-b* is hit again.
2. *Change the view*: Hit *7* (*on the number pad*) to switch to the *XY*-view (overhead). You should now be looking down on the synaptic vesicles above the presynaptic membrane. Hit *Ctrl-Tab* and select "Faces" in the "Select Mode" box that appears. Hit *z* to make the mesh opaque, so that subsequent color changes for regions will be easily visible.
3. *Define the region for VGCCs*: Click "New" under "Link and Materials" to assign a default material to the entire object. Then right click on the face directly below one of the vesicles (**Fig. 4f**; the face will extend beyond the vesicle's diameter). While holding *Shift*, right click on the corresponding face under the other vesicle (as noted earlier, the faces in a region do not need to be contiguous). Again, click "New" under "Link and Materials". Click in the gray box beside the words "2 Mat 2". Click in the green area of the horizontal color bar that appears, and then click near the upper right hand corner of the saturation-value gradient box above it. Now click "Assign", and the faces under the vesicles will change to a green color.
4. *Name the region*: Hit *Tab* to change into "Object Mode". Hit *F5* to change to the "Shading" section of the "Buttons Window" and, under "Links and Pipeline" in the text box titled "Link to Object", change the field that reads "MA:Material.003" (*see* **Note 13**) to "MA:VGCC_Reg". Hit *F9* to return to the "Editing" section of the "Buttons Window".
5. *Cancel the clipping box*: Hit *Alt-b*.

6. *Save the current mesh objects*: Hit *F2* to save the current meshes (*see* **Note 11**).

3.1.6. Define a Region for Postsynaptic Receptors

Similar to **Subheading 3.1.5**, we now define the region on the spine head (postsynaptic) membrane that will hold the ligand-gated neurotransmitter receptors (**Fig. 4G**).

1. *Reset view*: Hit *1* (*on the number pad*) to switch back to the *XZ*-view. If you are continuing from the previous section, the presynaptic bouton is still selected. Hit *h* to hide it.
2. *Define the postsynaptic receptor region*: Right click on the spine head mesh to select it and then hit *Tab* to switch to "Edit Mode". Then click "New" under "Link and Materials" to apply a default material to the entire mesh. Select the top row of faces by hitting *b* and then clicking and dragging a rectangle around them. Then hit *Ctrl-Minus* (*on the number pad*) three times to perform three "Select Less" operations, and thus deselect the three outermost rings of vertices. Click "New" under "Link and Materials". Click the gray color selector box and choose a blue shade. Click "Assign" to apply this material to the selected faces.
3. *Name the region*: Hit *Tab* to change into "Object Mode". Hit *F5* to change to the "Shading" section of the "Buttons Window" and, under "Links and Pipeline" in the text box titled "Link to Object", change the field that reads "MA:Material.004" (*see* **Note 13**) to "MA:Receptor_Reg". Hit *F9* to return to the "Editing" section of the "Buttons Window".
4. *Save the current mesh objects*: Hit *F2* to save the current meshes (*see* **Note 11**).

3.1.7. Add Multiple Spines to a Dendritic Shaft

In this section we create a dendritic shaft from a cylinder and then join it to the existing dendritic spine mesh. We then create replicated spines at different positions along the dendrite, and outline ways to change the dimensions of the spines to make a biologically realistic model similar to that shown in **Fig. 1**. Note that the presynaptic bouton mesh remains hidden throughout these operations since it will not be replicated. Thus, for the sake of subsequent illustrative simulations, we will end up with a single synapse onto one spine head, although the model dendrite will include multiple spines similar to those illustrated in **Fig. 1**.

1. *Create the dendritic shaft:* Hit *1* (*on the number pad*) for the *XZ*-view. Hit *Tab* to go into "Object Mode". Hit *Shift-c* to center the cursor at the origin. Hit the *spacebar*, mouse over "Add", then "Mesh", and click on "Cylinder". In the "Add Cylinder" menu that appears, set "Vertices" to "16", "Radius" to "1.00", "Depth" to "1.00", make sure that "Cap Ends" is deselected, and then click "OK". Hit *r*, type *11.25*, and hit *Enter*. This final rotation step around the axis of the cylinder

will align one of the cylinder faces with the base of the spine object.

2. *Assign material properties to the dendritic shaft*: In **step 8** later, the shaft will be joined to the dendritic spines, but we want the shaft to be a separate mesh region rather than an extension of the spine region. Thus, we need to apply new material properties to the shaft. Simply click "New" under "Link and Materials" to apply a new set of default material properties.
3. *Subdivide the faces along the length of the shaft*: For subsequent joining of spines to the shaft, it is preferable to subdivide the cylinder faces along their length. Hit *7* (*on the number pad*) for the *XY*-view. Click "Beauty" under "Mesh Tools". Hit *w* and, in the "Specials" menu, select "Subdivide Multi". In the "Number of Cuts" menu, select "2" and hit "OK". Each of the faces that made up the sides of the cylinder should now have been subdivided into three faces. Hit *1* (*on the number pad*) to change back to the *XZ*-view.
4. *Move the shaft to the end of the dendritic spine*: Hit *Tab* to go into "Object Mode". Hit *g*, then *z*, type *–3.0*, and hit *Enter* to confirm.
5. *Create multiple spines*: Hit *Shift-s* and click "Cursor –> Selection" in the "Snap" pop-up menu. Right click on the "SpineHead" object to select it (from the current view the cylinder may seem to have disappeared, but it is still present). Hit *Tab* to go into "Edit Mode". Hit *a* and verify that the entire object is selected. Under "Mesh Tools" in the "Buttons Window", set "Degr" to "270.00", "Steps" to "3", and "Turns" to "1". Directly above these text boxes, click "Spin Dup". This will create three copies of the original spine head and place them at 90° increments around the cursor, which lies on the axis of the cylinder.
6. *Join the spines to the shaft*: Hit *Tab* to go into "Object Mode". All of the spines will be selected automatically following the preceding step. Hold *Shift* and right click on the shaft (cylinder) object to add it to the selected set of objects. Hit *w* and click "Union" in the "Boolean Tools" pop-up menu. This will perform a Boolean operation and merge the spine objects with the shaft object to create a new object (*see* **Note 14**). However, at this stage the original objects are still present and are still selected. Instead of deleting them, move them to another layer in case any changes are necessary later. This is done by hitting *m* (move), followed by *2* (*not on the number pad*), and clicking "OK". The original objects have now been moved to layer 2.
7. *Name the object created by the union operation*: Right click on the new merged object to select it. To name it, look under "Link and Materials", click in the text box that reads "OB:Tube.001", and change it to "OB:Dendrite".

8. *Replicate, extend, and merge the entire object*: Hit *Ctrl-a* and select "Apply scale and rotation" (*see* **Note 15**). Under the "Modifiers" tab in the "Buttons Window", click "Add Modifier" and select "Array". Change "Count" to "3". Change the "Relative Offset" of "*X*" to "0.000" and "*Y*" to "1.000". Click the "Merge" button. This operation will replicate the object twice (for a total count of 3), move the replicates along the *Y*-axis to the end of the previous piece, and then merge all of the pieces together. To see the result, change back to the *XY*-view [hit *7* (*on the number pad*)] and note how the shaft extends along the *Y*-axis.
9. *Generate a preliminary rotation of the spines*: Hit *1* (*on the number pad*) to change back to the *XZ*-view. Hit the *spacebar*, mouse over "Add", and select "Empty". This will create and select an empty object to which we will add a rotational transformation. First, hit *Alt-r* and click "Clear Rotation" to negate any preexisting rotations applied because the object was created using a certain view (*XZ*-view in this case). Next, hit *r*, type *45.0*, and hit *Enter* to rotate the empty object 45° around the current view axis. Now, right click on the dendritic shaft to select it. In the "Modifiers" tab, click "Object Offset" and then type "Empty" in the text box below. This will add the rotation of the "Empty" object to the dendritic shaft segments created in the preceding step. The amount of rotation is the product of the segment index and the specified 45°. The segment indices are (0, 1, 2), and so the first segment is not rotated, the second is rotated by 45°, and the third is rotated by 90° (and thus is realigned with the first segment).
10. *Finalize the rotation of the spines (array extensions)*: At this point we can view the preliminary rotation of the array extensions, but the rotation has not yet been finalized. Until it is finalized, we cannot make changes to the individual faces and vertices when in "Edit Mode". To finalize the rotation, click "Apply" in the "Array" modifier.
11. *Apply optional changes to the spine dimensions*: At this stage, each of the spine necks can be lengthened or shortened, and the spine head dimensions can be modified as well. For example, to lengthen a particular spine, hit *Tab* to go into "Edit Mode". Hit *b* and drag the select marquee around the vertices of the spine head. Then, along the bottom of the "3D View" pane, change the drop-down "Orientation" button from "Global" to "Normal". Hit *g*, then *z twice* (*see* **Note 16**), and then type in a positive value to lengthen the spine or a negative value to shorten it. Hit *Enter* to apply the change. In a similar fashion, individual spine heads can be scaled by selecting them, hitting *s*, and then entering a

value between 0 and 1 (shrink) or greater than 1 (expand). Finally, similar operations could be used to shrink or expand the diameter of the spine neck, or move the presynaptic bouton together with the postsynaptic spine head.

3.1.8. Smooth the Meshes

Sharp angles between mesh faces can cause inaccuracies and/or instabilities for many computational algorithms (e.g., computing a gradient on the mesh), and hence it is often desirable or necessary to smooth or otherwise "optimize" the mesh. In effect this amounts to low-pass filtering of the mesh to remove sudden (high-frequency) changes in shape (curvature). Smoothing is considerably less important for MCell simulations, because diffusing molecules in MCell move as discrete particles between meshes (volume molecules in solution) or on meshes (surface molecules in membranes), and the mesh *per se* is not used for gradient or other calculations. Nevertheless, here we illustrate smoothing in Blender to achieve a more "biological" appearance of the geometry (compare the objects in **Fig. 4** to the smoothed version in **Fig. 1**).

1. *Smooth the new mesh object that includes the dendrite and spines*: If you are continuing from the previous section, hit *Tab* and make sure you are in "Edit Mode". If necessary, hit *a* to select the entire new object that includes the dendritic shaft and all of the spines (the presynaptic bouton is still hidden). In the "Buttons Window", under "Mesh Tools", click "Smooth" five times. This will iteratively soften sharp edges between faces by moving edge vertices, but will not change the total number of vertices and faces.
2. *Unhide and select the presynaptic bouton mesh*: Hit *Tab* to go into "Object Mode". Hit *Alt-h* to make the "PresynapticBouton" reappear. Right click on the "PresynapticBouton" to select it.
3. *Smooth the presynaptic bouton mesh*: Hit *Tab* again to go into "Edit Mode". If necessary, hit *a* to select all of the presynaptic bouton mesh and, under "Mesh Tools", click "Smooth" five times. Hit *Tab* to go back into "Object Mode".
4. *Save the current mesh objects*: Hit *F2* to save the current meshes (*see* **Note 11**).

3.1.9. Export MDL Files for MCell Simulations

To complete this section we export the current mesh objects in MDL format for use in MCell simulations of presynaptic calcium influx and binding (**Subheading 3.3**). Export of MDL files utilizes the MDL plug-in for Blender that was installed in **Subheading 2.3**.

