Published in final edited form as:
Neuroinformatics. 2010 June ; 8(2): 63–82. doi:10.1007/s12021-010-9066-x.

Cellular Dynamic Simulator: An Event Driven Molecular Simulation Environment for Cellular Physiology

Michael J. Byrne, **M. Neal Waxham**, and **Yoshihisa Kubota**
Department of Neurobiology and Anatomy, University of Texas Medical School, 6431 Fannin, Houston, TX 77030, USA

Abstract

In this paper, we present the Cellular Dynamic Simulator (CDS) for simulating diffusion and chemical reactions within crowded molecular environments. CDS is based on a novel event driven algorithm specifically designed for precise calculation of the timing of collisions, reactions and other events for each individual molecule in the environment. Generic mesh based compartments allow the creation / importation of very simple or detailed cellular structures that exist in a 3D environment. Multiple levels of compartments and static obstacles can be used to create a dense environment to mimic cellular boundaries and the intracellular space. The CDS algorithm takes into account volume exclusion and molecular crowding that may impact signaling cascades in small sub-cellular compartments such as dendritic spines. With the CDS, we can simulate simple enzyme reactions; aggregation, channel transport, as well as highly complicated chemical reaction networks of both freely diffusing and membrane bound multi-protein complexes. Components of the CDS are generally defined such that the simulator can be applied to a wide range of environments in terms of scale and level of detail. Through an initialization GUI, a simple simulation environment can be created and populated within minutes yet is powerful enough to design complex 3D cellular architecture. The initialization tool allows visual confirmation of the environment construction prior to execution by the simulator. This paper describes the CDS algorithm, design implementation, and provides an overview of the types of features available and the utility of those features are highlighted in demonstrations.

Keywords

Event driven algorithm; Molecular dynamics; Simulator; Molecular simulation; Monte Carlo; Stochastic simulation

Introduction

Investigating sub-cellular molecular dynamics requires a simulation tool with a high resolution of spatial detail (Stiles et al. 1996; Andrews and Bray 2004; Boulianne et al. 2008; Ridgway et al. 2008). Molecular diffusion, aggregation, channel transport, and complicated chemical reaction networks of both freely diffusing and membrane bound multi-protein channel receptors are common features of molecular neurophysiology. Unlike traditional computational methods, such as those employing reaction-diffusion equations, particle-based

Yoshihisa.Kubota@uth.tmc.edu.

Information Sharing Statement
The Cellular Dynamic Simulator package and links to required Java libraries can be found on the CDS website (http://nba.uth.tmc.edu/cds/). The package includes a user guide and example simulations.

single molecule level stochastic simulations offer deeper insight into spatial and temporal dynamics that can be used to model phenomena such as translocation and trafficking (Bredt and Nicoll 2003; Groc and Choquet 2006). These simulations are also capable of supporting non-uniform geometry of the cellular boundary and intracellular organelles which may greatly influence the diffusion and trafficking of signaling (Santamaria et al. 2006; Hugel et al. 2008) and may hinder molecular diffusion significantly.

The Cellular Dynamic Simulator (CDS) was designed to simulate these types of crowded environments and molecular behaviors with the fundamental feature of handling diffusion and reactions in volume excluded spaces. Three dimensional particle-based simulators may be constructed in a variety of ways given the particular environments they are designed to represent. For example, the spatial coordinates of molecules can be continuous or discretized (on lattice). The molecules can be modeled as size-less point particles or the size of molecules can be taken into account when collisions between molecules are computed. Also, the algorithm that drives the simulator may be time-driven or event-driven (Sigurgeirsson et al. 2001). CDS uses continuous space and an event-driven algorithm. The event-driven algorithm (Sigurgeirsson et al. 2001) iterates from one event to the following event where an event is defined as any occurrence within the simulation environment including a collision, reaction, transport, stimulus, etc. This algorithm, when combined with an accurate collision detection scheme, guarantees that no collision or reaction is missed. Accurate event detection is critical in crowded environments like the inside of cells where the collision probability becomes very high.

CDS as a software package includes three applications: an initialization interface, the simulator itself and a post simulation visualization tool. CDS is generally designed to support a variety of uses and does not constrain the user into a specific type of simulation. This provides tremendous power and flexibility to the user to address complex problems. A variety of behavior, such as mobility, degradation, and numerous possibilities for molecular interactions, can be achieved from molecular species by assigning properties or a combination of properties. Assigning properties to species is made easy by the initialization interface, but its main purpose is to aid in creating the structural shape of compartments. Making a tool that simplifies this process yet allows for the creation of complex designs is essential for a simulator of this nature to be accessible to a wide audience.

The paper is organized as follows. First, we explain the outline of the algorithm in “Event-Driven Molecular Collision and Chemical Reaction Model”. Then, the implementation (“CDS Implementation”), design and features of the CDS simulator (“Features”), and initializer/visualizer (“Supplementary Tools”) will be described in detail together with specific use examples. In addition, we compare simulation results with Smoluchowski’s theory for the diffusion-limited irreversible chemical reaction ($A+B \rightarrow C$) and provide a numerical validation of the algorithm (“Features”). The goal of this paper is to introduce the CDS to potential users in the field of molecular neurophysiology and neurobiology, however, the open framework of the simulator provides opportunities to all areas of cell biology.

Event-Driven Molecular Collision and Chemical Reaction Model

The CDS is a novel particle-based, event-driven molecular collision and chemical reaction algorithmic scheme to simulate the impacts of stochasticity, volume-exclusion, and molecular crowding on cellular signaling systems. In this section, we introduce the principle underlying this new algorithm and explain the strategy we use to simulate molecular diffusion and chemical reactions in a crowded cellular interior. In “Overview of Pre-Existing Algorithms”, we give a brief overview of the commonly used stochastic simulators in the field and discuss the necessity of the event-driven scheme. In "Outline of the CDS Algorithms", we describe the outline of the algorithm.

Overview of Pre-Existing Algorithms

The stochastic fluctuation of molecular dynamics is an important issue when we study the function of small subcellular compartments such as dendritic spines. Conventional PDE approach such as Virtual Cell (Moraru et al. 2008) does not address this stochastic fluctuation. Some stochastic reaction-diffusion simulators, STEPS (Wils and De Schutter 2009) and MesoRD (Hattne et al. 2005), incorporate stochastic fluctuation and spatial gradients of molecules. These algorithms are efficient and useful tools to understand the overall molecular dynamics of the cell. However, in the reaction-diffusion master equation approach, the size of the compartment cannot be arbitrarily small. The error increases as the size of compartment decreases as pointed out by Erban and Chapman (2009). Thus, the spatial resolution of the reaction-diffusion stochastic simulator has limitations.

Several simulation packages use the particle-based stochastic scheme. These include GridCell (Boulianne et al. 2008), MCell (Kerr et al. 2008), Smoldyn (Andrews and Bray 2004), and the coarse-grained molecular simulator described by Ridgway et al. (2008). These single-molecule level simulation methods overcome the limitation of the spatial resolution associated with the stochastic reaction-diffusion algorithms.

In GridCell, molecules are moved according to a cubic grid structure; therefore, the motions of molecules are not continuous. In MCell and Smoldyn, individual diffusing molecules are treated as size-less particles and the volume exclusion principle is not fully taken into account. To overcome this problem, Ridgway et al. (2008) takes into account the size of molecules and moves a particle in random direction using a fixed time step (Δt). For a molecule with diffusion coefficient D, the jump length is $\sqrt{6D\Delta t}$. The collision is detected when the distance of two molecules becomes less than the sum of their radii at the time Δt. If, upon collision, no reaction occurs, the move is rejected and the molecules are returned to their original positions.

This collision detection algorithm may be problematic. Two particles may collide between time 0 and Δt and may not overlap with each other at the time Δt. This undetected collision between time 0 and Δt may influence the trajectories of nearby molecules. The problem produced by this chain of interdependence of molecular collisions is amplified when the environment is crowded as in the cell's cytoplasm. In addition, one might incorrectly alter the motion of molecules by moving particles back to their original positions. These complications may result in an unexpected reduction or distortion of molecular diffusion, especially in a crowded environment. Interestingly, several simulation works including Ridgway et al. (2008) predicted anomalous diffusion, as opposed to normal (i.e., Brownian) diffusion, in crowded cellular environment with fixed or mobile obstacles. However, experimental measurements often report a reduction of molecular diffusion but not necessarily anomalous diffusion in the cytoplasm (reviewed in Dix and Verkman 2008). The collision detection scheme with a fixed time-step may be one of the reasons for the discrepancy between simulations and experiments.

In the molecular dynamics (MD) literature, the simulation schemes are classified into two types: event-driven and time-driven. In the event-driven scheme, one calculates the temporal order of all molecular collisions and executes them one-by-one so that no collision event is ever missed. While in the time-driven scheme, one uses a fixed time step to move the molecules. As we pointed out, no matter how small this time step is, one might miss the potential collision between molecules in a crowded environment and one collision event might influence the trajectories of neighboring molecules (Sigurgeirsson et al. 2001). Thus, the event-driven scheme is the only accurate way to simulate molecular collision in a crowded environment.

Translating this event-drive scheme to a mesoscopic Monte Carlo algorithm is, however, not straightforward. In the case of the classical MD scheme, the particles move according to Newtonian mechanics. The calculation of collision time between a pair of molecules is relatively simple. The detection of collision between two moving Brownian particles requires special care and the development of a computationally efficient algorithm of this type is still an open area of research. Therefore, it is critical to have a flexible platform that allows users to test various event-driven algorithms under the same conditions. We have established this important platform called the CDS.

Outline of the CDS Algorithms

The development of event-driven algorithm was inspired by the first passage concept used in the famous walk-on-spheres (WOS) method and related first passage time (FPT) algorithm (Muller 1956; Hwang et al. 2001). The CDS algorithm uses the discretized Brownian trajectories and does not directly use the exact formulation of the first passage theory. However, its underlying principle and some of the mathematical formulations are closely related to the first passage process. For this reason, we give a brief description of the WOS method first and then introduce the CDS algorithm using the first passage process theory (Redner 2001).

To explain the WOS method, let us suppose there are three molecules in the simulation as illustrated in Fig. 1a. Two of them (A_1 and A_2 molecules) are large immobile spherical molecules of radius of R_A while the other one (B molecule) is a smaller diffusing particle of radius R_B. To move particle B, we draw the largest circle around B of radius (R_B+σ) in which the B molecules can move within this circle without colliding into the A_1 or A_2 molecule. We call this circle as first passage sphere (FPS) of B (circle, dashed line). Note the center of the B molecule remains in a sphere of radius σ (smaller than FPS); we take into account of the size of the B molecule (hard sphere volume exclusion). Then, the mean first passage time (FPT=τ) of B molecule to touch (reach) this first passage sphere is given by

$$\tau=\sigma^2/(6D_B) \tag{1}$$

(Klein 1952;Redner 2001) where D_B is the diffusion coefficient of the B molecule. We randomly select a new location of the B molecules so that the B molecule touches the surface of the first passage sphere (FPS) and advance the time clock by τ. We repeat this process (= “first passage propagation”) until the B molecule falls within a thin-shell of the A_2 molecule when we decide that the B molecule has collided into the A_2 molecule. Alternatively, Hwang et al. (2001) draw a small circle that intersects the target molecule (A_2) when the B molecule comes sufficiently close to it (Fig. 1b). They calculate the mean hitting probability and the mean time for the B molecule to reach A_2 (the solid line and the region indicated by S_1 in Fig. 1b). This way, we can calculate the hitting time of the B molecule (in this case, the target is A_2). Note the time advancement and jump length differ from one first passage propagation to another. The WOS method does not use a fixed time step.

Opplestrup et al. (2006) proposed an extension of the WOS method to the multiple moving particle system. However, we have come to realize that elaborate theoretical analyses and derivations of mathematical formulae are required to establish such an algorithm for accurately simulating chemical reaction kinetics. Furthermore, such a detailed algorithm may be computationally expensive and slow. The CDS is designed flexible enough to implement any event-driven algorithm but we have decided to develop a simple yet accurate algorithm that approximates the first passage scheme.

The core principle of this simplification lies in Eq. 1. Note Eq. 1 is consistent with the conventional notion of the Brownian diffusion in which the mean-square displacement of a molecule of diffusion coefficient D_B over a time period τ is equal to $\sigma^2 = 6D_B\tau$. In other words, the first passage propagation in the WOS method in Fig. 1a is (statistically) equivalent to the discrete Brownian trajectory in which the jump length is defined by the distance between the B molecule and its target. The Eq. 1 defines the time advancement during this jump and the direction of the move is randomly selected as in the discretized Brownian motion.