1. Click "File –> Export MCell (.mdl)". In the top text box that appears, navigate to the desired directory (folder) in which to

store the new files (you can create a new directory first if necessary). In the text box for the filename enter "Synapse.mdl".

2. Click "Export MDL" and click "OK". This will create five new MDL files in your specified directory. "Synapse.mdl" is the main file and will be read when the MCell simulation is started. It contains MDL statements that specify initial values for some important simulation parameters (*see* **Note 17**). It also includes statements for default visualization output for use with DReAMM (**Subheading 3.4**), and lists the remaining four files that also must be read (included) when the simulation is initialized. These remaining files are "Synapse_PresynapticBouton.mdl", "Synapse_Dendrite.mdl", "Synapse_Vesicle1.mdl", and "Synapse_Vesicle2.mdl", and they all contain mesh geometry and region information (*see* **Note 18**). Note that the names of the files correspond to the names of the mesh objects defined in the preceding sections.

3.2. Adding an Expanding Synaptic Vesicle Fusion Pore to the Model

In **Subheading 3.5** we will illustrate MCell simulations of neurotransmitter release and binding to postsynaptic receptors. The neurotransmitter molecules will diffuse through expanding fusion pores that connect the synaptic vesicles to the presynaptic membrane, so here we show how Blender can be used to create and scale the expanding pore structures. In brief, we create the pores with their initial and final dimensions (radius) and then morph between those limits to generate a set of intermediate pore structures. In a more realistic diffusion simulation project there might be several hundred intermediate structures, but in this simple example we will use only ten configurations including the initial and final. Each configuration will be written to a set of MDL files separate from those exported in the preceding section, and then in **Subheading 3.5** MCell will be used to read the succession of new mesh files using a feature called checkpointing (*see* **Note 19**).

1. *Join the meshes of both vesicles*: Hit *1* (*on the number pad*) for the *XZ*-view. Pan and zoom in on the presynaptic bouton so that the vesicles are clearly visible. Hit *z* to make the faces transparent. Now the vesicles should be visible inside the presynaptic bouton. Right click on the left vesicle to select it, and then, while holding *Shift*, right click on the other vesicle to select it as well. Next hit *Ctrl-j* and click "Join Selected Meshes" when prompted by the "OK?" dialog box. Hit *Tab* to go into "Edit Mode".
2. *Remove the bottom faces of each vesicle*: By removing the bottommost faces of each vesicle, we will create holes that can be extruded to create cylindrical pores, similar to the way that the axon and spine neck extensions were extruded in **Subheading 3.1.3**. Hit *Ctrl-Tab* and click "Vertices" in the "Select Mode" menu. Hit *a* to deselect everything. Right click on the

bottommost vertex of the left vesicle and, while holding *Shift*, right click the bottommost vertex of the right vesicle (use zoom and pan if necessary). Hit *Ctrl-Plus* (*on the number pad*) once to select one ring of vertices. Hit *x* and click "Faces" in the "Erase" menu.

3. *Extrude the pores*: Hit *b* and then click and drag the select marquee around the vertices remaining at the bottom of the left vesicle. Then hit *b* again and repeat for the right vesicle. Hit *e*, click "Only Edges", then hit *z*, type *–0.03*, and hit *Enter*. You should now see the extruded pores passing through the presynaptic bouton (**Fig. 4H**, left). The diameter of these temporary pores is approximately 0.015 units (*see* **Note 20**).
4. *Merge the fusion pores with the presynaptic membrane*: Hit *Tab* to go into "Object Mode". The vesicles should already be selected, so, while holding *Shift*, right click on the surrounding presynaptic bouton mesh. Now hit *w* and click "Difference" from the "Boolean Tools" menu. The vesicles, pores, and presynaptic bouton membrane should now form a continuous mesh (*see* **Note 14** and **Fig. 4H**, right).
5. *Save the original meshes*: The original objects are still present after performing the Boolean difference operation in the preceding step. Rather than delete them, move them to another layer for later use if any changes are required. Hit *m*, followed by *2* (*not the number pad*), and then click "OK".
6. *Name the object created by the difference operation*: Right click on the new object to select it. To name it, look under "Link and Materials", click in the text box that reads "OB:PresynapticBouton.001", and change it to "OB:PresynapticBouton".
7. *Scale the pores to their desired initial diameter*: The desired initial diameter for the pores is about 13.3% of the current diameter (compare **Fig. 4I**, left, with **Fig. 4H**). In the subsequent MCell simulation, this initial diameter will correspond to about 2 nm. Hit *Tab* to enter "Edit Mode". Hit *Ctrl-Tab* and click "Faces", so that we can now select faces rather than vertices as in preceding sections. Hit *b* and then click and drag the rectangular select marquee over the middle of a pore, but do not include the upper and lower vertices of the pore. This will select only the faces of the pore. Hit *7* (*on the number pad*) to change to the *XY*-view (overhead). Hit *s* and then *Shift-z* to simultaneously scale along the *X*- and *Y*-axes. Type *0.133* and hit *Enter*. Hit *1* (*on the number pad*) to change back to the *XZ*-view. Hit *a* to deselect. Repeat the selecting and scaling steps for the second pore.
8. *Take a snapshot of the initial pore configurations*: This snapshot subsequently will be used in **step 9** later when we morph the pores between their initial and final configurations. Hit

Tab to enter "Object Mode". Select the presynaptic mesh by right clicking on it. On the "Shapes" panel in the "Buttons Window", click "Add Shape Key". We have now defined a "Basis Key" that corresponds to the initial pore configuration. All subsequently defined "Shape Keys" will be relative to the "Basis Key".

9. *Take another snapshot to be modified for the final pore configuration*: Click "Add Shape Key" again. Now we will rescale the pore dimensions to their final diameter (**Fig. 4I**, right; about 10 nm in the subsequent MCell simulation). Hit *Tab* to enter "Edit Mode". Hit *b* and then click and drag the rectangular select marquee over the middle of a pore as in earlier **step 6**. Hit *7* (*on the number pad*) to change to the *XY*-view (overhead). Hit *s*, then *Shift-z*, type *5.0*, and hit *Enter*. Hit *1* (*on the number pad*) to change back to the *XZ*-view. Repeat for the second pore. At this point the first snapshot contains the initial pore configuration, and the second snapshot contains the final pore configuration.
10. *Interpolate between the snapshots*: Hit *Tab* to enter "Object Mode". Click on the icon in the upper left-hand corner of the "Buttons Window" (**Fig. 2D**) and select the "Action Editor" from the drop-down list. Click on the arrow beside the word "Sliders" near the bottom of the window, and sliders for the list of available "Shape Keys" will appear. In this simple case, only "Key 1" is present. Now we must map the final pore configuration (Key 1) to the endpoint of a timeline, and the "Basis Key" to the beginning of the timeline. Identify the vertical green line to the right of the "Shape Key" slider. Click on the line, drag it to the right, and then release it at the point marked 10 (requesting ten snapshots). Now move the slider next to "Key 1" from "0.00" to "1.00", and a diamond-shaped marker will appear at position 10 on the timeline. Next drag the green line back to 1 and then drag the slider from "1.00" back to "0.00". A diamond marker will now appear at position 1 on the timeline. Hit *Shift-Alt-a* to see an animation of the interpolated pore configurations. Hit *Esc* to stop playback.
11. *Save the current mesh objects*: Hit *F2* to save the current meshes (*see* **Note 11**).
12. *Save the interpolated mesh snapshots as MDL files*: We will now export MDL files for use with MCell in **Subheading 3.5**. As in earlier **Subheading 3.1.9**, click "File –> Export –> MCell (.mdl)". Navigate to the desired directory (folder) in which to store the new files (use a different directory from that used in **Subheading 3.1.9**). In the text box for the filename enter "VesicleFusion.mdl". Click "Export MDL", then "Enable Anim.", and "Iterate Script". Also change "Stop" to "10". Once these changes have been made, click "OK" (*see* **Note 21**).

13. *Save the current mesh objects and quit Blender*: Hit *F2* to save the current meshes (*see* **Note 11**) and quit Blender by clicking on the X in the upper right-hand corner of the window.

3.3. MCell Simulations of Presynaptic Calcium Influx and Binding

In **Subheading 3.1** we used Blender to create a set of pre- and postsynaptic meshes and then exported the meshes as MDL files for use with MCell. As outlined in **Subheading 3.1.9**, five files were created: the main file "Synapse.mdl" and the four geometry files "Synapse_PresynapticBouton.mdl", "Synapse_Dendrite.mdl", "Synapse_Vesicle1.mdl", and "Synapse_Vesicle2.mdl". In Blender the meshes were composed of quadrangular faces (**Fig. 4**), and the absolute spatial dimensions were arbitrary. In MCell, however, the spatial units will be interpreted as microns. In addition, MCell's collision detection algorithms require triangular faces, so each quadrilateral face in Blender was automatically split into two triangles when the meshes were exported. We will now supplement the exported MDL files in order to populate the meshes with molecules, define reactions between molecules, and provide commands that control how the MCell simulations will be run. For convenience we will use separate MDL files for many of these distinct operations. The final simulations then will be controlled from the main file that reads or "includes" all the subordinate files in the proper order when the simulation is initialized.

3.3.1. Define Molecules

We first create a new MDL file to describe the molecules included in the model. It will specify whether they exist in solution (a "volume" molecule) or on a surface (a "surface" molecule), and their diffusion coefficients.

1. At a command line, change into the directory where you exported the MDL files in **Subheading 3.1.9**.
2. Create a new file called "Molecules.mdl" (*see* **Note 22**).
3. Define the molecules: Enter the following block of text (*see* **Notes 23** and **24**):

```
DEFINE_MOLECULES {
Ca {DIFFUSION_CONSTANT_3D = 1E-6}
VGCC_C {DIFFUSION_CONSTANT_2D = 0}
VGCC_O {DIFFUSION_CONSTANT_2D = 0}
CaBS {DIFFUSION_CONSTANT_2D = 0}
CaBS_Ca {DIFFUSION_CONSTANT_2D = 0}
}
```

This simple model includes only diffusing calcium ions, VGCCs, and calcium binding sites that might, for example, be based on synaptotagmin molecules located on the synaptic vesicles. In MDL statements like those above, the names of the

molecules are specified by the user and thus are usually chosen to be easily recognizable. The only (obvious) restriction is that a user-specified name may not be an exact match of an MDL keyword. In this example, the calcium ions are simply named "Ca", and since they are to be diffusing volume molecules they are given a nonzero 3D diffusion coefficient (cm^2/sec). We require both a closed and open state for the VGCCs, named "VGCC_C" and "VGCC_O", respectively. The channels will be stationary surface molecules and so are given a 2D diffusion coefficient with a value of 0. Finally, we require unbound and bound states for the calcium binding sites, named "CaBS" and "CaBS_Ca", respectively. Similar to the channels, the binding sites will be considered part of static surface molecules, and hence are given a 2D diffusion coefficient with a value of 0.

4. Save the file and quit.

3.3.2. Add Molecules to Mesh Regions

The only molecules that are to be present when the simulation begins are the closed VGCCs ("VGCC_C") and the unbound calcium binding sites ("CaBS"). The remaining molecules or states will be generated by reactions during the simulation. We add ten "VGCC_C" molecules to the presynaptic mesh region "VGCC_Reg" that was defined in **Subheading 3.1.5** (**Fig. 4F**). Similarly, we add ten "CaBS" molecules to the "CaBS_Reg" region defined on the synaptic vesicles in **Subheading 3.1.4**. The actual locations of the molecules within these regions will be randomized by MCell when the simulation is initialized (*see* **Note 25**).

1. Create a new file called "RegionModifications.mdl" (*see* **Note 22**).
2. Define the numbers and locations of molecules present at simulation start-up (*see* **Note 26**):

```
MODIFY_SURFACE_REGIONS {
PresynapticBouton[VGCC_Reg] {
MOLECULE_NUMBER { VGCC_C, = 10 }
}
Vesicle1[CaBS_Reg] {
MOLECULE_NUMBER { CaBS' = 10 }
}
Vesicle2[CaBS_Reg] {
MOLECULE_NUMBER { CaBS' = 10 }
}
}
```

These MDL statements modify (add molecules to) the preexisting mesh regions defined automatically when the "Synapse_Presyn-

apticBouton.mdl", "Synapse_Vesicle1.mdl", and "Synapse_Vesicle2.mdl" files were exported from Blender. The comma or apostrophe that follows the molecule's name specifies how the molecule is oriented when it is added to the surface (a surface molecule may have a reactive domain, e.g., a binding site, on the front and/or back of the surface; *see* **Note 17**).

3. Save the file and quit.

3.3.3. Add Reactions

1. Create a new file called "Reactions.mdl" (*see* **Note 22**).
2. Define the reactions: Enter the following block of text:

```
DEFINE_REACTIONS {
VGCC_C' -> VGCC_O' [5E5]
VGCC_O' -> VGCC_C' [500]
VGCC_O' -> VGCC_O' + Ca' [1E3]
Ca' + CaBS' -> CaBS_Ca' [1E7]
CaBS_Ca' -> Ca' + CaBS' [500]
}
```

These MDL statements specify the stoichiometry, rates, and directionality for the reactions in the simulation. In the first line, closed VGCCs are able to undergo a conformational change to the open state with a first-order mass action rate constant of 5E5 s^{-1}. The reverse transition occurs in the second line, albeit at a much slower rate. A channel in the open state is also able to generate diffusing calcium ions in the presynaptic bouton (third line). Hence, in any given simulation time step, an open channel may close, generate one or more calcium ions, or simply remain open, all based on relative probabilities. This method for generating diffusing calcium ions from open channels is far more efficient than explicitly simulating separate pools of extracellular and intracellular calcium ions that pass through the open channel. In the fourth line, calcium ions bind to the calcium binding sites with a bimolecular mass action rate constant of 1E7$M^{-1}s^{-1}$. The last line specifies calcium unbinding with a first-order rate constant of 500 s^{-1}.

In all of these reactions, the apostrophes again specify the directionality of the reactions with respect to the orientation of the surface molecules. This is why calcium ions produced by an open channel enter the presynaptic bouton rather than the synaptic cleft space (*see* earlier **Subheading 3.3.2**, **step 2** and **Note 17**).

3. Save the file and quit.

3.3.4. Specify Reaction Data Output

MCell is able to count and save many different types of events during a simulation. In this simple example, we will just count the number of molecules present during each time step throughout the entire simulation space (world). The results for each molecule

will be written to a separate ASCII file containing two columns. The first gives the simulation time in seconds, and the second gives the counted quantity.

1. Create a new file called "ReactionData.mdl" (*see* **Note 22**).
2. *Specify the desired reaction data output*: Enter the following block of text:

```
REACTION_DATA_OUTPUT {
{COUNT[VGCC_C, WORLD]} => "./reaction_data/VGCC_C.dat"
{COUNT[VGCC_O, WORLD]} => "./reaction_data/VGCC_O.dat"
{COUNT[CaBS, WORLD]} => "./reaction_data/CaBS.dat"
{COUNT[CaBS_Ca, WORLD]} => "./reaction_data/CaBS_Ca.dat"
{COUNT[Ca, WORLD]} => "./reaction_data/Ca.dat"
}
```

In this case, each file (.dat suffix) will be created automatically in a subdirectory called "reaction_data".

3. Save the file and quit.

3.3.5. Add Spatial Partitions to Speed Computation

During a simulation, diffusing molecules follow random walk steps that must be traced to detect possible collisions with surfaces and other molecules. This becomes very time-consuming unless each molecule looks only in its local environment first, and continues into adjoining space only if necessary. To define the local environments, the simulation world is partitioned into subvolumes. In effect, the partitions are transparent planes along the *X*-, *Y*-, and *Z*-axes, and the subvolumes are the cuboidal spaces created between the partitions.

1. Create a file called "Partitions.mdl" (*see* **Note 22**).
2. *Specify the locations of partitions along each axis*: Enter the following block of text:

```
PARTITION_X = [[-1.25 TO 1.25 STEP 0.1]]
PARTITION_Y = [[-1.25 TO 1.25 STEP 0.1]]
PARTITION_Z = [[0 TO 1 STEP 0.1]]
```

3. Save the file and quit.

3.3.6. Add the Include Files and Set the Number of Iterations

We now add all the pieces together by referencing ("including") the newly created MDL files in the main simulation file "Synapse.mdl". Thus, when the simulation is started using the main file, all of the subordinate MDL files will be read in the proper order.

1. Open the main file "Synapse.mdl" in a text editor (*see* **Note 22**).
2. *Reference the subordinate MDL files using INCLUDE statements*: Before the first preexisting INCLUDE statement add the following text:

```
INCLUDE_FILE = "Partitions.mdl"
INCLUDE_FILE = "Molecules.mdl"
INCLUDE_FILE = "Reactions.mdl"
```

Now, after the last preexisting INCLUDE statement add:

```
INCLUDE_FILE = "RegionModifications.mdl"
INCLUDE_FILE = "ReactionData.mdl"
```

3. *Change the iteration number*: By default, the simulation is set to run for only one iteration with a default time step of one microsecond. This will generate visualization output for the start-up conditions, and thus would allow verification of initial mesh and molecule placement using DReAMM (*see* **Subheading 3.4** later). This is an extremely useful step for large models that take a long time to simulate. In this case, however, we will increase the number of iterations so that we can see the appearance of diffusing calcium ions and occupation of calcium binding sites at later times. Change the first line of the file from:

   ```
   iterations = 1
   ```
   to
   ```
   iterations = 5000
   ```
4. Save the file and quit.

3.3.7. Run the Simulation

Assuming that you have installed MCell with the name "mcell3", run the simulation simply by entering:

mcell3 Synapse.mdl

at the command line in the directory where you created the MDL files. MCell will start and display initialization messages, display updates as iterations complete, and then display a variety of runtime summary statistics when the simulation is finished.

3.4. Visualize MCell Results with DReAMM (Part 1)

It is crucial to check any MCell model visually using DReAMM to verify that all components (location of molecules, reactions, geometry, etc.) have been set up properly (*see* **Note 27**). Here, we outline the essential steps for the model created in the preceding section.

3.4.1. Import the MCell Visualization Data and Select Mesh and Molecule Objects

1. *Start DReAMM*: Enter "dreamm" at a command line. The "DReAMM Image Window", "Quick Controls", and "Sequence Control" should all appear. In principle you can start DReAMM from within any directory and then navigate to the visualization files output by MCell. To simplify this example, however, start DReAMM from the same directory in which you ran the MCell simulation in **Subheading 3.3**.
2. *Import visualization data*: Click the "Import & Select" button near the top of the "Quick Controls" window (labeled

1 in **Fig. 3a**). Two menus (windows) should open. In the "Import & Select Objects" menu, click the ellipsis ("…") in the "Viz Data File" text box. Navigate to the "Synapse_viz_data" directory that was created by MCell and select the file named "Synapse.dx". Click "OK". DReAMM will automatically read the visualization files referenced by "Synapse.dx", display the names of available objects, and import the data for the first time step. By default, however, DReAMM will not display any data until it is selected.

3. *Select all meshes to be displayed*: The lower left-hand side of the "Imported Objects" menu will now display the names of the meshes we created previously in Blender and used in MCell: "World.Dendrite", "World.PresynapticBouton", etc. These objects thus are available to be rendered (displayed) and for other operations. In the lower center section locate the field named "Choose Operation", click on "Add All", and then click the "Apply Operation" button to select all of the meshes. All of the object names should now appear in the "Current Objects" list to the right, and all of the mesh objects now will be displayed in the "DReAMM Image Window" using DReAMM's default mesh colors and "Software Rendering Mode" (*see* **Note 28**).
4. *Select all volume molecules*: Click "Volume Molecules" in the central "List" field of the "Import and Select" menu, and the "Imported Objects" list will change to show the volume molecules in the model. In this case only "Ca" is present. Hit the "Apply Operation" button, and "Ca" will appear in the list of "Current Objects". However, no calcium ions are yet visible in the "Image Window" because no calcium ions are present at the beginning of the simulation, and we are currently viewing the first time step.
5. *Select all surface molecules*: Click "Surface Molecules" under "List" and the "Imported Objects" list will change again, this time showing all surface molecules in the model (calcium channels and binding sites). Click "Apply Operation", and all the surface molecule names will appear in the list of "Current Objects". At this point the surface molecules present at the beginning of the simulation are being rendered using DReAMM's default molecule rendering properties (white pixels). This may be difficult to see if there are few molecules or they are sparsely distributed but is the least expensive display option. In the following section we will customize the display properties so that the molecules can be seen easily.
6. *Center the view*: Select the "DReAMM Image Window" and hit *Ctrl-f* to center the view of the displayed objects. To see the synapse from a side view, hit *Ctrl-v* to bring up the "View Control" menu and then select "Bottom" under "Set View" (*see* **Note 29**).

7. *Close menus*: Click the "Close" button to the right of the "Import & Select" field on the "Quick Controls" menu (labeled *5* in **Fig. 3A**).

3.4.2. Visualize the Calcium Binding Site Regions

1. *Open the rendering properties menus*: Click "Set Rendering Prop". under "Quick Controls" (labeled *2* in **Fig. 3A**). Four separate menus should appear.
2. *Turn on the preview window*: In the "Rendering Properties" menu, click on the "Enable Preview" button in the center and the "Rendering Preview" window will appear. It displays the current rendering properties that can be applied to selected objects, including color (separate front and back colors for mesh objects), lighting, and shading (*see* **Note 30**).
3. *Make the presynaptic bouton semitransparent*: By default, the "Rendering Properties" menu will display the names of the imported mesh objects, and the first will be highlighted. Click on "World.PresynapticBouton" to select it, and then, if necessary, click on any others that remain highlighted to deselect them. In the lower left-hand corner, change "Opacity" from "1.0" to "0.3" (*see* **Note 31**) and then click on the "Once" button next to "Apply Operation" in the center of the menu. In the "DReAMM Image Window", the mesh for the "PresynapticBouton" will now be semitransparent, revealing the vesicles within it. Furthermore, the change in opacity is also reflected in the "Rendering Preview" window.
4. *Visualize the mesh regions using a colormap*: In order to distinguish different mesh regions in DReAMM, we use a colormap to render and display the object. When a region is defined in Blender, each mesh face within the region is automatically assigned a unique numerical metadata tag (value). The tag values begin with 0 for the first region, and are incremented thereafter. Similarly, in MCell's MDL, triangles that belong to a particular region can be assigned a numerical VIZ_VALUE. Meshes exported from Blender to MCell automatically inherit region VIZ_VALUEs from the metadata tags assigned by Blender. DReAMM subsequently uses the VIZ_VALUEs and colormaps to visualize regions. The default colormap visible in the "Colormap Editor" menu uses a stair-step pattern ranging from purple to red for tag values within the indicated numerical limits. We will now change the upper limit so that the default colormap will work for our mesh regions. To do so, click in the upper box that displays a numerical value, type "1.4", and then hit *Enter* (*see* **Note 32**).
5. *Apply the colormap to the synaptic vesicles*: In the "Rendering Properties" menu, select "World.Vesicle1", then "World.Vesicle2", and finally deselect "World.PresynapticBouton". Change "Use Color and Opacity Map" from "No" to "Yes"

and "Color Dependence" from "Vertices" to "Elements" (*see* **Note 33**). Then click "Apply Operation Once". The bottom of the vesicles, i.e., the region "CaBS_Reg", should now be yellow because it was assigned a VIZ_VALUE of 1. The remainder of each vesicle is purple because it has a VIZ_VALUE of 0. You may need to zoom in to see this clearly (*see* **Note 29**).