We extend this scheme to the multiple particle system and introduce a new event-driven algorithm using the discretized Brownian trajectories. In fact, the only simplification we need to introduce in creating this new algorithm is the step where we calculate the first passage probability. Suppose we have two nearby particles (A and B in Fig. 1c) that potentially collide with each other. To generate discretized Brownian trajectories for these molecules, we use a "tentative" sample time step *dt* and select random jump directions for each molecule (the arrows in Fig. 1c). As usual the jump lengths for these molecules are set to $\sqrt{6D_A dt}$ and $\sqrt{6D_B dt}$ where D_A and D_B represent the diffusion coefficients of A and B molecules, respectively. Knowing the current locations of molecules and the direction of jump (indicated by the arrows in Fig. 1c) and the jump length, we can decide whether A and B molecules collide with each other or not and, if they collide, we can calculate where the collisions happens. Here we decide that A and B molecules collide when the center-to-center distance of these molecules becomes equal to the sum of their radii after they have traveled distances $\sqrt{6D_A \tau}$ and $\sqrt{6D_B \tau}$ where $0 < \tau < dt$. Note these travel distances are consistent with Eq. 1. We can calculate (and determine) the existence of such τ by solving a simple quadratic equation. After the collision, and if A and B do not react, we assign new directions of movement for these molecules.

For the multiple particle system, we use the collision detection scheme in Fig. 1c to calculate all possible collision times (between time 0 and *dt*) for all molecular pairs and generate an ordered list (heap) of collision events. This allows us to execute the collision events one-by-one according to their temporal order. While executing collisions, we advance the time clock and move all other molecules as well. Upon collision and if a binary reaction is selected, two molecules recombine to generate a new molecule (See "Features" for more details). If no collision happens, we move all molecules as in the WOS method. In other words, the CDS algorithm is consistent with the WOS method or the first passage algorithm except that the collision detection scheme in Fig. 1b is an approximation. Thus, the molecular diffusions and chemical reactions under the CDS are accurate up to this approximation. On the other hand, it overcomes the shortcomings of the time-driven Brownian dynamics such as MCell, Smoldyn and the coarse-grained molecular simulation by Ridgway et al. (2008).

In what follows, we describe other details of the algorithms. For example, we need to explain how other events such as Ca^{2+} flow through channels, dissociation reaction, and aggregation can be implemented under the event-driven scheme. We first describe the overall flowchart of the algorithm ("CDS Implementation") and explain the individual event in "Features".

CDS Implementation

The CDS package is programmed entirely in Java. This allows CDS to be copied between various operating systems and executed with no need to compile code. Both the initialization tool ("CDS Initializer") and the visualization tool ("CDS Visualizer") are graphical applications while the simulator itself is non-graphical and runs through a command prompt or terminal. The separation of these tools was designed for desktop development and analysis and large scale remote computation. Due to the stochastic nature of the underlying algorithms, a simulation may require many executions with the results averaged. The Initializer allows for the simulator to be executed within the graphical environment which is practical for short test simulations and convenience to new users. Both the Initializer and simulator are self-contained and require no additional libraries other than a current version of the Java Runtime Environment

(JRE 1.6 or newer). The Visualizer requires Java3D and Java Media Framework libraries for the 3D rendering and movie creation, respectively.

Event Driven Algorithm

As mentioned earlier, CDS is based on an event-driven algorithm (Sigurgeirsson et al. 2001). In CDS, an event is anything that can occur within the simulation environment including collisions, reactions, transport, user-defined stimuli, etc. The event-driven algorithm iterates from one event to the next. For every possible event, the time until next occurrence is computed and stored. The simulator searches for the next earliest occurring event, and then all molecules within the simulation environment are advanced up to that time (Fig. 2a). That event is processed, and the simulator continues the loop, searching for the next event. Accuracy and integrity of the simulator depends on accurate collision detection between all diffusing and static entities, regardless of the density of the simulation environment. When each collision is treated as an event, no collisions are missed by the simulator and this allows for conservation of populations within closed volumes and precise interactions between molecules, obstacles, and membranes.

Meshes

Meshes are a universal method of describing arbitrary 3D shapes and have been used to describe 3D reconstructions of sub-cellular compartments such as dendritic trees (Fiala and Harris 2001). All 3D simulators that offer off-lattice diffusion and custom geometry use some type of mesh construction for compartments. A mesh is comprised of vertices and faces where each face is a triangle defined by three vertices. The vertices are arranged in each triangle such that the normal vector (the vector perpendicular to the plane that the triangle resides on) points in the outside direction (assuming that it is a closed compartment). This is needed for the calculation of volume and as a way to determine intra versus extracellular space when molecules interact with the membrane. Surface area is the sum of the areas of all triangles in the mesh. The standard format that CDS uses is the Wavefront object format (.obj) which is a very simple file format for meshes. The initialization tool also allows importation of VRML 1.0 (.wrl) meshes. The object format allows faces to be grouped together by name. This way, the meshes describing compartments may be labeled with sub-regions so that each region has unique properties, e.g., neck or head of dendritic spines. The proteins created by these meshes can also be broken into multiple sub-units using this same strategy.

In addition to meshes, we have introduced primitive spheres to lower memory overhead when simulating simple elements such as ions. Using a mesh to describe a diffusing molecule is typically only required when simulating multiple subunit protein complexes since a sphere cannot be broken down into sub-surfaces.

Programming Design

All physical entities that exist in CDS are managed by a World object which also serves as the link to the event-driven algorithm. This object is abstract which means that some methods are not defined. In the World object, the abstract methods are those that control molecular movement or collision detection. This method of programming allows many different types of worlds to be created to handle specific simulation environments. Environments not currently included in CDS may be added later with no changes to the current code. Three types of worlds are presently packaged with CDS. *SimpleWorld* is an infinite space (lack of a boundary) where collision detection of a diffusing element is checked against all other compartments and physical elements in the world. This world is very useful for simple simulations, as it does not include the overhead of tracking molecule locations involved with optimization strategies. *GridWorld* subdivides the simulation space into small cubic cells such that a diffusing element only performs collision detection against elements in the neighboring cells. The size of the cubic cell is optimized based upon the size of the world and the greatest diffusion coefficient such that no molecule can travel further than half a cell width before its trajectory is recalculated. This cell method is used in other Monte Carlo and Molecular Dynamics simulations to speed up the collision detection (Sigurgeirsson et al. 2001; Donev et al. 2005). An absorbing boundary can be also defined that if hit, the element is removed from the simulation environment. *PeriodicWorld* divides the space in the same manner as *GridWorld*, but the boundary is periodic. A special care is necessary for collision detection in *PeriodicWorld* for those molecules near or crossing the boundary. The CDS takes into account of this special nature of periodic boundary collision detection.

Three types of Java classes define all physical objects in CDS, an element, compartment, and moving compartment. An element is any potentially diffusing molecule and can range from a fast diffusing ion to a static obstacle to a membrane bound protein complex. A compartment is a static structure that has a membrane and internal volume. Moving compartments are functionally similar to moving elements, but contain an internal volume that may be populated by diffusing molecules. All three share a hierarchy of parent objects (defined by Java classes) such that they share most of the same code and have similar properties (Fig. 2b). The root object is a shape which can either be a mesh or a primitive sphere. A Surface object extends the shape and adds properties such as reactions, an internal state, binding sites, and many other possibilities that will be described in the following sections. A Structure object extends the Surface and adds properties of location, trajectory, rotation, and can also store an array of sub-surfaces. Each sub-surface may be a subset of the overall mesh and may have its own unique properties and reactions. MovingStructure adds functionality for diffusion, aggregation and collisions. Compartment objects are static and have internal volumes that may be populated, while child objects of moving structures are potentially diffusing. In maintaining the generality of CDS and potential broad uses, these terms will be used throughout this paper, but it must be remembered that an element can represent a variety of molecules and structures.

Parallelization

The computational cost of the event-driven algorithm is highly variable depending on the crowding of the environment and the number of fast diffusing elements. Common optimization strategies are utilized such as subdividing the world (Donev et al. 2005; Sigurgeirsson et al. 2001); however, parallelization with multi core processors is an additional option for reducing the computation time of CDS. Time consuming operations of CDS are parallelized using Java threads which take advantage of multi-core SMP machines (e.g. an Intel quad core desktop processor). Collision detection typically consumes about 75% of the total simulation time. When CDS is run with a single thread, collision detection is performed on each diffusing element one at a time. Increasing to two threads divides the elements such that half are processed by one thread and the rest by the other thread. The elements are assigned to each thread in an alternating fashion such that both threads will perform collision detection on an equal number of elements of a similar diffusion coefficient and molecular size (based on how elements are added to storage arrays). Without any additional thread management, both threads should complete collision detection in a similar amount of time. The same style of parallelization is also applied to the method that advances the world by the event time, which is the second most time consuming portion of CDS. Both cases allow for easy multi-threading since shared data in memory is only read during these times and not written. CDS is single threaded at all other times during the simulation.

Features

CDS was designed with the intention of simulating small sub-cellular compartments and processes involved in those compartments. Initial simulations that would benefit from CDS involve signaling cascades in a single compartment; however, future interests were kept in mind during development. Systems were considered such as whole synapses where there may be multiple adjacent terminals each with complex second messenger cascades, release of transmitter through multiple processes, and also feedback between terminals dependent on their respective levels of activity. A simulator must be able to reproduce certain behaviors other than reaction diffusion systems to capture all the dynamics in these systems. For example, if a pump or channel is placed on an impermeable membrane separating two volumes, how is the interaction and relocation of ions implemented? If proteins can aggregate at a membrane, how is the complex created from many independent diffusing proteins, and how does it break apart? Described in the sections below are the currently implemented strategies in CDS and explanations on how they may be used to generate these and other molecular behaviors.

Diffusion and Collision Detection

CDS uses discretized Brownian motion to simulate molecular diffusion. These discretized Brownian motion trajectories are used to compute the next collision time between molecules ("Event-Driven Molecular Collision and Chemical Reaction Model"). As explained earlier, we use a "sample time interval" (typically 10~100 ns) to generate the initial Brownian trajectories and to probe the space to detect the possible molecular collisions. However, the collisions and other events are executed one-by-one and the time clock is advanced by these events and not by a fixed time step.

There are two principle methods for collision detection, sphere on triangle tests and sphere on sphere tests. In all cases when a collision is detected, the value returned is the time until the collision will take place. If the simulation environment is advanced by a particular collision time, then the two colliding entities will be touching. Every element has a collision queue which stores all possible collisions it could potentially be involved in. The collision data is stored by order of collision time where the next occurring collision is stored at index zero of the array. When the event-driven algorithm searches for the next occurring event, it only checks the first index of the collision queue. The collision data stored in the queue include the IDs of the elements and/or compartments involved in the collision and also the IDs of the faces that would be touching if the collision were to occur. The collision detection only needs to be performed when the trajectory of a molecule is changed or at the start of each Monte Carlo step when all molecules have their trajectory re-randomized.

Diffusing molecules perform collision detection against compartments on a per face basis. The test is called a sphere on triangle test (Fauerby 2003) which checks against the triangle plane and against edges and vertices if the sphere lies within the plane of the triangle upon collision. The collision detection between diffusing molecules is performed with a so-called sphere on sphere test as in Sigurgeirsson et al. (2001). If the molecule is described as a mesh, the sphere radius used is the minimum radius that surrounds all the vertices of the mesh.

The static element that a particle may collide into could have a very complicated and odd shape such as actin filaments. For these odd-shaped elements, the sphere on sphere test is used as a first pass for the purpose of optimization. If a collision is registered on the first pass then a sphere on triangle test is used against the faces of the static element. If the first pass fails, then a collision is not possible. This allows for accurate collision detection with odd-shaped meshes where the binding sphere of the mesh may be distant from some faces. In cases where the diffusing molecule is already overlapped with the static element's maximum radius, then collision detection goes straight to the sphere on triangle test.

Binary Reactions

General chemical reactions can be classified into three categories: binary reaction (Fig. 3, A1 and A2), dissociation reaction (Fig. 3, A3), and isomerization (Fig. 3, C2–C3). The binary reactions are further classified into four different subtypes for the purposes of the CDS. First is a simple recombination reaction (see Eq. 2 below). The second is the binding of a ligand to a receptor molecule that has multiple ligand binding sites (Eq. 3). The third is a binary reaction between a multi-subunit protein and its reactant (Eq. 4 and Fig. 3, B1–B4). Lastly, the reactant molecules A and B can have multiple activation states and the binary reaction in Eq. 5 takes place between A and B only when they are in designated activation states:

$$A+B \rightarrow C{:}1.0 \tag{2}$$

$$A.state1+B \rightarrow A.state2{:}1.0 \tag{3}$$

$$A.surface1.state1+B \rightarrow A.surface1.state2{:}1.0 \tag{4}$$

$$A.state1+B.state1 \rightarrow A.state2{:}1.0 \tag{5}$$

Binary reactions take place upon collision of two molecules (Fig. 3, A1). The decision as to whether or not a reaction will take place is typically determined during the collision detection process using a physical radius of participant molecules. Occasionally, a reaction radius, different from physical radius, will be use to detect “reactive collision” between molecules. In CDS, a reaction radius may be specified by users which will override the physical radius during collision detection between reactive elements. The probability that a reaction will take place is determined by a probability value between 0.0 and 1.0 where a value of 1.0 results in a 100% chance of the reaction upon collision (i.e., diffusion-limited reaction). The relation between this microscopic reaction probability and the macroscopic reaction rate is established by a look-up table similar to Andrews and Bray (2004) and its modification made by Erban and Chapman (2009). We needed to improve their formulation using the reaction diffusion equation with a boundary condition that reflects the volume exclusion principle (Carslaw and Jaeger 1986; Kim and Shin 1999; van Zon and ten Wolde 2005). However, the simple simulation experiments help users to establish this look-up table without relying on this theory. CDS selects a random number between 0 and 1. If the random number is less than the reaction probability, the binary reaction will be selected. If the reaction is in the form of Eq. 2, then reactants A and B will be physically removed from the world and a new molecule C will be added at the midpoint between A and B’s prior location (by default). The mass of molecule C is the sum of molecules A and B. Therefore the size of molecule C can be different and larger than A and B. In case the newly generated molecule C overlaps with nearby molecule(s), CDS will search for free space to place C molecule near the initially calculated point. During each iteration, a new direction from the initial point is randomly generated. Every 10 iterations, the distance from the initial point is increased by the radius of C. If after 50 iterations no free space is found, then the reaction does not take place. The original A and B molecules are placed back to the position where they collide and their new trajectory will be computed. There is no theory that presently exists to determine the location for molecule C under the crowding situation. CDS allows the users to define or assign the location of C molecule differently than the default. We have extensively tested our default strategy for non-crowding situations and the simulation results converge to the classical chemical reaction theory (see Fig. 4a below).

If the reaction is in the form of Eq. 3, then reactant B will be removed from the world and A will retain its current location and trajectory (“Events and Actions” describes how additional behavior can be added to this default implementation). In all types of association reactions where the reactants are removed from the World, a reference to those reactants is stored in the product. If the product eventually dissociates back to those reactants, they will have the same unique IDs that they were assigned when the element was first created.

Special care is necessary with those association reactions in which two possible reaction pathways are possible between the same reactants. For example, species A can bind two B molecules, each one to a different terminus which can be designated by a different state of A (e.g. *A.1+ B→A.2 : 0.15* and *A.1 + B→A.3 : 0.25*). CDS selects a random number between 0 and 1. If the random number is less than the probability of the first reaction to occur, then the first reaction is selected to occur. Otherwise, the probability of the second reaction is added to the probability of the first reaction and if the originally generated random number is less than this cumulative reaction probability, the second reaction will be selected. This continues until a reaction is chosen, or there are no more valid reactions left which will result in no reaction occurring.

The binary reaction can be greatly influenced by geometry (shape) of the boundary, the density of non-reacting obstacles or other crowding agents. This is because the binary reactions depend on the molecular collision between diffusing particles. The crowding agent will hinder molecular diffusion thereby influencing the binary reaction. Figure 4 examines this crowding impact using molecule A of 1.6 nm and molecule B of 2 nm radius and of diffusion coefficients, 32 nm^{2}μs^{-1} and 20 nm^{2}μs^{-1}, respectively. These molecules are typical of small proteins found in cells (Kubota et al. 2008). In the simulation, we impose a periodic boundary condition. At time zero, the same number of A and B molecules are placed randomly (i.e., homogeneous distribution) at a density of 30 molecules in a cubic compartment (100 nm per side). This molecular density corresponds to a concentration of ~50 μM. The reaction probability per collision was set to 1 (the diffusion limited reaction). Upon collision, a new molecule C (2.3 nm radius and diffusion coefficient of 17.4 nm^{2}μs^{-1}) is created. The radius of C molecules was calculated so that the volume of molecule C is equal to the sum of A and B molecules. The diffusion coefficients these molecules obey Stokes-Einstein relation (Sutherland 1905).

First, we examine the binary reaction A+B→C without crowding agent. An average of 500 simulations fits well to the analytic solution derived from the chemical reaction theory, $1/(A_0–C(t))–/1A_0=k*t$, where $C(t)$ is the concentration of C molecules at time t and A_0 is the initial concentration of A and B molecules (Fig. 4a). This provides a numerical validation of the CDS algorithms that we discussed in “Event-Driven Molecular Collision and Chemical Reaction Model”. The resultant reaction rate (*k*) is 1,173.4 μM^{-1} s^{-1}. Next, we place obstacles (2.5 nm spheres) in the space to examine the impact of molecular crowding. The reaction probability and molecular crowding may influence the binary reaction in a highly complicated manner. For example, hindered diffusion prevents the encounter of two molecules traveling a long distance while it increases the collision rate of two molecules confined in a small crowded environment. Figure 4b shows the time course of the product molecules (per 100 nm cubic volume) with different volume occupancy of crowding agents. Here all crowding agents are immobile. As we increase the volume occupancy of crowding agent (see 29 % volume occupancy in blue), the binary reaction slows down. At 42 % volume occupancy of the crowding agent (red), the reaction is stalled and most reactants will never interact. At this volume occupancy (42 %), reactants (A and B) are trapped in subspaces. Those reactants trapped in a small region may react quickly, but the probability of reaction quickly approaches zero as reactants are unable to interact due to hindered diffusion from the restrictive obstacles. Note the impact of diffusing crowding agents is difficult to study using MCell and Smoldyn because these simulators ignore the volume exclusion principle.

First Order Reactions

In this section, we discuss two other types of chemical reactions (isomerization and dissociation), which we call first order reactions. Upon initialization at runtime or whenever there is a state change or new species formed (e.g., due to a binary reaction), CDS searches all valid first order reactions and computes their event times. The time of next first order reaction is calculated from the corresponding reaction rate based upon a Poisson distribution as in the Gillespie algorithm (Gillespie 1976) using the formula:

$$reactionTime = -\log(random)/rate \tag{6}$$

where *random* is a uniformly distributed number between 0 and 1. Like binary association reactions, the dissociation reactions may be one of the following types:

$$C \rightarrow A+B:0.1 \tag{7}$$

$$A.state2 \rightarrow A.state1+B:0.1 \tag{8}$$

$$A.surface1.state2 \rightarrow A.surface1.state1+B:0.5 \tag{9}$$

$$A.state2 \rightarrow A.state1+B.state1:0.1 \tag{10}$$

As with association reactions, CDS attempts to place newly generated molecules (A and B in Eq. 7–Eq. 10) close to the location of C molecules. In the case of a $C \rightarrow A+B$ reaction (Eq. 7), the larger of the products is placed at the former location of C (suppose we assume the larger molecule is A). Then free space for B is searched for by choosing a random point about the former location of C at a distance of A’s radius times 1.1 plus B’s radius. There is a slight offset in the distance to prevent an immediate re-association. CDS searches new and different locations for B molecules as it increases the distance between B and A molecules. This iteration process is similar to the free space search for association reaction as described in “Binary Reactions”. If no free space is found to add B, then the reaction does not take place. The locations and states that existed before the reaction are restored and a new reaction time for the selected reaction is calculated. This algorithm for finding free space will only occur in highly dense environments. If locations are found for both molecules, the new discretized Brownian trajectories are generated for these molecules (the default is that they are opposite of each other). As pointed out in Andrews and Bray (2004), there is no chemical reaction theory that presently exists to determine the distance between newly dissociated molecules even in a non-crowded environment. This is an open area of chemical reaction theory and CDS allows the users to define or assign different values than the defaults described here as demanded by their specific applications.

Surface reactions, as described so far, are inter-molecular meaning that a surface of a molecular species may only interact with another molecule (Fig. 3b). To accommodate interactions between subunits within the same element CDS allows the user to group together surfaces. A group of surfaces may have its own internal state and may have first order reactions that may change this state. Groups allow monitoring of subunits that are assigned to them. An example of this would be a receptor where individual subunits have their own agonist and antagonist reaction schemes. In CDS, “rules” given to a group of these subunits can check their different states and influence the overall state of the protein complex based upon various combinations of subunit states (Fig. 3c). Whenever there is a state change, these rules are checked and if all conditions are met, feedback actions are applied to the element. This system of feedback is described in “Events and Actions”.

Aggregation and Lateral Diffusion

As described above, when a binary reaction occurs the reactants are physically removed from the simulation space and an instance of the specified product is created in a nearby place. In CDS, binding properties are added to elements and membranes such that elements become physically attached to a target and can form aggregated structures (Fig. 5). Two types of binding can occur, an element to a membrane and an element to another element. There is no limit to the size of the chain of bound elements. Analogous to dissociation, unbinding can also be defined and is treated as a distinct event. When molecule A binds to molecule B, B is checked for any unbinding events associated with molecule A. If this unbinding event is defined in the input file, an event time for that unbinding is calculated by using Eq. 6 (with a defined unbinding rate).

When a freely diffusing molecule binds to a membrane, it by default maintains its orientation. If an alignment vector is specified by the user, CDS will rotate the molecule such that the alignment vector points in the same direction as the normal vector of the colliding face (perpendicular to the triangle). A standard convention for meshes, and one used strictly in CDS, is that all normal vectors point away from the center of the mesh. This is used to determine (for a closed compartment) what direction indicates the inside volume and outside volume. This is particularly relevant for channel transport where the direction of dissociation or movement of ions is important. When the mesh is rotated, all vectors associated with the molecule are rotated accordingly. The result of the rotation is that the alignment vector points in the same direction of the triangle’s normal vector. The angle of rotation is specified by three values, the angles of rotation about the x-axis, y-axis and z-axis.

When a freely diffusing molecule binds to a membrane, unless the movable property is switched off, that molecule will continue to diffuse laterally along the membrane (Fig. 5b). Membrane bound diffusing molecules have a random trajectory that lies in the plane described by the triangle that the molecule resides on. Collision detection in this mode is performed on the center of the molecule against the three line segments that make up each side of the triangle. When a molecule hits the edge of a triangle it will calculate a new trajectory that lies in the neighboring triangle and the molecule will realign itself based upon the normal vector of the new triangle. Freely diffusing molecules may become membrane bound from the internal or external volume. If an unbinding property is defined, another parameter may be specified to instruct the unbinding molecule what volume to unbind to. By default the molecule unbinds into the volume it existed in prior to becoming membrane bound, but other options include always unbinding internally, always unbinding externally, unbinding to the volume other than its prior parent, or unbinding randomly to either the internal or external volume. The initial placement of the unbound molecule is based upon the direction of the normal vector (or negative normal if the unbinding is internal) with a magnitude of 1.1 times the radius. There is a slight offset in the distance to prevent an immediate re-association. As with dissociation reactions, free space for the unbinding molecule is checked about this initial point but in a conical fashion about the normal vector. The region of placement is in the shape of a cone to prevent an element from being placed on the wrong side of a membrane from which is was intended to be placed. The maximum angle that an element may be placed is 45 degrees from the normal vector. While not as important in a cubic compartment, it is essential when membranes become more convoluted.

Transport and Permeability

The transport property is an event triggered upon collision of a diffusing molecule to a surface and is used to instantaneously move the colliding molecule from one point in space to another. While generally implemented, the principle application of this is to create pumps and channels to transport ions across impermeable membranes (Fig. 6b). As with binary reactions, a probability between 0 and 1 may be assigned to determine whether the transport event will occur upon collision of a freely diffusing element. In the example of a Ca^{2+} pump, the transport probability combined with the Ca^{2+} concentration in a compartment will translate to a macroscopic rate at which Ca^{2+} will be pumped out of the compartment. A vector is used to describe the placement of the transported molecule relative to the location of the channel (or any structure assigned with the transport property). CDS will attempt to move the molecule to that point in space or if no room exists, then it will try to find room in a conical fashion based around the placement vector. The default angle to constrain distance about the placement vector is 45 degrees. Depending on the non-linearity of the membrane where the channel resides, a wider angle (approaching 90 degrees) may result in placement on the wrong side of the membrane. While the direction of placement is constrained, within a defined number of tries, the distance from the initial placement is incremented by the radius of the transported element. Also as with dissociation reactions, if free space cannot be found within a specified number of tries, then the event does not take place.

By default, all membranes are impermeable and all diffusing molecules are solid. Transparency is a surface property that can selectively eliminate collision detection of specified elements or compartments (Fig. 6a). Since it is a surface property, it can be applied to both diffusing elements as well as compartment membranes. For diffusing molecules, this could be used for certain collisions that are insignificant for the simulation and will decrease simulation time. For example, calcium ions can include a transparency property to itself so all collision detection between Ca^{2+} and Ca^{2+} is ignored. This property works slightly different for membranes. Collision detection is still performed against a permeable (i.e., transparent) membrane, but this is only for changing the molecule’s parent volume and the trajectory is not changed during this collision event. Two connecting volumes that border each other should have a transparent membrane such that the population change between each volume is registered. Where a semi-permeable membrane may have a few specific targets where collision detection is ignored, surfaces that act as a border between two volumes may be only for registering volume change and should not obstruct diffusion of any molecules. The transparency property can have a target that is a wildcard (designated by “*”) such that any molecule will register as a target to the transparent border.