3.4.3. Highlight Different Molecules and Visualize the MCell Time Series

By default, the visualization data for each simulation time step (frame) has been saved, and DReAMM will automatically read and display the selected data for each frame in sequence. During playback calcium ions will appear and diffuse, and calcium binding sites will become occupied. For maximum speed they will all be rendered as white pixels under default conditions, so in the later step we will change the rendering properties for each molecule so that they may be distinguished clearly. We will also change the color of the postsynaptic mesh (dendrite and spines).

1. *Play the time series data*: Simply press the Play button on the "Sequence Control" menu (right arrowhead, **Fig. 3B**) to begin playback of the MCell simulation time series. No visible changes will be evident until several hundred frames have been displayed, so skip ahead if desired by clicking on the "Sequence Control" button that displays the frame number. Then use the pop-up "Frame Control" menu to select a subset of the available frames, and/or change the interval between the displayed frames.
2. *Reset the opacity*: In the "Rendering Properties" menu, change "Opacity" back to "1.0". This change will be visible in the "Rendering Preview" window but will not affect any objects until we assign properties to them.
3. *Choose a color from the Color Library*: In the "Color Library" or "Rendering Properties" menu, toggle the "Source" selector from "Rendering Properties" to "Color Library". This will change the source of the colors displayed in the "Rendering Preview" window from the color fields of the "Rendering Properties" menu to the color selected in the "Color Library" menu. In the "Display List Filter" text box of the "Color Library" menu, change the search string from "*" to "*yellow*" and hit *Enter* (*see* **Note 34**). Select "lightyellow3" from the list and hit the "Load" button. The RGB (Red, Green, Blue) values for "lightyellow3" are now listed in the "Front Color" and "Back Color" (half intensity) fields of the "Rendering Properties" menu, and the corresponding hue is also visible in the "Colormap Editor".
4. *Assign the new color to the dendrite*: In the "Rendering Properties" menu select "World.Dendrite", deselect "World.Vesicle1" and "World.Vesicle2", and then click "Apply Operation Once". The dendrite mesh, which includes the spines and spine heads, will now be rendered with "lightyellow3".

5. *Assign a yellow spherical glyph to the calcium ions*: In the "Rendering Properties" menu, click on "Molecules" in the upper middle section by the word "List". The list of object names will switch to all of the molecules (volume and surface) in the model. In the lower right-hand corner change "Glyph" from "pixel" to "sphere (simple)" and both "Height" and "Radius" to "0.0025". At the lower left manually change "(Front) Color" to yellow by entering RGB values of 1, 1, and 0, respectively ("Back Color" is ignored for glyphs). Make sure that only "Ca" is highlighted in the list of object names. Assign the yellow glyph properties to calcium ions by clicking "Apply Operation (Once)".
6. *Assign a black arrow glyph to the unbound calcium binding sites*: Surface molecules have an XYZ location that lies on a surface mesh element, and they also have an orientation with respect to the plane of the mesh element. Thus, to visualize surface molecules we will use a directional (asymmetric) glyph (*see* **Note 35**). First, in the "Rendering Properties" menu, select "CaBS" and deselect any other highlighted molecules. Change the glyph to "arrow (simple)". Change the "Height" to "0.01" and the "Radius" to "0.0025". To make the arrow glyphs black, change all of the "(Front) Color" RGB values to 0. Click "Apply Operation Once". Outward pointing arrows should now be visible at the position of the CaBS molecules within the CaBS_Reg of each synaptic vesicle (**Fig. 5A**). The precise number will depend on the time step that you are viewing. At the beginning of the simulation (frame 1) there are ten unbound calcium binding sites. Later, the number will change as calcium ions bind and unbind.
7. *Assign a cyan arrow glyph to the bound calcium binding sites*: In the "Rendering Properties" menu, select "CaBS_Ca" and deselect "CaBS". Change the front RGB values to 0, 1, and 1, respectively. Click "Apply Operation Once". The bound calcium binding sites will now be visible as cyan, outward pointing arrows. As outlined in the preceding step, the number will depend on the time step that you are viewing (**Fig. 5B**).
8. *Assign a red arrow glyph to the closed VGCCs*: In the "Rendering Properties" menu, select "VGCC_C" and deselect "CaBS_Ca". Change the front RGB values to 1, 0, 0, respectively. Click "Apply Operation Once". The VGCCs (in the presynaptic membrane beneath the synaptic vesicles) that currently are closed will now appear as red arrows pointing toward the interior of the presynaptic space (**Fig. 5A**).
9. *Assign a green arrow glyph to the open VGCCs*: Similar to the preceding step, select "VGCC_O" and deselect "VGCC_C". Change the front RGB values to 0, 1, 0, respectively, and click "Apply Operation Once". The VGCCs that currently are open will now appear as green arrows (**Fig. 5B**).

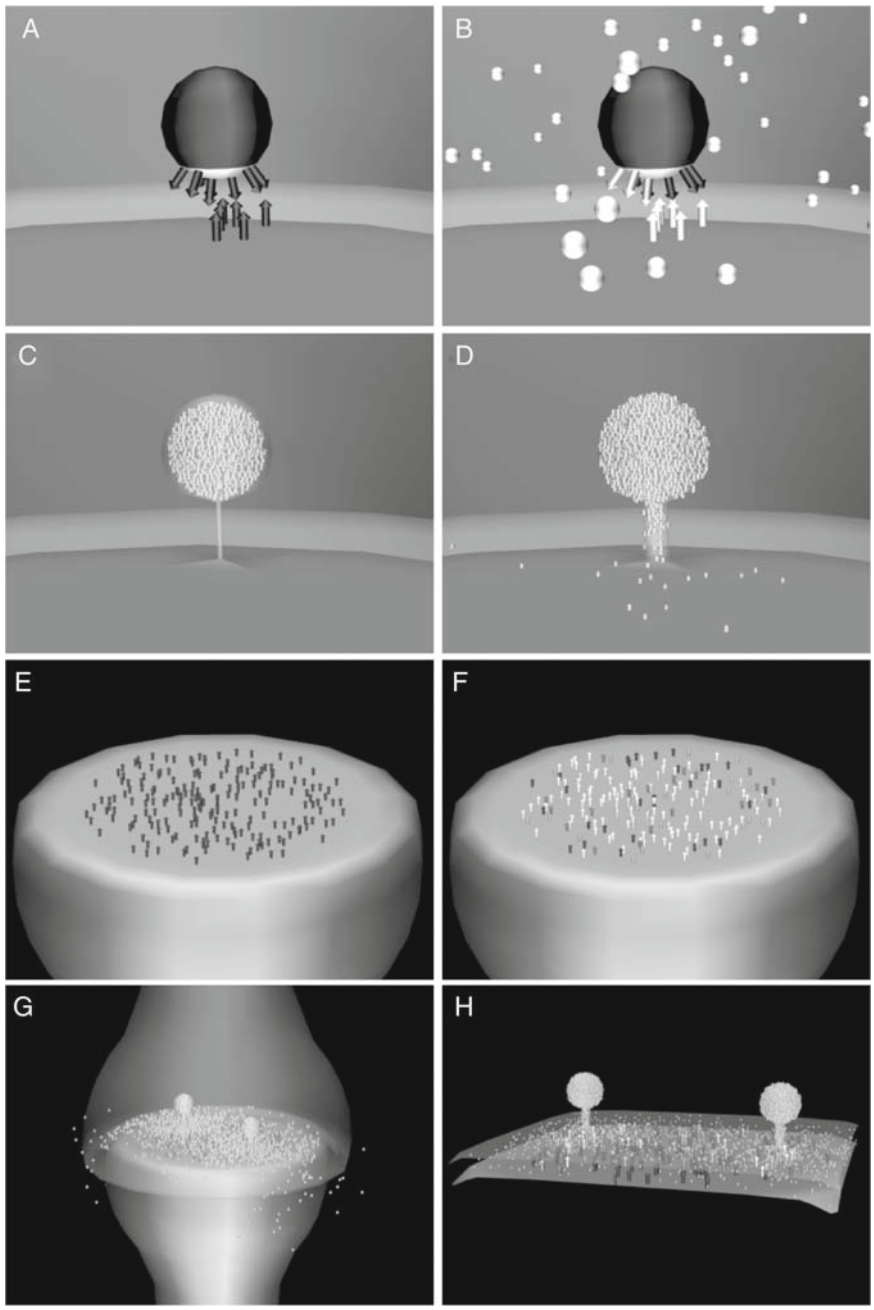

Fig. 5. Screen captures from simulations as visualized with DReAMM. (Note that addition of colors to objects is described in the text, whereas this image has been converted to grayscale and is described here accordingly). (**A**) At the beginning of the first simulation (**Subheading 3.3.7**), the bottom of the synaptic vesicle is populated with unbound calcium binding sites (*black downward pointing arrows*). Closed voltage-gated calcium channels (*black upward pointing arrows*) are located directly underneath the vesicle on the presynaptic membrane. (**B**) Later, calcium channels open (*white upward pointing arrows*) and release calcium ions (*white spheres*) into the presynaptic bouton. Some calcium ions then bind to available sites on the vesicle (*white downward pointing arrows*). (**C**) At the beginning of the second simulation (**Subheading 3.5.7**), neurotransmitter molecules (*white spheres*) fill the synaptic vesicles, and then diffuse out as the simulation proceeds and the fusion pore expands (**D**). For clarity, the postsynaptic receptors are not shown in **C** and **D** even though they are present in the simulation. (**E**) All of the postsynaptic receptors are in the unbound state (*black arrows*) at the start of the second simulation (presynaptic bouton not shown). (**F**) Later, a mixture of single-bound, double-bound, closed, and open receptors is present as indicated by different colors.

10. *Play the time series data*: Press the Stop button (square) on the "Sequence Control" menu (**Fig. 3B**) and then press the Play button. This will restart the time series from the first frame (of the selected interval, if you are using the "Frame Control" settings). Use the Pause (parallel lines), Single-step (double lines and arrowheads), Reverse (left arrowhead), Loop, and Palindrome (loop forward and backward) buttons to modify playback.