The two options discussed so far for any particular target molecule is that a membrane may be impermeable or permeable to that target. It is also possible to have absorbing membranes. Any compartment, whether designed as a boundary or cellular structure, can have reactions assigned to it in the exact same way that reactions are defined for elements. Defining an association reaction to the surface of a compartment will cause it to absorb the target upon collision. Again, a special care is necessary for (partially) absorbing boundary and we follow the strategy described in Andrews and Bray (2004) to calculate reaction probability. If no dissociation reaction is defined, then the compartment will indefinitely hold the element. If there is a dissociation reaction, when the reaction event is processed, the element will be placed in space about where the association took place.

Events and Actions

Everything that occurs in the simulation environment is termed an event. This includes all previously mentioned properties such as collision, reaction, binding, unbinding, and transport. While events such as reactions have an implicit behavior (the creation of products), CDS allows users to apply additional feedback when that event occurs. This feedback can be added onto any event and is called an "action". Actions can change properties, simulation variables, add or delete elements in a volume or surface, and can terminate the simulation. An action can be applied to only the molecules involved in the particular event or can have global effects. If a reaction occurs of the form $A+B \rightarrow C$, a local action would only (potentially) affect those elements involved with the reaction. An example of this would be *C. Set.DiffusionCE.1.6*. The default value for C may have a different diffusion coefficient. Using this action would change the diffusion coefficient only for this new instance of C. A global action may change the properties of a population of elements such as *All.Channel.Set.Current-State.Open*. These actions are defined by a code made of several keywords (separated by periods) that CDS uses to traverse through different components of the simulator.

CDS allows users to create their own events that can trigger a series of actions and provides a variety of ways on how and when these events are performed. Triggering vesicle release, channel influx, or bath application of a drug are examples of what can be handled with user events. These events can be singular or recurring and can be triggered based on an explicit time, by a rate, or conditionally. They may also be activated by an action created from a different event. Events triggered by an explicit time use the specified value as the time until the event occurs. That event time is decreased as the simulation environment advances. Events triggered by a rate have their event time calculated by a Poisson distribution identical to that of first order reactions. Conditional events are checked if their Boolean function returns true or false at the start of every time step. If true, the actions associated with that event are performed, and nothing occurs if false. One example of this could be a simulation that has an infinite run time and instead terminates with a conditional statement. For singular events, once the event is processed, no new event time is calculated. For recurring events, when the event is processed, a new event time is calculated. Singular events once completed are still loaded in memory and are simply dormant. They can always be referenced and activated by an action.

When reactions involving state change occur, the internal state is implicitly changed when the reaction is processed. In the previous section, binding events were described. During binding the only processing performed is determining the behavior of the newly aggregated complex. If the user wants to perform binding reactions, actions can be added onto the binding event that can cause state changes.

Moving Compartments

A third type of structure included with CDS is a moving compartment. These compartments are nearly identical to the capabilities of elements with the exception that they have a single populatable internal volume. With respect to static compartments, they allow a similar behavior of containing membrane bound and freely diffusing molecules. Diffusion, binding and other properties available to elements apply to the moving compartments in exactly the same fashion. Diffusing molecules within a moving compartment only perform collision detection on each other and the parent moving compartment. No outside entities are checked for collisions in order to boost computation time; especially since moving compartments are typically computationally expensive. As one example, any fast diffusing ions or transmitter in a small confined volume results in very frequent collisions and therefore frequent collision detection.

Bath Application and Degradation

During a simulation, elements can be spontaneously added uniformly into any volume or onto any surface. This is accomplished via an action so bath application can be invoked by any event. Input arguments for the bath application are the name of the volume or surface, the type of element to be added, and the number of instances to be created. All instances of the element are created instantaneously when the action is processed. An alternative to a singular event triggering bath application is to use a recurring event to spontaneously add elements one at a time. While similar behavior, the difference between this method and using a first order dissociation reaction on a channel would be that this method gives a uniform spatial distribution of the influx.

Any element can have a degradation property added to it which will trigger an event that removes the element from the world. This is identical to a first order degradation reaction except that there is more flexibility with this property. The property can be initialized such that it is active or inactive. If active, an event time will be calculated for the degradation event and if inactive, a degradation event will never occur. Suppose there is a bath application of a buffer that has a fast clearance rate. Rather than attempting to use other methods available in CDS to create the desired rate of clearance, the degradation rate can be explicitly stated in this property. Upon creation of the buffer in the world, an event time will be calculated for the degradation event.

Variables and Integration

Most instances where CDS takes input as a parameter value can be defined as a function instead of a constant. Functions can be comprised of almost any mathematical expression and include basic functions provided in the Math class of the Java language (e.g. sqrt, pow, min, max, floor, exp, rand, log, etc…). Variables may also be referenced from within functions. A CDS variable is an object that stores a double precision value. Variables are passed to different parts of CDS with a reference so that many components of CDS may access the value stored by the variable. Like actions, a code is used for the referencing of variables (e.g. *Volume.Spine.Population.Ca*, *Time*, or *World.TimeStep*). Common properties that may take a function as input include the end time, time step, diffusion coefficient, reaction radius, and reaction rates. One example of this would be to have a channel that injects Ca^{2+} into an intracellular compartment when extracellular space is not being simulated. The function describing the influx rate can be dependent upon the intracellular concentration of Ca^{2+}.

In addition to functions as expressions, functions may also return values based upon a conditional statement. The difference between these two is automatically parsed by the Function object based upon the syntax used.

$$function\_name=x*y+z \tag{11}$$

$$function\_name=(a==1\&\&b>0)?x+y:z \tag{12}$$

If the format is in the form of Eq. 11, when the function is solved, this expression is calculated. If formatted as Eq. 12, when the function is solved, first the conditional statement is calculated. If the returned value is true, then the solution to the true expression is returned, else the solution to the false expression is returned. The syntax of the question mark and colon is standard shorthand for if/then/ else statements in C and Java programming languages. The use two different identifiers before the true and false functions allow for nested conditional functions.

CDS includes an integrator for solving a system of ordinary differential equations (ODE). ODE variables that are being integrated can also be referenced from various components in CDS. Likewise, parameters written in an ODE may reference CDS variables. The default integration method is a fourth order Runge-Kutta method using a fixed time step that is by default the time step used for creating Brownian motion. If that time step is not small enough for accurate integration, then a smaller time step can be specified just for use by the integrator. ODEs are integrated as the simulator progresses. At the end of each time step, the system is integrated up to the current simulation time.

One possible application of using a system of ODEs would be to model membrane potential with Hodgkin-Huxley equations (Bower and Beeman 1998) and use the potential to trigger the opening and closing of voltage-gated channels (Fig. 7). Another example would be simplifying fast diffusing ions to a single pool described by an ODE. Instead of binary reactions with physically simulated Ca^{2+}, Ca^{2+} binding could be simulated by a first order reaction that references the Ca^{2+} variable in the reaction rate function. A Ca^{2+} target protein can have a Markov type reaction scheme described by the type of reaction in Eq. 13 as opposed to the binary form in Eq. 14. In the following equations, k_1 represents the reaction rate while p_1 describes the binary reaction probability.

$$CaTarget.0 \rightarrow CaTarget.1{:}k_1{*}ODE.Ca \tag{13}$$

$$CaTarget.0+Ca \rightarrow CaTarget.1{:}p_1 \tag{14}$$

Analysis and Output

There is a plethora of possible quantitative output that could be extracted from a simulation, so to allow maximum flexibility and options to the user, analysis and output code is not directly implemented. Every component in CDS allows modules to be added to it, where a module is any object that extends an abstract Module object. Whenever an event occurs, the components associated with that event will send its reference and an event ID to all modules stored within that component. The module has the ability to call any method and pull all data stored within that component and the event ID tells the module what type of event had occurred. The ID might indicate a collision, association reaction, transport, binding and so forth. A module might have a very specific output and would only be active for a small number of events. Users can write custom modules that can be loaded into CDS at runtime to dictate which type of information will be recorded.

There are several modules included in CDS that offer standard types of output. The VizOutput module writes the visualization file that can be read by the Visualizer. It is added to any compartments, elements, or moving compartments that the user wants to visualize. Position data can be recorded to provide continuous animation or for longer simulations the environment is saved at a specified interval for snapshot rendering. The PopulationOutput module can be added to any volume or surface. Species and their states are grouped together by their parent compartment and written as a time series. The output can be in the form of the number of species or as concentration/density. SingleTracking records the position and distance travelled by an individual element. If added to a group of elements, a separate file will be written for each element instance. This module takes into account the periodic boundary when determining location. ReactionOutput will record information when reactions occur. This can track a particular reaction or several reactions within a single output file. When a reaction occurs, the reaction, the species IDs and the location of the reaction in space is written to the output file. There's no limit to the number and types of modules loaded in a simulation and multiple ReactionOutput modules can be created where each one looks for a specific reaction and each module has its own output file. A single ReactionOutput can also be created which looks for multiple reactions and saves all the data to a single file.

Another module included with CDS has the ability to generate a simulation file based upon the state of the environment upon completion of the simulation. All changes in properties, deleted elements and added elements are included in this file. This generated simulation file can be reloaded as input into the Initializer and further modified or run from the simulator. This module is useful for users who want to run a simulation until an equilibrium is reached in the system and use that environment as a beginning point for further simulations. This module is added to the world object and is only active after the simulation is complete. Since the module is added to the world, it has the ability to reference all instances of compartments and elements and all current property values stored within those instances.

Supplementary Tools

CDS Initializer

Constructing the simulation environment is a labor intensive task to perform manually even for very simple simulations. The CDS Initializer is a stand-alone graphical tool for creating and populating the simulation environment. The initialize interface has three two-dimensional views (front, side and top) for viewing the world and a floating frame used to manage and assign properties to compartments and elements (Fig. 8). When the Initializer first loads, an empty world is created which is an infinite void.

When compartments are first created, a shape must be set which is defined by a mesh. Once the shape is created, the compartment is placed at the origin of the world and is given the properties of a unique ID and a location. Each compartment is unique and a single instance exists in the simulation. When creating an element, only a base definition of the element is created which initially defines a shape and no other properties. Using this base definition, many instances of that element can then be added to the world in the form of groups. For example, the user may select a compartment and uniformly distribute 50 instances of element X into the volume of that compartment with a group name *xgrp*. The definition of element X can be expanded which will show all groups created with element X. Each group can be expanded to view the individual instances that were created. When each instance is created, it initially has the properties of a unique ID, a location, and in some cases an initial rotation. The point of the hierarchy is that properties may be defined in the base definition, and all instances will inherit those properties. However, it also allows properties to be changed on a group by group or instance by instance basis. Any property defined in a group will override the same property defined in the base definition, and the same is true for properties added to a single instance. In practice, a group of receptors may be initialized with some internal state while another group is initialized in a different internal state, but all other properties that define that receptor are identical.

When new compartments and elements are created, a mesh must be assigned to it (or also a sphere for elements). The two options to the user are to create a new mesh or import a mesh. New meshes can be created from a list of several prefabricated designs including primitive shapes such as spheres, cuboids, cylinders, pyramids and so forth. There are some more specific prefabrications such as those that resemble dendritic spines. All these prefabrications are created based on user specified parameters such as a radius for a sphere or length/ width/height for a cuboid. Most of them also take a resolution parameter which indicates how many triangles comprise the mesh. For example, a sphere with a resolution of 1 is diamond shaped. Every increase in resolution doubles the number of lines of latitude and longitude. Once the mesh is created, the Initializer provides tools for manipulating the shape of the mesh. The goal of these tools is to allow the user to create fairly complex structures without having to rely on third party 3D modeling software which can be very complicated to use and are generally expensive. The entire mesh may be scaled, translated and rotated. Individual or groups of vertices can be selected and manipulated also (Fig. 9). Whenever the mesh is transformed surface area and volume are recalculated.

The file saved by the Initializer is a simulation file (.cds) that can be read by the simulator. Simulation files are based on a XML format. The hierarchal format of XML works well with the tree data structure used in CDS to eliminate redundancy of the element definitions. XML also facilitates easy file creation, parsing and can be loaded into a tree structure for quick searching and reference. This also allows for some nodes to be pulled from the tree and saved into separate files to store in a library. For example, a large protein reaction scheme may be created in a simulation. This scheme can be saved separately into a library that can be loaded into future simulations. This is also true for element and compartment definitions.

CDS Visualizer

The CDS Visualizer is a stand-alone tool for rendering and animating CDS output created from the VizOutput module (Fig. 10). Compartment and elements are displayed in a tree on the left side of the window to control color and rendering properties. A bottom panel has animation playback controls and the main window shows the rendered view of the simulation. Playback is controlled through VCR style buttons. Animation can be paused, advanced frame by frame, played in real time, or in a fast forward (or slow motion) mode. The same controls also exist for backwards playback. If the time unit in the simulation is in microseconds, then when animating in normal “Play” mode, for every second of real time, one microsecond of simulation time will elapse. Fast forward and fast backwards work based upon a multiplier such that a multiplier of 1 will equate to the play mode. Slow motion can be viewed by setting this multiplier between 0 and 1 and fast forward if greater than 1. The user can also jump instantaneously to different time points in the simulation.

The view in the main window is what is rendered based upon the orientation of the camera. The camera can be manipulated using a three button mouse. The left mouse button rotates the view, the wheel zooms in and out, and the right button pans the camera. If the user desires smooth motion of the camera for recording purposes, the camera can be automated by controls that are driven by the animation playback. The Visualizer can take screenshots of the rendered view and create QuickTime™ movies. A record button, once clicked, will output the rendered view to a movie file while playback is occurring. The playback can be paused during recording so lighting, rendering properties, and the camera may be altered and the playback may resume under the new changes. The user can manually jump to different time points as well. This allows movie creation as several segments using this software without resorting to third party movie editors for stitching together movies. Screenshots can be taken in two different ways and saved in a variety of formats. Simple screenshots can be taken at the resolution based on the current size of the view. High resolution screenshots can also be taken with a resolution specified by the user. The maximum size is dependent on the graphics card, but most cards can handle upwards of 4000 by 4000 pixels; plenty for publishing quality.

The data read by the Visualizer is a series of time stamps where each time stamp is followed by the molecules that were updated at that time. An update is registered whenever there is a change in location, rotation, or internal state. The Visualizer interpolates the time-stamped position data to provide continuous movement. Elements can be color coded by internal state to visually identify when reactions take place. Rendering properties are assigned to elements in the same manner of hierarchy as in the Initializer. Individual molecules can be colored uniquely, or the base molecule definition may be given rendering properties that apply to all instances of that definition. The Visualizer can also load simulation files to see the snapshot of the simulation environment at initialization. The 2D views in the Initializer can become very crowded in dense environments so confirming the environment creation can be done in a 3D view.

Summary

CDS has been constructed as a tool with a wide variety of uses, from theoretical simulations (a zero compartment void with a periodic boundary) to physiological representations of sub-cellular compartments. CDS will continue to undergo development as new research directions may call for increased functionality. Potential modifications may include simulating active transport, adding depth to membranes, and the ability to import reactions from Systems Biology Markup Language (SBML) models. The source code will be available to researchers with an open source mindset. Incorporating a variety of researchers in different areas of molecular modeling will help give insight in improvements to CDS as well as an expanded development team such that the simulator can continue to become more flexible and efficient in design. The initialization and visualization tools may also be improved based upon the feedback from users with different levels of experience in 3D environments. Every effort will be made to insure that the software remains intuitive for advanced and novice users so that the CDS will find utility for a broad base of users. It is hopeful that the generality and power of CDS will attract numerous research projects in various fields of molecular biology and neurophysiology.

Aside from the difficulty in creating 3D simulation environments, the greatest problem with particle simulators of this nature is the large computation time when presented with complex environments with large molecule numbers. Typical workstations (containing CPUs of 2 to 3 Ghz clock speed and 2 to 4 cores) limit the range of simulation time of complex systems to seconds and simple simulations to a timescale of minutes. As processors become more powerful and the number of cores increases which will allow for greater multi-threading, this will become less of an issue. More powerful computers in combination with more efficient optimization techniques will gradually allow complex systems to be simulated to timescale of minutes with a computation time measured in days. Greater computational power in the future also leaves open the possibility of expanding the scale of these simulations into a cellular rather than sub-cellular scale.

Acknowledgments

This work was supported by U.S. National Institutes of Health grant NS038310. We would like to thank Pete Swulius and Dave Howard for their invaluable help on developing an early version of the visualization tool.

References

Andrews SS, Bray D. Stochastic simulation of chemical reactions with spatial resolution and single molecule detail. Physical Biology 2004;1:137–151. [PubMed: 16204833]

Boulianne L, Al Assaad S, Dumontier M, Gross WJ. GridCell: a stochastic particle-based biological system simulator. BMC Systems Biology 2008;2:66. [PubMed: 18651956]

Bower, JM.; Beeman, D., editors. The book of GENESIS: Exploring Realistic Neural Models with the General Purpose Neural Simulation System. second edition. New York: Springer-Verlag; 1998.

Bredt DS, Nicoll RA. AMPA receptor trafficking at excitatory synapses. Neuron 2003;40(2):361–379. [PubMed: 14556714]

Carslaw, HS.; Jaeger, JC. Conduction of heat in solids. 2nd ed.. New York: Oxford University Press; 1986.

Dix JA, Verkman AS. Crowding effects on diffusion in solutions and cells. Annual Review Biophysics 2008;37:247–263.

Donev A, Torquato S, Stillinger FH. Neighbor list collision-driven molecular dynamics simulation for non-spherical hard particles. I. Algorithmic details. Journal of Computational Physics 2005;202:737–764.

Erban R, Chapman SJ. Stochastic modelling of reaction-diffusion processes: algorithms for bimolecular reactions. Physical Biology 2009;6(4):46001.

Fauerby, K. Improved collision detection and response. 2003. http://www.peroxide.dk/.

Fiala JC, Harris KM. Extending unbiased stereology of brain ultrastructure to three-dimensional volumes. Journal of the American Medical Informatics Association 2001;8(1):1–16. [PubMed: 11141509]

Gillespie DT. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. Journal of Computational Physics 1976;22:403–434.

Groc L, Choquet D. AMPA and NMDA glutamate receptor trafficking: multiple roads for reaching and leaving the synapse. Cell and Tissue Research 2006;326(2):423–438. [PubMed: 16847641]

Hattne J, Fange D, Elf J. Stochastic reaction-diffusion simulation with MesoRD. Bioinformatics 2005;21 (12):2923–2924. [PubMed: 15817692]

Hugel S, Abegg M, Paola V, Caroni P, Gahwiler BH, McKinney RA. Dendritic spine morphology determines membrane-associated protein exchange between dendritic shafts and spine heads. Cerebral Cortex 2008;19(3):697–702. [PubMed: 18653666]

Hwang C-O, Given JA, Mascagni M. The simulation-tabulation method for classical diffusion Monte Carlo. Journal of Computational Physics 2001;174(2):925–946.

Kerr RA, Bartol TM, Kaminsky B, Dittrich M, Chang J-CJ, Baden SB, et al. Fast Monte Carlo simulation methods for biological reaction-diffusion systems in solution and on surfaces. SIAM Journal on Scientific Computing 2008;30(6):3126–3149. [PubMed: 20151023]

Kim H, Shin KJ. Exact solution of the reversible diffusion-influenced reaction for an isolated pair in three dimensions. Physical Review Letters 1999;82(7):1578–1581.

Klein G. Mean first-passage times of Brownian motion and related problems. Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences 1952;211(1106):431–443.

Kubota Y, Putkey JA, Shouval HZ, Waxham MN. IQ-motif proteins influence intracellular free Ca^{2+} in hippocampal neurons through their interactions with Calmodulin. Journal of Neurophysiology 2008;99:264–276. [PubMed: 17959737]

Moraru II, Schaff JC, Slepchenko BM, Blinov ML, Morgan F, Lakshminarayana A, et al. Virtual cell modelling and simulation software environment. IET Systems Biology 2008;2(5):352–362. [PubMed: 19045830]

Muller ME. Some continuous Monte Carlo methods for the Dirichlet problem. Annals of Mathematical Statistics 1956;27(3):569–589.

Opplestrup T, Bulatov VV, Gilmer GH, Kalos MH, Sadigh B. First-passage Monte Carlo algorithm: diffusion without all the hops. Physical Review Letters 2006;97(23):230602. [PubMed: 17280187]

Redner, S. A guide to first passage processes. New York: Cambridge University Press; 2001.

Ridgway D, Broderick G, Lopez-Campistrous A, Ru'aini M, Winter P, Hamilton M, et al. Coarse-grained molecular simulation of diffusion and reaction kinetics in a crowded virtual cytoplasm. Biophysical Journal 2008;94(10):3748–3759. [PubMed: 18234819]

Santamaria F, Wils S, De Schutter E, Augustine GJ. Anomalous diffusion in Purkinje cell dendrites caused by spines. Neuron 2006;52(4):635–648. [PubMed: 17114048]

Sigurgeirsson H, Stuart A, Wan W-L. Algorithms for particle-field simulations with collisions. Journal of Computational Physics 2001;172:766–807.

Sutherland W. A dynamical theory of diffusion for non-electrolytes and the molecular mass of albumin. Philosophical Magazine 1905;9:781–785.

Stiles JR, Van Helden D, Bartol TM Jr, Salpeter EE, Salpeter MM. Miniature endplate current rise times less than 100 microseconds from improved dual recordings can be modeled with passive acetylcholine diffusion from a synaptic vesicle. Proceedings of the National Academy of Sciences of the United States of America 1996;93(12):5747–5752. [PubMed: 8650164]

Wils S, De Schutter E. STEPS: modeling and simulating complex reaction-diffusion systems with Phyton. Frontiers in Neuroinformatics 2009;3(15):1–8. [PubMed: 19198661]

van Zon JS, ten Wolde PR. Green's function reaction dynamics: a particle-based approach for simulating biochemical networks in time and space. Journal of Chemical Physics 2005;123(23):234910–234916. [PubMed: 16392952]

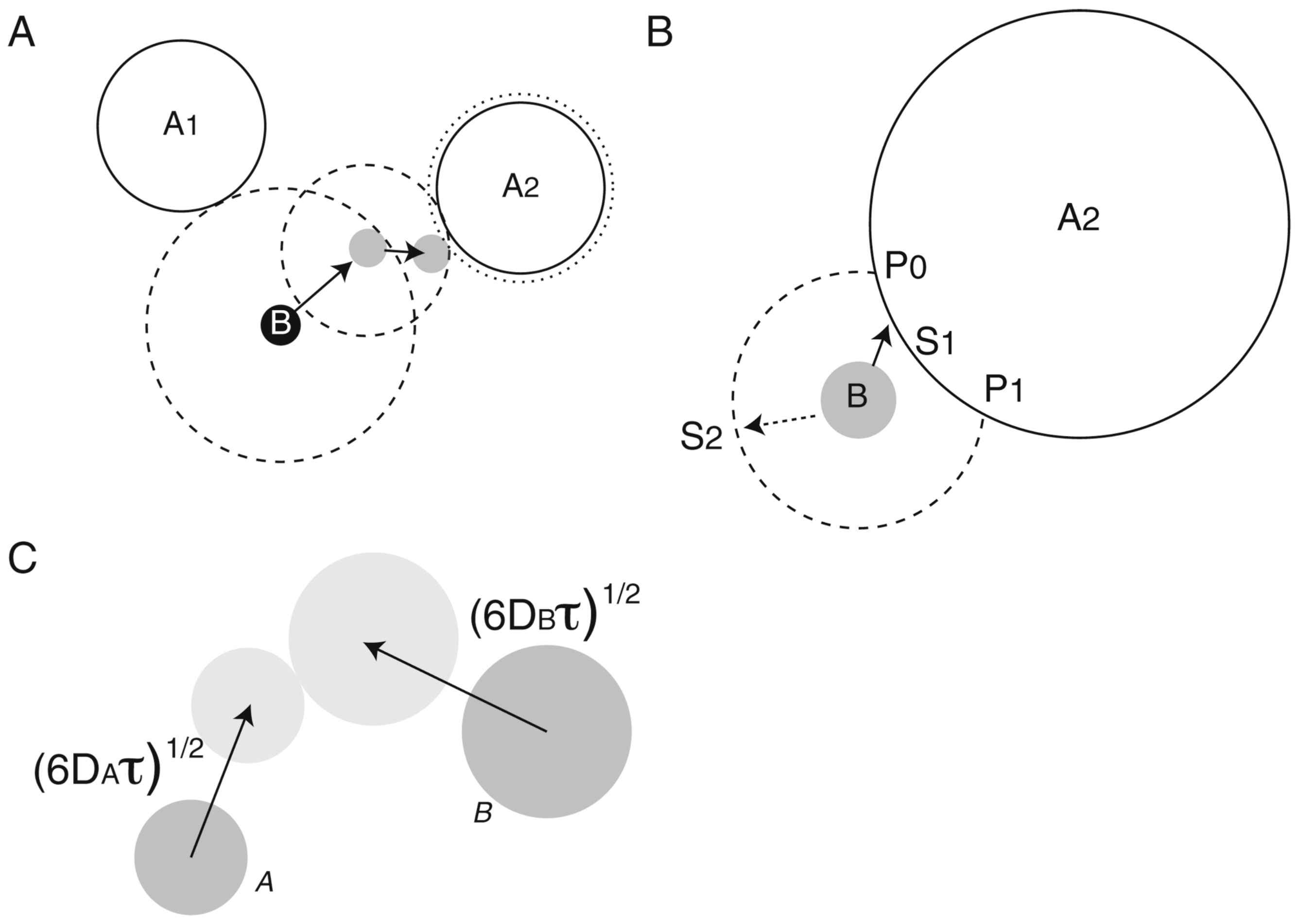


Fig. 1.