3.4.4. Save DReAMM Settings for the Simulation Objects

All DReAMM settings such as choice of rendered objects, rendering properties (including colormaps), etc., can be saved to a file for subsequent reuse.

1. *Save settings*: In the "Read/Write Settings" menu, click on the ellipsis ("…") button next to the "Write File" text box. In the pop-up menu, navigate to the directory where you would like to save the file, enter the name "SynapseCustom.dx", and then click "OK". Click the "Write Once" button in the "Read/Write Settings" menu to save the file (*see* **Note 36**).
2. *Quit DReAMM*: Quit DReAMM by clicking the X in the upper right-hand corner of the "DReAMM Image Window". When prompted, "Do you want to save the project file", click "No".

3.5. MCell Simulations of Fusion Pore Expansion and Neurotransmitter Release

We now adapt the MCell model of **Subheading 3.3** to include the expanding fusion pores (**Fig. 4I**) created with Blender in **Subheading 3.2**. We will add neurotransmitter molecules that originate within the vesicles and diffuse out through the expanding pores. We will also add postsynaptic receptors in the form of ligand-gated ion channels. As mentioned previously (**Subheading 3.2** and **Note 19**), MCell simulation of the expanding pores will use a feature called checkpointing. In brief, we will run a series of MCell simulations, saving the locations and states of all molecules after each run in the series. The first run will use the initial pore configuration and will proceed for a certain number of iterations, allowing neurotransmitter molecules to begin diffusing. The second run will use the next pore configuration but will use the molecule locations and states from the previous run as initial conditions. This pattern then will continue for all ten of the expanding pore configurations. In principle many other parameters can also change between checkpoint runs, and there are a variety of ways to automate setup of the files. Here, we use a simple example for the sake of illustration.

Fig. 5. (Continued) (**G**) The complete synapse is shown at a late time point, with the presynaptic bouton semitransparent and the spine head opaque. Neurotransmitter molecules can be seen diffusing within the synaptic cleft and escaping into the surrounding volume. (**H**) Image clipping is used to provide a better view of the vesicles, synaptic cleft, diffusing neurotransmitter molecules, and postsynaptic receptors.

3.5.1. Define Molecules

As in **Subheading 3.3**, we first create a new MDL file to describe the molecules included in the model. Before starting, make sure that you are in the directory created in **Subheading 3.2** when the MDL files for the expanding pore were exported from Blender.

1. Using a text editor, create a new file called "Molecules.mdl" (*see* **Note 22**).
2. *Define the molecules*: Enter the following block of text (*see* **Notes 17** and **24**):

```
DEFINE_MOLECULES {
nt {DIFFUSION_CONSTANT_3D = 1E-6}
nt_R_0B {DIFFUSION_CONSTANT_2D = 0}
nt_R_1Ba {DIFFUSION_CONSTANT_2D = 0}
nt_R_1Bb {DIFFUSION_CONSTANT_2D = 0}
nt_R_2B_C {DIFFUSION_CONSTANT_2D = 0}
nt_R_2B_O {DIFFUSION_CONSTANT_2D = 0}
}
```

In these MDL statements, we define a diffusing volume molecule ("nt", neurotransmitter) and five different states of a stationary neurotransmitter receptor (stationary surface molecules). The receptor represents a ligand-gated ion channel with two independent binding sites. "nt_R_0B" is the receptor in its unbound state; "nt_R_1Ba" and "nt_R_1Bb" are the two single-bound states; "nt_R_2B_C" is the double-bound, closed channel state, and "nt_R_2B_O" is the double-bound, open channel state.

3. Save the file and quit.

3.5.2. Add Molecules to Mesh Regions

We next add unbound receptors to the postsynaptic receptor region defined in **Subheading 3.1.6** (**Fig. 4G**). The actual locations of the molecules within the region will be randomized by MCell when the first simulation of the checkpoint sequence is initialized. Recall that this region extends to all of the spine heads and hence we add a total of 2,400 receptors distributed randomly across 12 spine heads.

1. Create a new file called "RegionModifications.mdl" (*see* **Note 22**).
2. Define the numbers and locations of molecules present at simulation start-up:

```
MODIFY_SURFACE_REGIONS {
Dendrite[Receptor_Reg] {
MOLECULE_NUMBER { nt_R_0B' = 2400}
}
}
```

3. Save the file and quit.

3.5.3. Add Reactions

1. Create a new file called "Reactions.mdl" (*see* **Note 22**).
2. *Define the reactions*: Enter the following block of text:

```
DEFINE_REACTIONS {
nt' + nt_R_0B' -> nt_R_1Ba' [1E7]
nt_R_1Ba' -> nt' + nt_R_0B' [1E4]
nt' + nt_R_0B' -> nt_R_1Bb' [1E7]
nt_R_1Bb' -> nt' + nt_R_0B' [1E4]
nt' + nt_R_1Ba' -> nt_R_2B_C' [1E7]
nt_R_2B_C' -> nt' + nt_R_1Ba' [1E4]
nt' + nt_R_1Bb' -> nt_R_2B_C' [1E7]
nt_R_2B_C' -> nt' + nt_R_1Bb' [1E4]
nt_R_2B_C' -> nt_R_2B_O' [1E4]
nt_R_2B_O' -> nt_R_2B_C' [1.5E3]
}
```

In the first two lines, a neurotransmitter molecule binds reversibly to the first binding site on the unbound receptor. In the next two lines, a neurotransmitter molecule binds to the second binding site on the unbound receptor. Next, a neurotransmitter molecule binds reversibly to the second site when the first site is already occupied, generating the double-bound closed channel state. Similarly, the double-bound closed channel state can also be generated when a neurotransmitter molecule binds to the first site when the second is already occupied. Finally (last two lines), the double-bound receptor can change conformations reversibly between the closed and open channel states. As outlined in **Subheading 3.3**, the apostrophes specify the directionality of the reactions with respect to the orientation of the surface molecules. In this case the neurotransmitter molecules are able to bind to receptor molecules from within the synaptic cleft space.

3. Save the file and quit.

3.5.4. Specify Reaction Data Output

1. Create a new file called "ReactionData.mdl" (*see* **Note 22**).
2. *Specify the desired reaction data output*: Enter the following block of text:

```
REACTION_DATA_OUTPUT {
  {COUNT[nt_R_0B, WORLD]} => "./reaction_data/nt_R_0B.dat"
  {COUNT[nt_R_1Ba, WORLD]} => "./reaction_data/nt_R_1Ba.dat"
  {COUNT[nt_R_1Bb, WORLD]} => "./reaction_data/nt_R_1Bb.dat"
  {COUNT[nt_R_2B_C, WORLD]} => "./reaction_data/nt_R_2B_C.dat"
  {COUNT[nt_R_2B_O, WORLD]} => "./reaction_data/nt_R_2B_O.dat"
  {COUNT[nt, WORLD]} => "./reaction_data/nt.dat"
  }
```

Each file (.dat suffix) will be created automatically in a subdirectory called "reaction_data".

3. Save the file and quit.

3.5.5. Add Initial Distributions of Neurotransmitter Molecules

MCell's MDL supports several different ways to release molecules at different times and locations during simulations. In this case, the neurotransmitter molecules will originate inside each of the synaptic vesicles, and then will diffuse out through the expanding pore into the synaptic cleft. To create the initial distributions of molecules within the synaptic vesicles, we will use a simple method (once for each vesicle) that places a specified number of volume molecules at random locations within spherical bounds of specified diameter. By default, this will occur when the simulation is initialized, so the molecules will begin diffusing immediately. Each vesicle will initially contain 3,000 neurotransmitter molecules.

1. *Add the neurotransmitter release sites*: Use a text editor to modify the first of the main MDL files for the expanding fusion pore, "VesicleFusion_1.mdl" (*see* **Subheading 3.5.6** later). Add the following text at the end of the INSTANTIATE block (before its closing curly brace; *see* **Note 37**):

```
first_release_site SPHERICAL_RELEASE_SITE {
LOCATION = [-0.106, 0.021, 0.105]
MOLECULE = nt
NUMBER_TO_RELEASE = 3000
SITE_DIAMETER = 0.032
}
second_release_site SPHERICAL_RELEASE_SITE {
LOCATION = [0.106, -0.021, 0.105]
MOLECULE = nt
NUMBER_TO_RELEASE = 3000
SITE_DIAMETER = 0.032
}
```

2. Save the file and quit.

3.5.6. Final MDL File Setup

1. *Reuse the "Partitions.mdl" file*: Copy the "Partitions.mdl" file from **Subheading 3.4** to the current directory for the expanding pore MDL model.
2. *Add INCLUDE statements to the main MDL files for the checkpoint sequence*: When the pore expansion series was exported from Blender, a set of main and subordinate (included) MDL files was created. These files are numbered in sequence, e.g., the main files are named "VesicleFusion_1.mdl", "VesicleFusion_2.mdl", etc. As in **Subheading 3.3.6**, we now need to

add the remaining (newly created) INCLUDE file references to each of the ten main MDL files for the expanding pore. This can either be done by hand for each of the files, or all at once by entering the following two commands in succession at the command line:

sed -e "9aINCLUDE_FILE = \"Partitions.mdl\"\nINCLUDE_FILE = \"Molecules.mdl\"\nINCLUDE_FILE = \"Reactions.mdl\"\n" -i VesicleFusion_[1-9]*.mdl

sed -e "16aINCLUDE_FILE = \"RegionModifications.mdl\"\nINCLUDE_FILE = \"ReactionData.mdl\"\n" -i VesicleFusion_[1-9]*.mdl

3.5.7. Run the Expanding Pore Simulation

Highly accurate simulation of neurotransmitter diffusion through the pore would require many iterations and a very fine time step so that the average random walk step length would be much smaller than the pore radius at all times *(4, 5)*. In this simple example, however, each of the ten main MDL files is set to run for only one iteration with a relatively long time step (one microsecond). This will allow quick visualization of the pore expansion, but to see the transmitter escape and bind to receptors, we will need to run the final checkpoint segment for many more iterations.

1. *Increase the number of iterations for the final run of the checkpoint sequence*: Using a text editor, open the last of the main MDL files, "VesicleFusion_10.mdl". Change the first line from:

   ```
   iterations = 10
   ```

   to

   ```
   iterations = 500
   ```

   Then change the line:

   ```
   CHECKPOINT_ITERATIONS = 1
   ```

   to

   ```
   CHECKPOINT_ITERATIONS = 491
   ```

2. Save the file and quit.
3. *Run the simulation*: When the MDL files were exported from Blender, a separate "script" file was also created to run the ten MCell simulations in succession (rather than starting each by hand). The script file ("VesicleFusion.py") is written in a high-level command language called Python. Run the Python script by entering:

   ./VesicleFusion.py

   at the command line. You will see MCell's default run-time messages as the checkpoint runs execute in sequence, and the reaction and visualization files will be created.

3.6. Visualize MCell Results with DReAMM (Part 2)

We now use DReAMM to visualize the results of the expanding pore simulations. We will reuse the settings file created previously in **Subheading 3.4.4**, and will also apply some additional customizations and animation settings.