Molecular collision detection schemes. **a** A two-dimensional schematic representation of a Brownian trajectory of the B molecule using the WOS (walker-on-spheres) method. A_1 and A_2 are static molecules. The B molecules move within a series of “protective” first spheres (*dashed circles*). The thin-shell (called ε-shell in Hwang et al. 2001) of the A_2 molecule is indicated by the *dotted circle*. **b** In the first passage algorithm, we draw a sphere (*dashed circle*) intersecting A_2 (target molecule) and calculate the first passage probability and first passage time. In this two-dimensional schematic representation, the circle intersects A_2 at P_0 and P_1. S_1 is the sub-region of the surface of A_2 bounded by P_0 and P_1. S_2 is a sub-region of the sphere around B (*dashed circle*). We calculate the first passage probability and first passage time of the B molecule to reach the region of S_1 and S_2. This way we can sample (select) the mean hitting time to A_2 (from the current location of the B molecule). **c** Collision detection scheme for a pair of molecules (A and B). In this panel, the A molecular has moved from its original location by the distance $\sqrt{6D_A\tau}$ and the B molecule by the distance $\sqrt{6D_B\tau}$ along their trajectories and they collide at time τ. If we cannot find such $\tau(0 < \tau < dt)$, the outcome is no collision

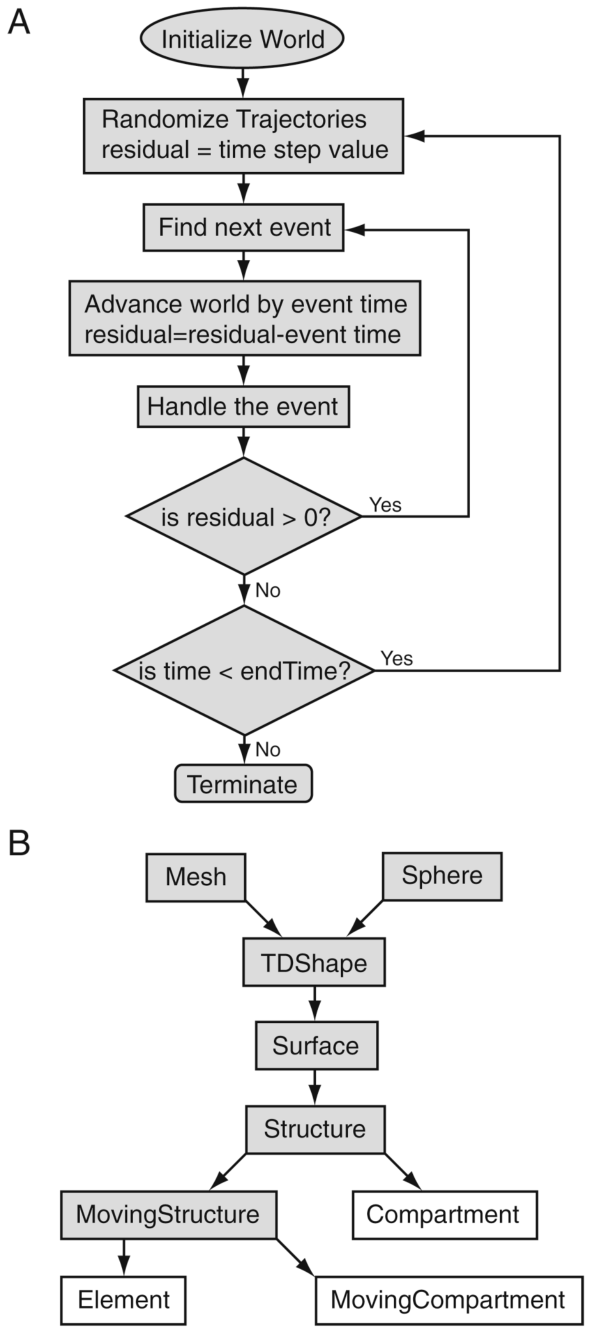


Fig. 2.

Flow chart of the basic organization of CDS. **a** Within the main loop of CDS, which continues while the simulation time is less than the termination time, a sub-loop exists which iterates the time step. Prior to the time step loop, trajectories of diffusing molecules are randomized. A residual value is set to the size of the time step. Within the loop, CDS searches for the next possible event with an event time less than the residual value. If an event is found, then the simulation environment is advanced to the time that the event will occur and the residual is subtracted by that time. If no event is found, the simulator is advanced by the residual time and executes the next time step. Whenever the simulation environment is advanced, all currently stored event times are decreased by the amount of time that was advanced. **b** The hierarchy of objects that make up all physical entities within CDS. Of the three final objects managed by the world (compartments, elements, and moving compartments), much of the code is shared in the form of parent objects

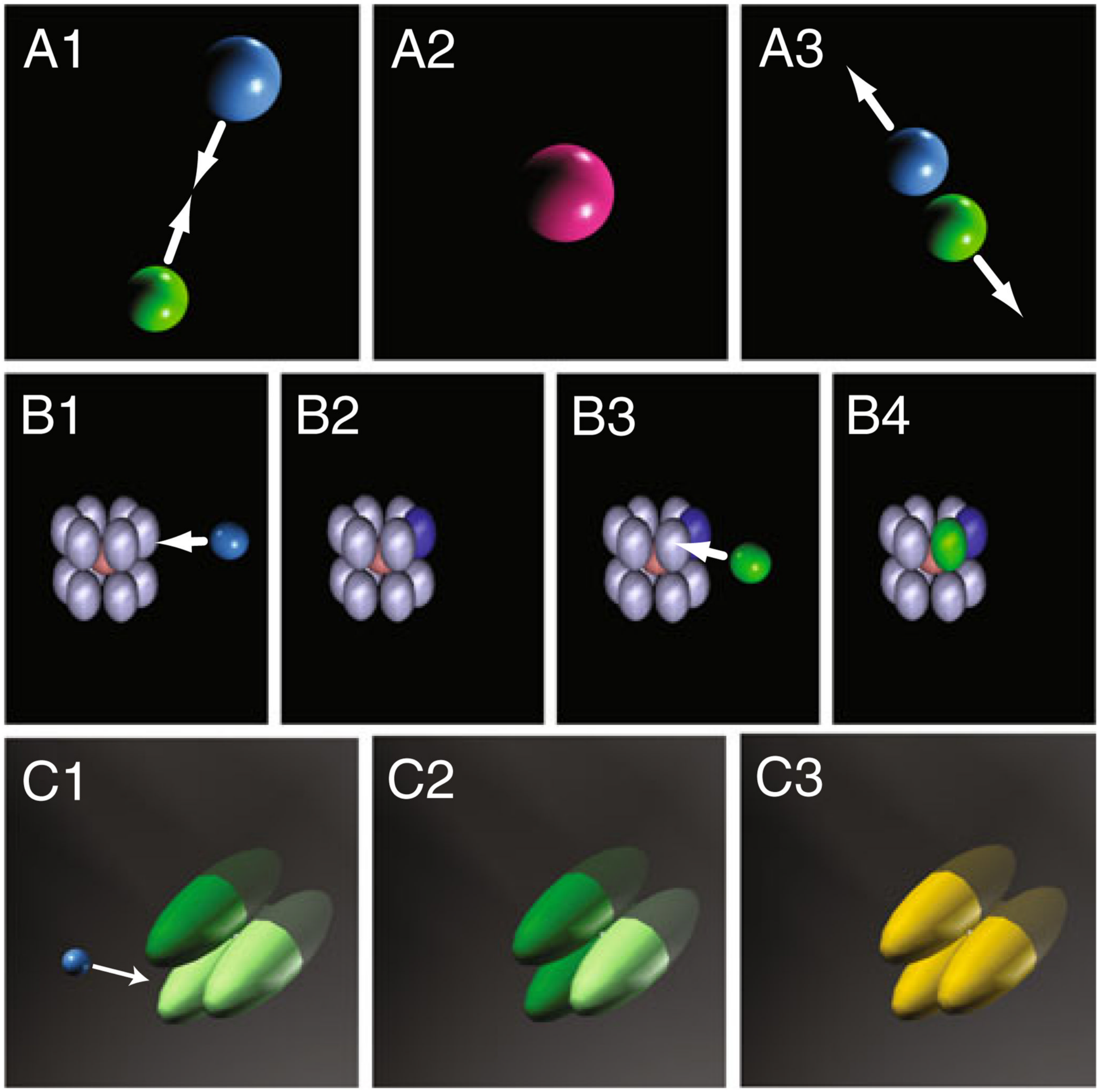


Fig. 3.

Binary and surface reactions. *A1*. Two colliding molecules A and B have detected a collision with an association reaction. *A2*. Upon collision, the product molecule C is created at the midpoint of the colliding locations of A and B. All possible reactions for C are calculated and their event times stored. *A3*. After some time, a dissociation reaction is found to be the next occurring reaction and the event is processed. The same instances of X and Y that were involved in the association are placed adjacently with opposing trajectories. *B*. A protein was created with 12 subunits each with a default initial state (indicated by *gray*). Each subunit has an association reaction for molecules A and B (Eq. 4). As A and B react, the colliding subunits change state as indicated by the color change. *C*. A membrane bound receptor (R) is created with three unique subunits that each has an irreversible association reaction with an agonist. *C1*. After a period of time, subunit R1 is in a bound state and the other two are unbound. A freely diffusing agonist, X (*pink*), has a collision detected with subunit R2 of the receptor and has a valid association reaction. *C2*. The reaction has been processed and subunits R1 and R2 are now in a bound state. *C3*. After time has elapsed, the R3 subunit has also reached a bound state. Upon all subunits reaching a bound state, there is a change in the activation state of the entire receptor (indicated as *orange*)

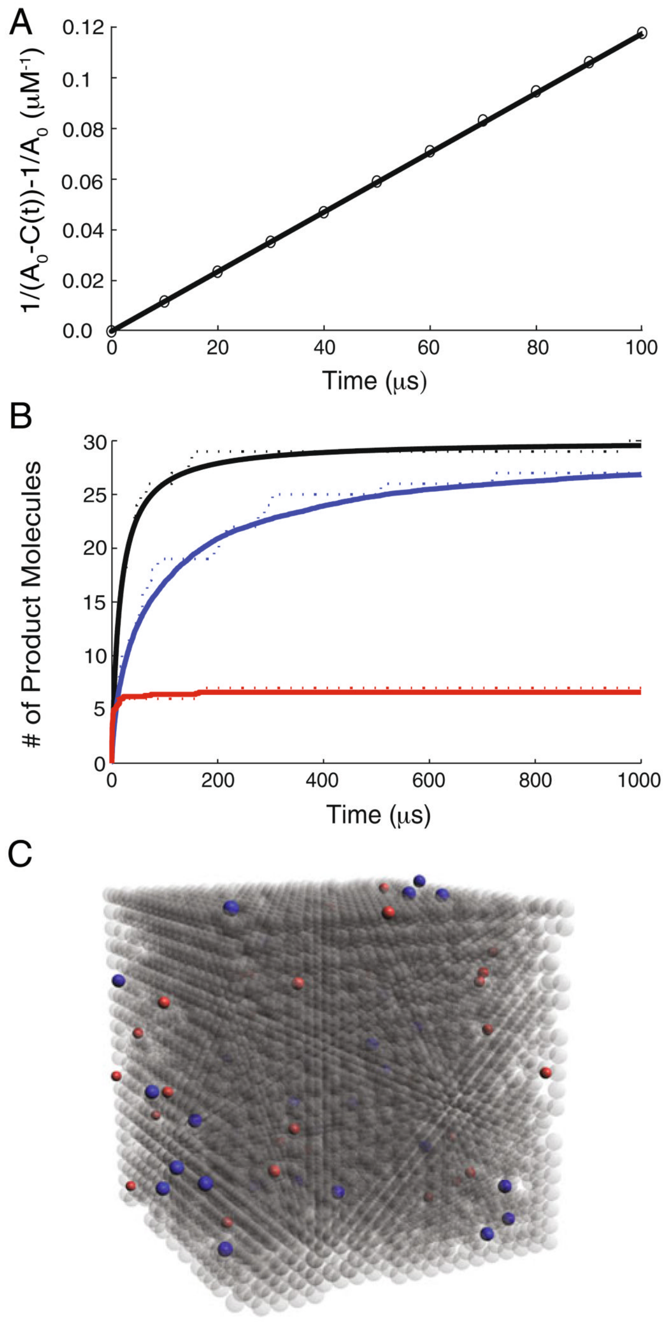


Fig. 4.