3.6.1. Import the MCell Visualization Data and Import the Settings File

1. *Start DReAMM*: Enter "dreamm" at the command line.
2. *Import visualization data*: Click "Import & Select" on the "Quick Controls" menu (labeled *1* in **Fig. 3A**). In the "Import & Select Objects" menu, click the ellipsis ("…") by the "Viz Data File" text box. Navigate to the "VesicleFusion_viz_data" directory and select the file "VesicleFusion.dx". Click "OK".
3. *Import the customization settings file*: In the "Read/Write Settings" menu, click the ellipsis ("…") by "Read File" and then navigate to and select the file "SynapseCustom.dx" that was created in **Subheading 3.4.4**. Click "OK". The "Dendrite" and "PresynapticBouton" meshes will appear with the rendering properties defined previously (*see* **Note 38**).
4. *Adjust the view*: Hit *Ctrl-f* to center the view. Now open the "View Control" menu (*Ctrl-v* from the "DReAMM Image Window") and then switch to a left view by setting "Set View" to "Left".
5. *Select all volume molecules*: In the "Import & Select Objects" menu, change "Add Selected" to "Add All". Next, click on "Volume Molecules" and click "Apply Operation Once".
6. *Select all surface molecules*: Click on "Surface Molecules" and click "Apply Operation Once".
7. *Close menus*: Click the "Close" button by "Import & Select" in the "Quick Controls" menu (labeled *5* in **Fig. 3A**).
8. *Apply previously defined volume molecule properties*: Click the "Set Rendering Prop". button in the "Quick Controls" menu (labeled *2* in **Fig. 3A**). In the "Rendering Properties" menu, click "Molecules" next to "List" (top center), then select "Ca", and click "Load from selected object". All of the rendering properties previously associated with "Ca" are now loaded in the "Properties" fields and in the "Colormap Editor" menu. To apply the same properties to the neurotransmitter molecules, select "nt" and deselect "Ca". Hit "Apply Operation Once", and all the "nt" pixels will change into yellow spheres (**Fig. 5C**).
9. *Apply previously defined surface molecule properties*: As in the preceding step, load the previously defined "VGCC_C" properties and apply them to "nt_R_0B". Then, load the properties for "VGCC_O" and apply them to "nt_R_2B_O". The unbound receptors will now appear as red arrows, and the double-bound open channel receptors will appear as green arrows.

10. *Assign rendering properties for the remaining receptor states*: Manually change "(Front) Color" to yellow by entering RGB values of 1, 1, and 0, respectively. Make sure that only "nt_R_1Ba" is highlighted in the list of object names. Click "Apply Operation (Once)". Now enter RGB values of 0, 0, 1 for blue, select only "nt_R_1Bb", and click "Apply Operation Once". Finally, enter RGB values of 0, 1, 1 for cyan, select only "nt_R_2B_C", and click "Apply Operation Once". The single-bound receptors now will be yellow or blue arrows, and the double-bound closed receptors will be cyan arrows. All surface molecules should now appear as colored arrows similar to their appearance in **Subheading 3.4.3** (**Fig. 5C–F**). If desired, try using the "receptor_1" or "receptor_2" glyphs for a more realistic appearance.
11. *Close menus*: Hit the "Close" button by "Set Rendering Prop". in the "Quick Controls" menu (labeled *6* in **Fig. 3A**).

3.6.2. Use Image Clipping

To see the expanding pore more clearly, we will use a clipping box (*see* **Note 39**) and visualize only the meshes and molecules in and around the synaptic cleft.

1. *Set the default view*: Hit *Ctrl-f*.
2. *Select the "Left" view*: Hit *Ctrl-v* to bring up the "View Control" menu and then select "Left" under "Set View".
3. *Align the clipping box*: Click on "Open Menu List" in the "Quick Controls" window (labeled *4* in **Fig. 3A**). This will open the "Menu List", which allows you to open individual DReAMM menus. Scroll down and click on "Image Clipping" to open the corresponding menu, and then click the button labeled "Align Clipping Box with Current View". This will orient the clipping box to the current viewing direction, so that the box's front and back faces are parallel to the screen and the left, right, top, and bottom limits are parallel to the corresponding screen edges.
4. *Set the clipping distance*: We now need to set the depth at which the clipping box begins; that is, the distance from the current camera location (look-from point) to the nearest point on the front face of the box. This is most easily done by "picking" a point on an object, and DReAMM will then calculate the distance to a plane that cuts through the picked point. In the "View Control" menu, select "Pick" under "Mode" and then choose "Clipping Distance" under "Pick(s)". Next, in the "DReAMM Image Window", click on the head of the presynaptic bouton. Note that the distance from the camera to the picked point now appears in the "Picked" field of "Near Distance" in the "Image Clipping" menu.

5. *Show the clipping box*: In the "Image Clipping" menu, click on the "Show Clipping Box" button. A yellow semitransparent box will appear and, due to its large default size, will fill the "DReAMM Image Window". With the image window active, hit *Ctrl-z* for "Zoom" mode (*see* **Note 29**) and then right click and drag to zoom out. If necessary, zoom in (left click and drag) or out again to see the entire clipping box in the image window.
6. *Adjust the clipping box size*: By default, the clipping box dimensions are 20 × 20 × 0.1 microns (width × height × thickness). Shrink the box's width and height by modifying the "Left", "Right", "Upper", and "Lower" limits in the "Image Clipping" menu. Continue until the box encloses only the synaptic vesicles and cleft, and then hide the box by clicking again on the "Show Clipping Box" button.
7. *Apply image clipping*: Click the "Apply Image Clipping" button. Only a small slice through the synaptic region will remain visible.
8. *Pan/zoom to reorient and magnify the view*: With the image window active, hit *Ctrl-g* for "Pan/Zoom" mode (*see* **Note 29**). Center the mouse pointer on the visible structures, and then left click and drag to enclose the objects within the marquee. When you release the mouse button, the view will be reoriented and magnified simultaneously.
9. *Rotate the view*: Hit *Ctrl-r* for "Rotate" mode. Point near the 3 o'clock position in the image window, and then right click and drag in a counterclockwise direction. The clipped objects will follow the pointer by rotating in the plane of the screen (a constrained rotation around the current view direction). Continue to the 12 o'clock position so that the synaptic cleft will be approximately horizontal. Next, point at the objects and left click and drag to perform a free rotation around the current look-to point. Drag primarily to the left or right to obtain an oblique view of the clipped objects.
10. *Adjust the clipping box thickness and fine-tune the dimensions*: In the "Image Clipping" menu, increase the thickness of the clipping box. A setting of about 0.35 microns should include both synaptic vesicles and most of the cleft space. With some minor adjustments to the box's dimensions, you can obtain a view similar to that shown in **Fig. 5H**. You may also find that you have to fine-tune the "Near Distance" for the box. This can be done by entering a value in the "Specify" field (start with a value very close to the indicated "Picked" value that was obtained in **step 4**) and then clicking on the "Specify" button.
11. *Play the time sequence*: Click the Play button on the "Sequence Control" to watch the pore open and the neurotransmitter

escape to bind to receptors. If necessary, use the Frame Controls to skip ahead in time (*see* **Subheading 3.4.3**, **step 10**).

3.6.3. Animate the Visualization

DReAMM includes many features to create sophisticated animations using keyframes (*see* **Note 40**). In this final section, we briefly introduce some of the possibilities by illustrating how to rotate the camera around the clipped objects.

1. *Temporarily turn off image clipping*: Again click the "Apply Image Clipping" button in the "Image Clipping" menu. By turning off image clipping temporarily, we will speed up and better illustrate the operations required to animate the camera.
2. *Open the "Make Animation" menus*: Click the "Make Animation" button on the "Quick Controls" menu (labeled *3* in **Fig. 3A**).
3. *Activate the "DReAMM Keyframe Editing Image Window"*: In the "Edit Keyframes" menu, click the "Show Camera Positions" and "Show Spline/Rotation Points" buttons. This will open a new image window that shows the model objects as well as glyphs representing the current camera and keyframe camera locations. Note that this new image window is fully interactive and operates with the same hot keys used in the main "DReAMM Image Window". With the new window active, hit *Ctrl-f* to center the view. Now hit *Ctrl-r* and rotate the view to see the model objects as well as the camera glyphs. The current camera location for the "DReAMM Image Window" view is shown by the common base of two orthogonal (perpendicular) arrows colored red and blue (look and up directions, respectively). By default, one keyframe camera location is also present, as indicated by orthogonal gold (look) and green (up) needle glyphs (line segments) with the numeral "1" alongside.
4. *Define the current view from the "DReAMM Image Window" as the starting point for the animation*: In the "Edit Keyframes" menu, click "Start New List" to begin a new list of keyframe data. The current camera location and view from the "DReAMM Image Window" will overwrite the existing default keyframe information. This will be indicated visually in the "Keyframe Editing Image Window" as the first keyframe camera glyph (labeled "1") becomes coincident with the current camera position glyph.
5. *Set up the rotation keyframes*: One of the most commonly desired animations is a simple rotation of the camera around some selected model objects. DReAMM allows you to do this operation in only a few simple steps. First, in the "Edit Keyframes" menu, click on the drop-down list that currently says "Add to End" and change it to "Compute Rotation Points".

Now click the "Apply Edit: Once" button. A set of magenta needle glyphs will appear in the "Keyframe Editing Image Window", showing potential camera locations and directions for the rotation. By default, a complete rotation is generated with 1° per step. Next, change the "Compute Rotation Points" drop-down setting to "Add Rotation to Range". Click the "Apply Edit: Once" button again. This finalizes the keyframe camera positions as indicated by the appearance of numbered keyframe camera glyphs.

6. *Close menus*: Click the "Close All" button in the "Quick Controls" menu (labeled *7* in **Fig. 3A**).
7. *Play the animation*: In the lower right-hand corner of the "Quick Controls" menu, change "Keyframe Mode" from "All Interactive" to "Keyframes" (labeled *8* in **Fig. 3A**). Click the Play button on "Sequence Control" (**Fig. 3A**) and watch the pore open and molecules diffuse while the camera rotates around the objects. If you leave the "Keyframe Editing Image Window" active, you will simultaneously see the current camera glyph moving from keyframe-to-keyframe around the model objects. To speed up animation playback, disable the "Keyframe Editing Image Window" by reopening the "Edit Keyframes" menu and clicking once again on the "Show Camera Positions" and "Show Spline/Rotation Points" buttons.
8. *Turn on image clipping*: Reopen the "Image Clipping" menu and click the "Apply Image Clipping" button again to see the animation of the clipped model objects. You can turn image clipping on and off during playback of the animation.
9. *Save your DReAMM settings*: As in **Subheading 3.4.4**, **step 1**, save your current DReAMM settings so that you can reload them and replay the animation at a later time.

4. Notes

1. There is presently a distinction between "modeling" software (e.g., Blender) and computer-aided design (CAD) software. Both allow the user to design 3D structures interactively, but "modeling" programs are specialized for animation and morphing of "smoother" or, in the present context, more "biological" shapes. CAD software, on the other hand, is oriented toward the design of objects for architectural or mechanical engineering, and may be integrated with computer-aided manufacturing (CAM) software and machinery.