Chemical reaction and molecular crowding. The time evolution of chemical reaction between two molecular species, A and B, is shown with or without crowding agents. A and B react upon collision to form a complex, C with a probability of 1 (i.e., the diffusion-limited reaction). Thirty molecules of A and B are placed in a small cubic volume (100 nm per side) with or without immobile crowding agent. The simulation was performed under a periodic boundary condition. The radii of A, B, and C molecules are 1.6, 2, 2.3 nm, respectively. The diffusion coefficients of A, B, and C molecules are 32, 20, and 17.4 nm^2/μs respectively. a An average of 500 simulation runs is compared to the analytic solution derived from the chemical reaction theory. Here we plot *$1/(A_0$–$C(t))$–$1/A_0$* (*empty circles*) where *C(t)* is the (averaged) concentration of C molecule at time (t) from simulation, A_0 is the initial concentration of A and B molecules. Thirty molecules of molecules in a cube of 100 nm per side correspond to a concentration of ~50 μM. The solid *straight line* represents an analytic solution. The inclination of this line is a chemical reaction rate of 1,173.4 $\mu M^{-1}\ s^{-1}$ as predicted for a diffusion-limited reaction. The simulation and theory are in a good agreement. The total time duration of the plot is 100 μs because the chemical reaction theory and the analytic solution are only valid for the initial phase of the reaction process. **b** The time evolution of the number of C molecules (per 100 nm cube) is shown. The immobile crowding agents are spheres of radius 2.5 nm. The volume occupancy of crowding agents are 0 % (*black lines*), 29 % (*blue lines*), and 42 % (*red lines*). The *solid lines* are average of 500 simulation runs and the *dotted lines* represent single simulation runs for each case. **c** A rendering of the crowded environment used in Fig. 4b. The volume occupancy of crowding agent (2.5 nm spheres, transparent gray) is 42%. A (*red*) and B (*blue*) are also shown

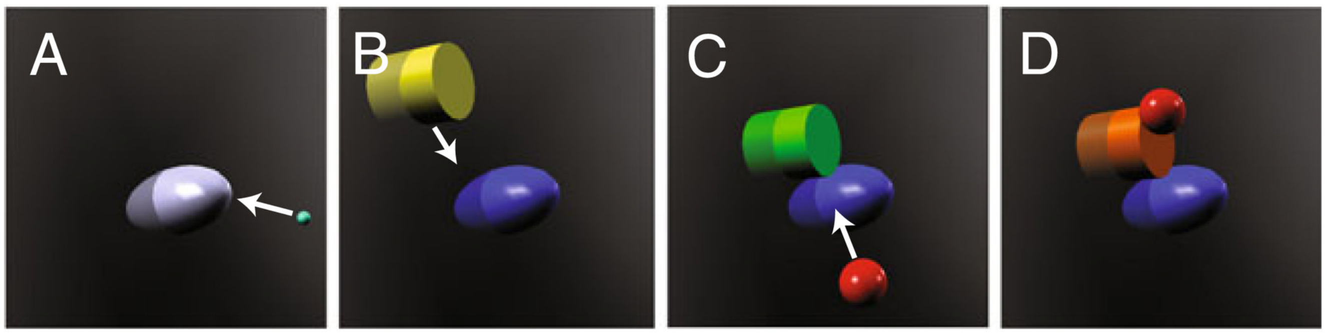


Fig. 5.
Aggregation and trafficking simulation. **a** Scaffolding protein S1 (*light blue*) is static and membrane bound. **b** Upon a freely diffusing agonist (cyan) reacting with S1, S1 switches state (*blue*) such that it may anchor the laterally diffusing receptor R1 (*yellow*). **c** Receptor R1 has bound to S1 and upon binding, R1 switches state. R1 in its new state now has the ability to anchor a freely diffusing agonist (*red*). **d** The final aggregated complex anchored about the immobile scaffolding protein

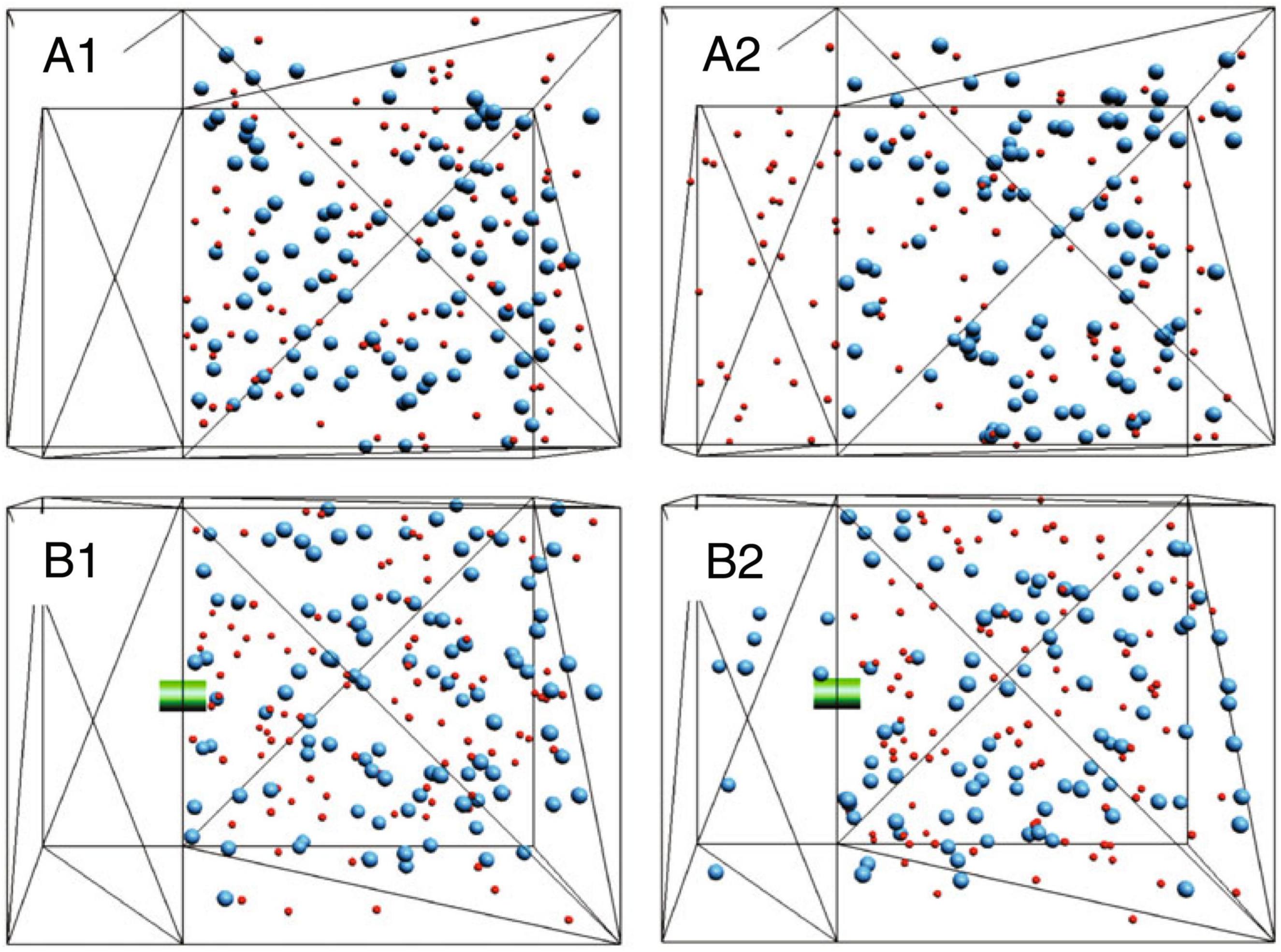


Fig. 6.

Membrane permeability and transport simulations. *A1*. Two volumes are separated by a semi-permeable membrane. This membrane is permeable to X molecules (*red*) and impermeable to Y molecules (*blue*). At initialization, all molecules are distributed in the right compartment. *A2*. After time has elapsed, Xs have begun to diffuse to the left compartment while Ys remain contained within the right compartment. After enough time, X molecules will be uniformly distributed over both volumes and Y molecules will remain uniformly distributed in the right volume. *B1*. Two volumes are separated by an impermeable membrane. A channel is located on the membrane that transports Y molecules from the right compartment to the left compartment upon collision. *B2*. After time has elapsed, Ys have been transported into the left compartment while Xs remain contained in the right compartment

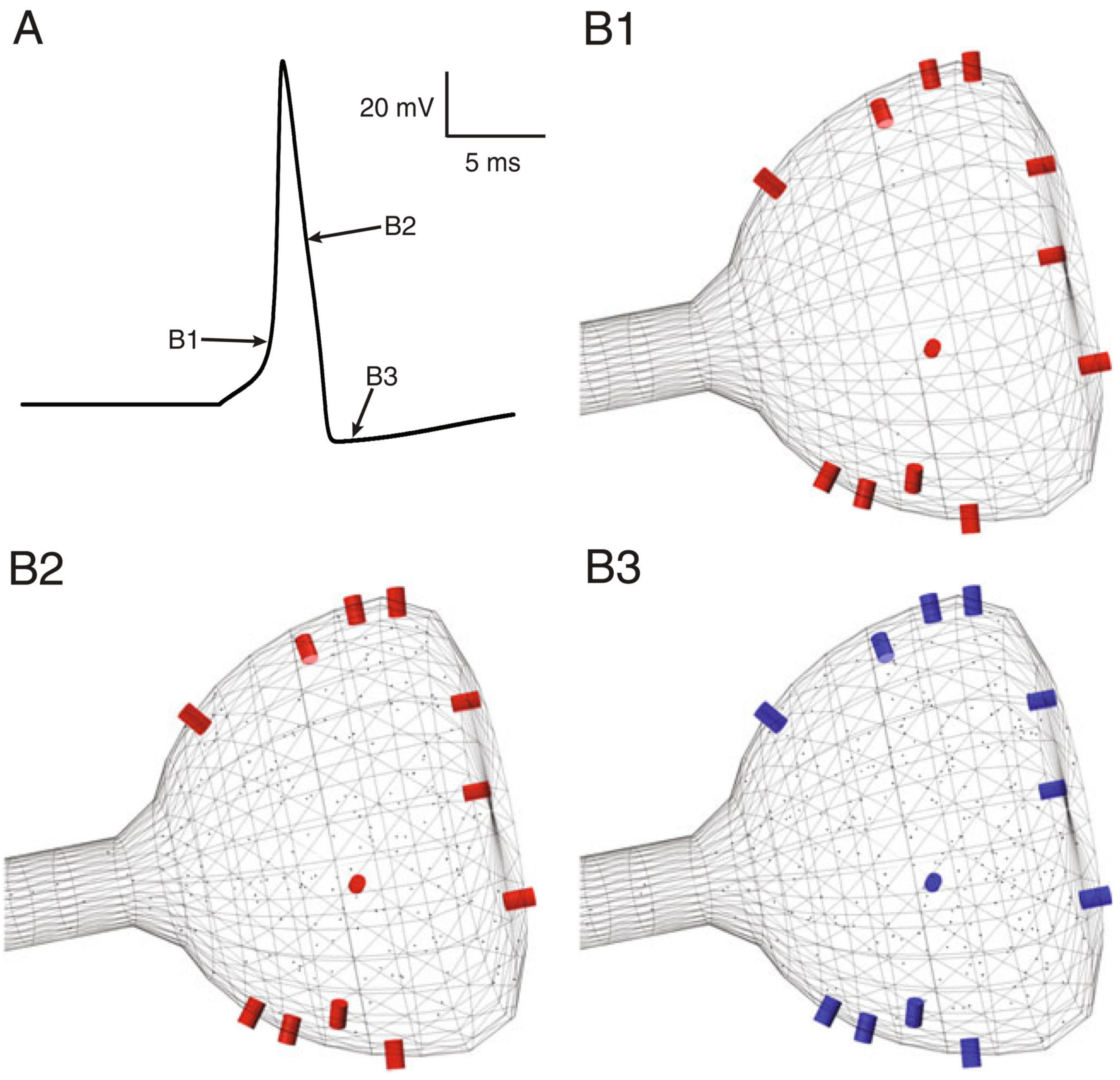
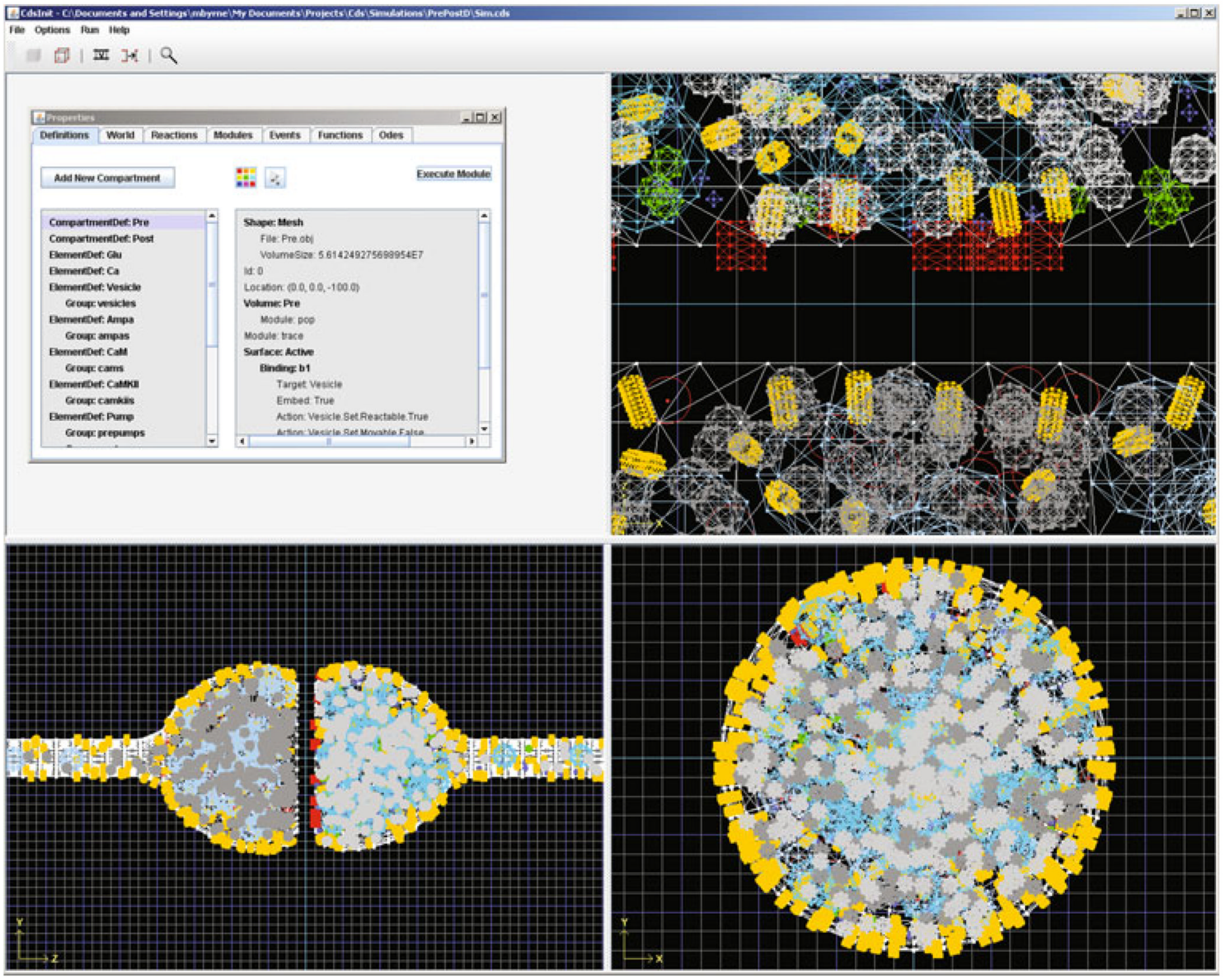