2. Developer libraries typically are not installed by default with Mac OS X. The same problem can arise with some Linux distributions such as Ubuntu, Debian, or others.
3. The DX file format (.dx) was originally created by IBM for its commercial DataExplorer software, a large visual programming and visualization environment. Eventually DataExplorer (DX) was released as open source code (OpenDX), which we have now improved and expanded into PSC_DX for use with DReAMM and large-scale MCell models. We continue to use the native DX file format because it is very general and efficient for use with hierarchical assemblies of arbitrary mesh objects, each of which may be associated with multiple datasets of arbitrary type, and also annotated with a variety of metadata tags. Although we do not explicitly illustrate export of .dx files from Blender in this chapter, it is easily done using the supplied plug-in. Direct export from Blender to DReAMM can be very useful for rendering and creation of sophisticated animations, as well as specialized mesh editing operations. DReAMM can also export MDL files for use with MCell.
4. In Blender, the *X*-, *Y*-, and *Z*-axes are red, green, and blue, respectively, and the cardinal views (*XZ*-, *YZ*-, or *XY*-plane) can be accessed using number keys on the number pad (*1*, *3*, or *7*, respectively). While in a cardinal view, a visual key for the axes is present in the lower left corner of the "3D view" window. For example, if you hit *1* (*on the number pad*) for the *XZ*-view, you will see a horizontal red line labeled X and a vertical blue line labeled Z, indicating that the *XZ*-plane lies in the plane of the screen. The *Y*-axis is perpendicular to the screen in this view.
5. Most Blender hot keys work only when the cursor is located inside the "3D View" window.
6. A UV sphere is composed of quadrangular faces between lines of longitude and latitude, similar to a globe. Icosahedral spheres (composed of hexagons and pentagons; triangulated by Blender) are another option, but UV spheres are easier to use for the region definition and extrusion operations in this chapter.
7. Spatial units are arbitrary in Blender, but, once objects are exported to MDL files for use with MCell, dimensions will be interpreted as microns.
8. Although the cardinal views are sufficient for the operations in this chapter, it is often helpful to rotate around an object for a better view. To rotate around the currently selected object, click and drag using the middle mouse button. Certain operations may also require zooming and panning. Scroll up with the middle mouse button to zoom in, and scroll down to

zoom out. To pan, hold *Shift* and one of the following keys on the number pad: *8* (up), *2* (down), *4* (left), or *6* (right).

9. If you accidentally hit one of the number keys on the main keyboard instead of the number pad, you will move to a different layer and your current view will disappear. If this occurs, simply hit *1* on the main keyboard to return to layer 1.
10. To make the faces and central vertex, we first replicate the original topmost vertices using an extrude operation. Ordinarily this would be used to create a cylindrical extension, and in a later operation we extrude vertices in that fashion to create the spine shaft. In this case, however, we will extrude using a length of 0, so the new vertices will be exactly coincident with the original vertices. Then we will scale the radius of the extrusion to a value of 0, and this will move all of the new vertices to the desired central point in the plane of the opening. Finally, we will remove all of the unnecessary duplicate vertices, leaving all of the new triangles connected to a single central point.
11. It is useful to save periodic snapshots of all Blender files (.blend) using distinct names. This allows you to start over from a previous snapshot if something unexpected occurs.
12. "Object Mode" is used for operations on entire objects (e.g., selecting, moving, scaling), while "Edit Mode" is used for operations on selected faces or vertices.
13. The numerical suffix for the material name may differ (e.g., if a .blend file is reloaded). If this occurs, just ensure that you change the material name for the correct object.
14. There is an error (a bug) that sometimes appears when using Blender's Boolean operations. Under some conditions, the mesh vertices are reconnected incorrectly, leaving undesired faces and/or holes. This should not occur if the steps of the chapter are followed precisely. If the problem does occur, however, it can be fixed by deleting any extra faces and/or making the vertices around a hole into one or more faces as necessary.
15. Blender uses both "Global" and "Local" axes. There is only one set of "Global" axes, but each object has its own unique "Local" axes, which may or may not coincide with the "Global" axes. By using "Apply scale and rotation" in this step, the "Local" axes become coincident with the "Global" axes. This ensures that the objects will be aligned with the "Empty" object created in **Subheading 3.1.7**, **step 9**.
16. Each surface mesh element (face) has a front and back. The direction perpendicular (normal) to a face defines the face's normal vector. Vertices shared by more than one face have an associated normal vector defined as the average of the face normals. When objects are scaled (shrunken or enlarged),

each vertex is moved along its normal (inward or outward, respectively), and so the faces become smaller or larger. On the other hand, when a group of faces is moved rather than scaled, all of the vertices are moved along parallel vectors. In the particular case of this step, we want a spine head to move "out" or "in" along the axis of the spine neck. To do so, we can move the head along a vector defined by the average of its face normals. Since each spine head is radially symmetric around the spine axis, the average of the face normals will be the spine axis itself.

17. A full introduction to the MDL syntax is beyond the scope of these stand-alone examples. Please consult the MCell Reference Guide for additional information (http://www.mcell.psc.edu).

18. Every surface mesh object is composed of individual polygons. To describe any mesh in a general but compact way, all of the vertices (*x*,*y*,*z* coordinates) are listed first (see VERTEX_LIST in later example), and then the triangular faces are listed next (ELEMENT_CONNECTIONS). Each face is described by an array of index numbers that refer to the vertex list (a triple of vertex indices). For example, a face that connects the first three vertices could be listed as [0,1,2] (zero-based indices are used; the order in which the vertices are listed, together with the right hand rule, determines the front and back sides of the face). Thereafter, mesh regions can be named and defined using arrays of index numbers that refer to the list of faces. Thus, [0,1,2,3,4] would define a region composed of the first five faces. In an MDL file for one or more mesh objects, each object is defined by a POLYGON_LIST with a user-specified name, and that includes vertices, triangular connections, and regions (if any). For example:

```
my_mesh_name POLYGON_LIST {
VERTEX_LIST {
(list of all x,y,z coordinates)
}
ELEMENT_CONNECTIONS {
(list of all triangular faces, each defined by
   three vertex index numbers)
}
DEFINE_SURFACE_REGIONS {
my_first_region_name {
ELEMENT_LIST = [list of face index numbers]
}
my_second_region_name {
```

```
(...)
}
(...)
}
}
```

19. Checkpointing is a general term in large-scale computing and simply means that a running process (e.g., simulation) is stopped temporarily so that it can be restarted later. At the checkpoint, the current state of the process is written to a file that is subsequently read when computation resumes. With MCell, the checkpoint file includes the locations and states of all molecules, as well as other run-time parameters. When the simulation is restarted after a checkpoint, a complete set of input MDL files is read in addition to the checkpoint data. Thus, any new changes made in the MDL files (e.g., mesh geometry) are incorporated into the continuing simulation. In the example of this chapter, ten different sets of MDL files are used over a checkpointing sequence, and the radius of the fusion pore increases in each step. All of the diffusing neurotransmitter molecules are contained within the expanding pore or the synaptic cleft space later, and so diffusion from the vesicle through the pore can be simulated without concern that molecules escape from the changing pore geometry.
20. To check dimensions, click the "Edge Lengths" button in the "Buttons Window" under "Mesh Tools 1". This panel is on the far right of the "Buttons Window", and, depending on window size and resolution, you may have to scroll over to see it. If this is the case, scroll "up" using the middle mouse button while the cursor is over the "Buttons Window". Select "Edge Lengths", and values will appear for all highlighted edges (to three decimal places). Additional measurement capabilities are available through other online plug-ins for Blender.
21. This step may take some time to complete, depending on the speed of your computer. Also, you may find that Blender runs slowly after this step. This is due to a caching problem that can be eliminated simply by reopening the final saved .blend file and continuing.
22. MDL files are plain text files and can be edited using any available text editing program. Popular choices on Unix-like operating systems include vim, emacs, pico, or nano. In addition, the Writer program within OpenOffice (similar to MS Word within MS Office) can be used as long as the files are saved in plain text format.

23. A "block" denotes a set of MDL commands enclosed in curly braces, and some blocks may be nested (blocks within blocks).
24. As in other cell modeling languages, a "molecule name" in MCell's MDL is quite general and may actually correspond to a functional state, such as an open or closed conformation of a molecule that includes a transmembrane ion channel. The relationships between molecules and states are defined in subsequent reaction statements.
25. Molecules can be added to surface regions on meshes in several ways. In one approach the molecules are added directly by hand-editing a DEFINE_SURFACE_REGIONS block within a POLYGON_LIST (*see* **Note 18**). This can be quick and easy for small meshes, but hand-editing large mesh files is generally undesirable. Thus, in this chapter we illustrate use of the MODIFY_SURFACE_REGIONS command. In addition to other functions, MODIFY_SURFACE_REGIONS allows addition of molecules to mesh regions defined previously in other (included) MDL files. To add molecules, one specifies the name of the mesh region to be modified, the name of the molecules to be added, the number to be added, and the molecules' orientations with respect to the front and back of the surface.
26. In this example molecules are added to surface regions at the beginning of the simulation. It is also possible for molecules to appear and disappear during a simulation as reaction steps occur, and additional molecules may also be "released" in closed volumes and on surface regions at specified times during running simulations. When a molecule is produced by a reaction, its initial location depends on the location of the reactant(s), and may also reflect user-specified directionality with respect to a surface.
27. It is important to do visual checks often, especially before running long simulations with large models. Visual checks are typically done by running a short number of iterations for rapid visualization with DReAMM.
28. By default, DReAMM will use rendering properties that minimize the time and memory required to display images. It will also use "Software Rendering", which means that the calculations required to generate the images are performed on the computer's CPU, rather than the graphics processor on the video card. DReAMM can also use "Hardware Rendering", which does take advantage of graphics hardware and under many conditions will be much faster. However, as models increase in size and complexity, rendering in Hardware may become slower than in Software, or may not

be possible at all if insufficient memory is available on the video card. You can switch back and forth between Software and Hardware rendering by clicking "Options -> Rendering Options…" on the "DReAMM Image Window" menu bar, and then clicking on the "Software" or "Hardware" button under "Rendering Mode" in the pop-up "Rendering…" menu. Either Software or Hardware rendering may be used throughout the examples of this chapter, except when Software rendering must be used with "Image Clipping" (**Subheading 3.6.2** and **Note 39**).

29. With DReAMM, hot keys function only if both "Num Lock" and "Caps Lock" are off on the keyboard. Most DReAMM hot keys select different viewing modes and require that the "DReAMM Image Window" is selected. A sequence of view changes can be undone and redone using *Ctrl-u* and *Ctrl-d*, respectively, and a default view of centered objects can be obtained with *Ctrl-f*. Commonly used viewing modes include the following:
    - "Rotate" (*Ctrl-r*) – left click and drag for free rotation around the current look-to point, or right click and drag for constrained rotation around the current look direction.
    - "Navigate" (*Ctrl-n*) – left click, hold, and move the pointer toward an object to move the camera toward the object; middle click, hold, and move the pointer to pivot the camera (change the look-to point without changing the look-from point), or right click, hold, and move the pointer to move the camera away from objects. While in "Navigate" mode, the "View Control" menu includes sliders for forward or backward speed of motion and relative pivot speed, and also a selector button that allows the camera to look in different directions while traveling forward or backward (e.g., looking 45° to the right while traveling forward). For best results, use "Navigate" mode while using Hardware rendering.
    - "Zoom" (*Ctrl-z*) – left click and drag to zoom in on a region outlined by a centered marquee, or right click and drag to zoom out by fitting the current display into a centered marquee.
    - "Pan/Zoom" (*Ctrl-g*) – similar to "Zoom" mode except that you reorient the view and zoom simultaneously, by first pointing to the object of interest and then clicking and dragging a marquee around it.
30. Custom rendering properties are used to visualize particular molecules, region boundaries, and molecules obscured by meshes. Here, we illustrate the use of custom rendering properties to visualize the proper location and orientations

of the synaptic vesicles and their calcium binding sites, the VGCCs, and calcium ions.

31. Values can be changed by clicking the up/down arrows or by directly entering the desired value into the field and hitting *Enter*.
32. In a "real" model, use of a colormap ensures that all regions were designed properly. Any errors must be fixed and the modified model should be rerun and retested until all regions are verified visually.
33. To see a sharp transition between mesh elements with different colors, change the "Color Dependence" selector from "Vertices" to "Elements". If "Vertices" is used instead, color changes between mesh elements are blended to give the mesh a smooth appearance.
34. You must hit *Enter* after typing in a DReAMM text box before the changes will be recognized.
35. A directional glyph like an arrow is often a good choice when checking the orientation of surface molecules.
36. DReAMM settings files contain the list of currently selected objects, all keyframe data, and all assigned rendering properties, including colormaps. They can be reloaded later for the same or different visualization data, allowing reuse and exchange of settings. You can also choose to reload particular portions of a settings file. For example, you may wish to load rendering properties previously assigned to another model, but not the selected objects or keyframe information from the other model.
37. The locations (X,Y,Z coordinates) for each release site can be obtained with Blender or DReAMM, and are approximately centered within each vesicle. With Blender, you can display a median value for a selected set of vertices by hitting *n* and then clicking "Global" in the "Transform Properties" pop-up menu. With DReAMM, you can use the "Probes" menu to activate 3D cursors that can be moved around in space, with a display of their positions.
38. The settings file created previously includes rendering properties assigned for meshes and molecules. Since the meshes in the new model share the same names as the meshes in the previous model, the rendering properties are applied immediately. On the other hand, the molecules in the new model do not have the same names as the molecules in the previous model. Thus, at this point the new molecules are still rendered using default properties (pixels). In subsequent **Subheading 3.6.1**, **steps 8 and 9**, the previously defined rendering properties will be reassigned (copied) to the new molecules.

39. DReAMM includes two different types of clipping operations accessed through separate menus. “Data Clipping” allows you to specify separate *X*-, *Y*-, and/or *Z*-limits for different objects (“Meshes”, “Wireframes”, “Boundaries”, “Volume Molecules”, or “Surface Molecules”). Molecules with positions that lie within the limits are included for rendering, as are mesh elements (triangles) with included centers of mass. Thus, when using “Data Clipping” the amount of data to be rendered is reduced. If only a portion of a very large model is visualized, rendering speed can increase enormously. However, meshes clipped in this manner may have a jagged edge because complete triangles are removed. “Data Clipping” works when using either Software or Hardware rendering. In contrast, “Image Clipping” allows you to specify an arbitrarily oriented clipping box anywhere in space, and all objects contained within the box are shown with smooth cut edges. This increases rather than decreases the amount of computation, and with current versions of DReAMM this must be done with software rendering. For large datasets, “Data Clipping” and “Image Clipping” can be used in combination.
40. DReAMM animations are designed using interpolated keyframes. Each keyframe includes information about the camera position and view, lighting, depth cueing, and stereo visualization settings. In general, keyframes can be added and removed in a variety of ways, and all of the keyframe data parameters can be interpolated using a spline function over the entire keyframe sequence or a specified portion thereof. For additional information, see the DReAMM animation tutorials in *(15)*.

Acknowledgments

The authors thank Aji Janis and Gary Blumenthal for careful reading and testing of the examples in this chapter. This work was supported by NIH R01 GM068630 (JRS), P41 RR06009 (JRS), and F32 GM083473 (MD).

References

1. Tao, L. and Nicholson, C. (2004) Maximum geometrical hindrance to diffusion in brain extracellular space surrounding uniformly spaced convex cells. *J. Theor. Biol.* 229, 59–68.
2. Hrabe, J., Hrabetova, S., and Segeth, K. (2004) A model of effective diffusion and tortuosity in the extracellular space of the brain. *Biophys. J.* 87, 1606–1617.

3. Tao, A., Tao, L., and Nicholson, C. (2005) Cell cavities increase tortuosity in brain extracellular space. *J. Theor. Biol.* 234, 525–536.
4. Stiles, J. R., Van Helden, D., Bartol, T. M., Jr., Salpeter, E. E., and Salpeter, M. M. (1996) Miniature endplate current rise times <100 μs from improved dual recordings can be modeled with passive acetylcholine diffusion from a synaptic vesicle. *Proc. Natl. Acad. Sci. USA* 93, 5747–5752.
5. Stiles, J. R., Bartol, T. M., Jr., Salpeter, E. E., and Salpeter, M. M. (1998) Monte Carlo simulation of neurotransmitter release using MCell, a general simulator of cellular physiological processes, in *Computational Neuroscience* (Bower, J. M., ed.), Plenum, New York, NY, pp. 279–284.
6. Stiles, J. R., Bartol, T. M., Salpeter, M. M., Salpeter, E. E., and Sejnowski, T. J. (2001) Synaptic variability: new insights from reconstructions and Monte Carlo simulations with MCell, in *Synapses* (Cowan, W. M., Stevens, C. F., and Sudhof, T. C., eds.), Johns Hopkins University Press, Baltimore, MD, pp. 681–731.
7. Stiles, J. R. and Bartol, T. M. (2001) Monte Carlo methods for simulating realistic synaptic microphysiology using MCell, in *Computational Neuroscience: Realistic Modeling for Experimentalists* (De Schutter, E., ed.), CRC Press, Boca Raton, FL, pp. 87–127.
8. Pawlu, C., DiAntonio, A., and Heckmann, M. (2004) Postfusional control of quantal current shape. *Neuron* 43, 607–618.
9. Stiles, J. R., Ford, W. C., Pattillo, J. M., Deerinck, T. E., Ellisman, M. H., Bartol, T. M., and Sejnowski, T. J. (2004) Spatially realistic computational physiology: past, present, and future, in *Parallel Computing: Software Technology, Algorithms, Architectures & Applications* (Joubert, G. R., Nagel, W. E., Peters, F. J., and Walter, W. V., eds.), Elsevier, Amsterdam, pp. 685–694.
10. Coggan, J. S., Bartol, T. M., Esquenazi, E., Stiles, J. R., Lamont, S., Martone, M. E., Berg, D. K., Ellisman, M. H., and Sejnowski, T. J. (2005) Evidence for ectopic neurotransmission at a neuronal synapse. *Science* 309, 446–451.
11. He, L., Wu, X. S., Mohan, R., and Wu, L. G. (2006) Two modes of fusion pore opening revealed by cell-attached recordings at a synapse. *Nature* 444, 102–105.
12. Kerr, R.A., Bartol, T.M., Kaminsky, B., Dittrich, M., Chang, J.J.C, Baden, S.B., Sejnowski, T.J., and Stiles, J.R. Fast Monte Carlo simulation methods for biological reaction-diffusion systems in solution and on surfaces. *SIAM J. Sci. Comput.* 30, 3126–3149.
13. Kerr, R. A., Levine, H., Sejnowski, T. J., and Rappel, W. J. (2006) Division accuracy in a stochastic model of Min oscillations in Escherichia coli. *Proc. Natl. Acad. Sci. USA* 103, 347–352.
14. Koh, X., Srinivasan, B., Ching, H. S., and Levchenko, A. (2006) A 3D Monte Carlo analysis of the role of dyadic space geometry in spark generation. *Biophys. J.* 90, 1999–2014.
15. www.mcell.psc.edu and pages therein.
16. www.blender.org.
17. Synapse Web, Kristen M. Harris, PI, http://synapse-web.org/. The publicly available VRML file that contains the data for the reconstruction can be downloaded from www.synapse-web.org/anatomy/Ca1pyrmd/radiatum/index.stm.

Updates to the model construction steps in the book chapter when using Blender version 2.49:

There are two significant changes in the given steps when using Blender version 2.49 compared to version 2.45, for which the text was originally designed. First, in Blender 2.49, new objects are created with their local axes aligned to the global axes, as opposed to being aligned to the current view in version 2.45. Second, the order in which an object is selected affects how a Boolean operation is applied, and this order was switched after Blender 2.45. The rest of the changes are fairly trivial and less likely to cause major problems. Finally, although this document continually references Blender version 2.49 (the newest version as of 07/17/2009), most of these changes are also true for versions 2.46 through 2.48 as well.

3.1.1.3

difference: Blender 2.49 creates objects in "Object Mode" instead of "Edit Mode".
correction: After the sphere is created, hit *Tab* to switch into "Edit Mode".

3.1.4.1-2

difference: Blender 2.49 creates objects in "Object Mode" instead of "Edit Mode", and objects are not created relative to the current view.
correction: After step 3.1.4.1, hit *Tab* to switch into "Edit Mode"and ignore step 3.1.4.2, since the object is already oriented properly.

3.1.7.1

difference: Blender 2.49 creates objects in "Object Mode" instead of "Edit Mode", and objects are not created relative to the current view.
correction: After the cylinder is created, hit *r*, then *x*, type *90*, and then hit *Enter* to rotate the object 90 degrees along the x-axis to get it into the desired orientation. Hit *Tab* to switch into "Edit Mode". Hit *r*, type *11.25*, and hit *Enter* to align one of the cylinder's faces with the base of the spine object.

3.1.7.6

difference: The order in which objects are selected affects Boolean operations in Blender. The effect of this order is reversed between the two versions of Blender. As a result, in Blender 2.49, an object created from a Boolean operation has a center based off of the first object selected as opposed to the second.
correction: Hit *Tab* to go into "Object Mode". Deselect the spines by hitting *a*. Then right click on the shaft. (Note: it may be hard to see the shaft from an overhead view). Hold *Shift* and right click on the spines. Follow the remainder of the steps in this subsection starting with the fourth sentence.

3.1.7.8

difference: The wording of the pop-up menu has changed when you hit *Ctrl-a* in Blender 2.49.
correction: Select "Scale and Rotation to ObData" under the new "Apply Object" pop-up menu, instead of "Apply scale and rotation". Follow all the other steps in this subsection starting with the second sentence.

3.2.4

difference: The order in which objects are selected affects a Boolean operation in Blender. The effect of this order is reversed between the two versions of Blender. Naturally, when using a "Difference", the resulting mesh that is created looks very different depending on which object is selected first.
correction: Hit *Tab* to go into "Object Mode". Right click on the "Presynaptic Bouton". Hold *Shift* and right click on the vesicles. Follow the remainder of the steps outlined in this subsection starting with the third sentence.

3.2.10

difference: The "Action Editor" now controls different functions besides just "ShapeKeys", such as the "Grease Pencil" in Blender 2.49.
correction: After you switch to the "Action Editor" window, you need to explicitly specify that you want to work with "ShapeKeys", by clicking the "Editor Mode" drop-down list and selecting "ShapeKey Editor". Follow the remainder of the steps in this subsection starting with the third sentence.