Fig. 7.

Triggering voltage gated channels with an action potential. *A*. The action potential is elicited with a 3 ms, 5 nA current. The voltage gated channels are switched to an open state when there is a greater than 20 mV depolarization. In an open state, each channel injects Ca^{2+} ions at a rate of 0.01 μs^{-1}. The *arrows* indicate the time points and potential at which the renderings in panel B were taken. In this simulation, we assume a uniform potential, i.e., the spine is treated as a single compartment. *B*. Renderings of the system are shown at 2.5, 4.5 and 6 ms (in panel *B1*, *B2* and *B3* respectively) after the start of current injection. Channels colored *red* indicate an open state and colored *blue* indicate a closed state. It should be noted that while the rendering at *B1* was taken 2.5 ms after the start of current injection, it was 100 μs after channel opening.

At this time the Ca^{2+} concentration was 0.43 μM. At 6 ms after current injection, Ca^{2+} concentration is 8.8 μM



```
CompartmentDef: Pre
CompartmentDef: Post
ElementDef: Glu
ElementDef: Ca
ElementDef: Vesicle
    Group: vesicles
ElementDef: Ampa
    Group: ampas
ElementDef: CaM
    Group: cams
ElementDef: CaMKII
    Group: camkiis
ElementDef: Pump
    Group: prepumps
```

```
Shape: Mesh
    File: Pre.obj
    VolumeSize: 5.614249275698954E7
Id: 0
Location: (0.0, 0.0, -100.0)
Volume: Pre
    Module: pop
Module: trace
Surface: Active
    Binding: b1
        Target: Vesicle
        Embed: True
        Action: Vesicle.Set.Reactable.True
        Action: Vesicle.Set.Movable.False
```

Fig. 8.

Snapshot of the Intializer with a model synapse loaded. The *right* and *bottom* panels show a 2D drawing of the simulation space from a *top view*, *side view* and *front view*. The frame at the top left displays the compartments and elements in the world and their properties. Two compartments (pre- and post-synaptic) exist opposite each other with the empty volume between representing the cleft. Elements distributed throughout these compartments include static obstacles, calmodulin, Ca^{2+}/calmodulin-dependent protein kinase II, Ca^{2+} pumps, vesicles, and glutamate receptors. These elements quickly crowd the viewing area and this illustrates the need for the Initializer to have algorithms for automatic distribution of elements. In such a crowded environment, with some exceptions, manually placing elements is not recommended. The presynaptic compartment is selected and its properties are shown on the *right side* of the floating frame. Two output modules are added to this compartment and a sub-region of the mesh has been designated the active zone which allows freely diffusing vesicles to bind to

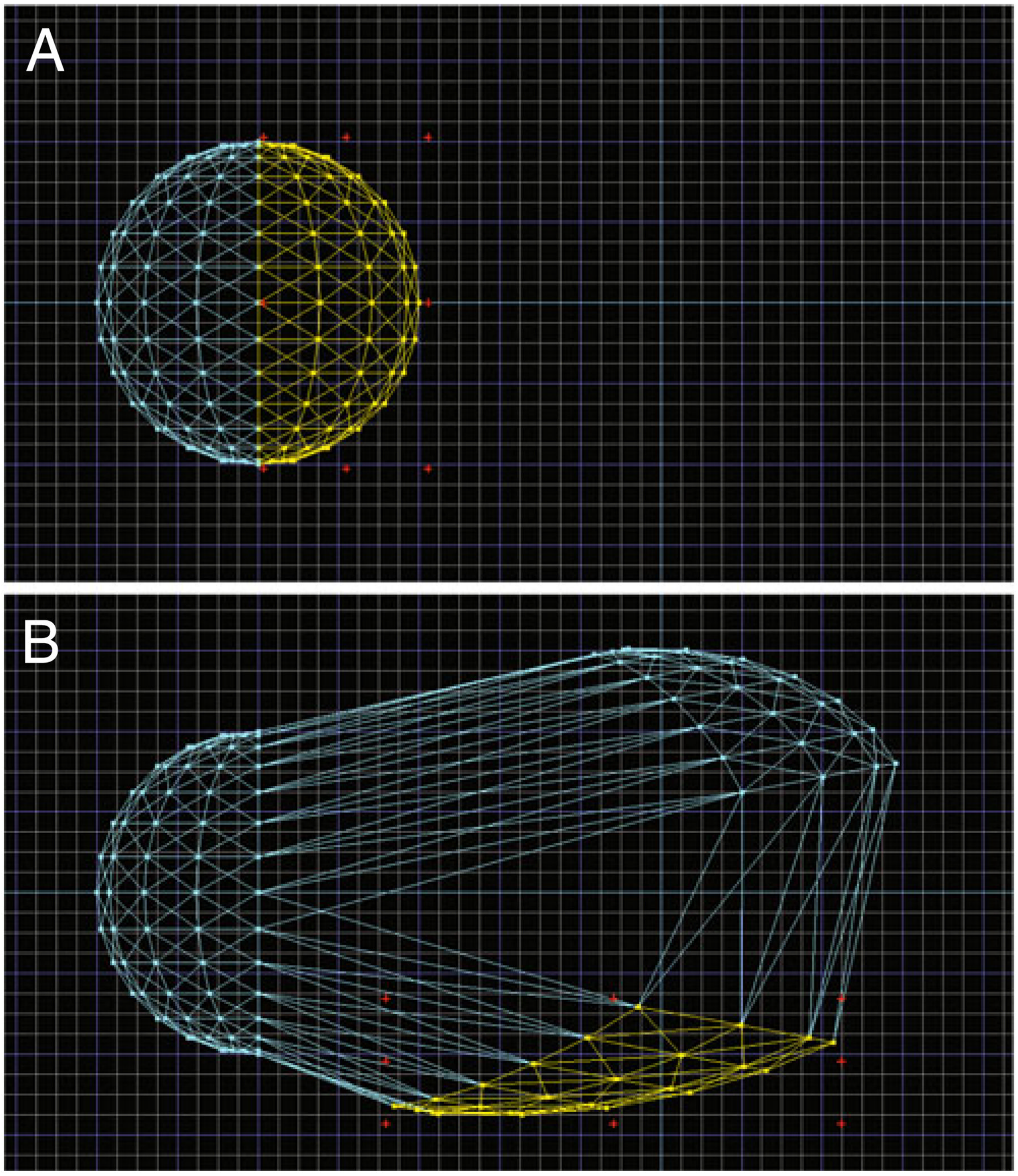


Fig. 9.

a A new spherical mesh was created and in the mesh editor mode, half of the vertices were selected. *Red crosses* on the sides and corners of the selected vertices allow for scaling and rotating by interaction via the mouse. **b** Two groups of vertices from the original sphere have been translated, scaled and rotated to form this new shape. This interaction allows for the construction of more complicated structures from a starting point of a simple prefabricated design

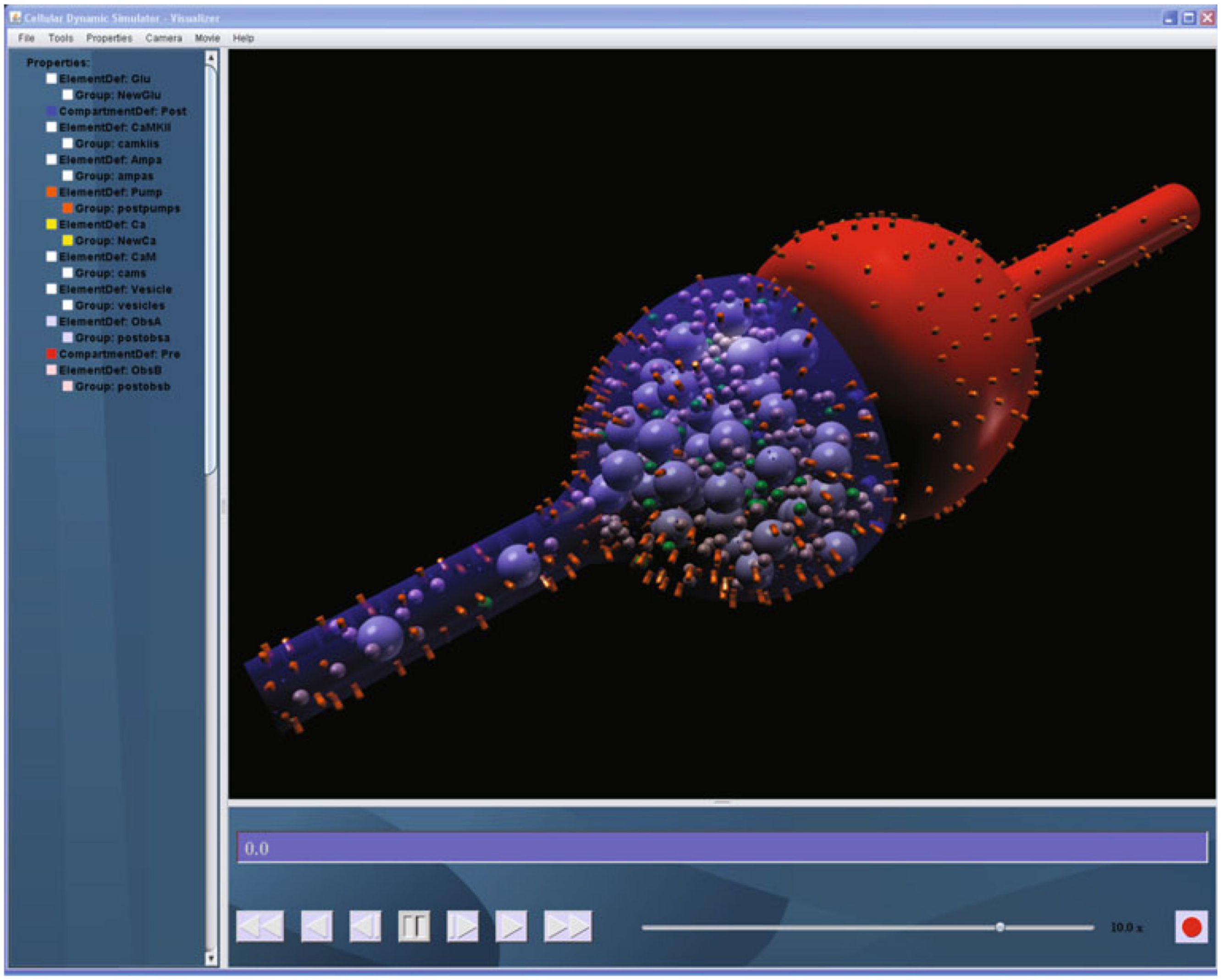


Fig. 10.
A view of the Visualizer window showing a rendering of the simulation described in Fig. 8. Meshes may be rendered opaque or with a variable degree of transparency and can be solid or wireframe. Colors can be assigned by surface, by state and or surface state. Colors can be adjusted by element, by group or by individual instances. Properties applied to an element are automatically applied to all instances of that element. A variety of playback controls allow precise rates of animation. The view can be panned, rotated and zoomed by using the three mouse buttons