FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

CellUnity

an Interactive Tool for Illustrative Visualization of Molecular Reactions

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medizinische Informatik

eingereicht von

Daniel Gehrer
Matrikelnummer 1125229

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ivan Viola
Projektleitung: Mathieu Le Muzic

Wien, 29.09.2014

(Unterschrift Verfasser) (Unterschrift Betreuer)

Erklärung zur Verfassung der Arbeit

Daniel Gehrer

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)	(Unterschrift Verfasser)

Acknowledgements

This bachelor thesis is written in the scope of the research project Visual Computing: Illustrative Visualization and is funded by the Vienna Science and Technology Fund (WWTF).

Kurzfassung

CellUnity ist ein interaktives Werkzeug zur Visualisierung von molekularen Reaktionen basierend auf der Unity Game-Engine. Bei aktuellen Visualisierungen von mesoskaligen Vorgängen wird gewöhlich auf teilchenbasierende Simulationen zurückgegriffen, die die räumlichen Informationen jedes einzelnen Teilchens bestimmen, um ein realistisches Verhalten der Metaboliten zu imitieren. Jedoch nutzt dieser Ansatz stochastische Simulationsmethoden, was jeglichen Einfluss auf das visuelle Ergebnis nimmt. CellUnity hingegen nutzt rein quantitative deterministische Simulationen, was volle Kontrolle über den räumlichen Ort der Reaktionen ermöglicht. Der User kann die Reaktionen auf Wunsch selbst auslösen, anstatt zu warten oder nach einer bestimmten Reaktion suchen zu müssen. Die Menge der angezeigen Moleküle beruht aber stets auf wissenschaftlichen Daten. Die Simulation in CellUnity läuft in Echtzeit, und ermöglicht es dem User auch Simulationsparameter zu verändern, während das System läuft. Dieses Tool wurde mit Unity umgesetzt, einer plattformunabhängigen Game-Engine, die auch in einer kostenlosen Version mit ausreichendem Funktionsumfang zur Verfügung steht, was ein einfache Verbreitung ermöglicht.

Abstract

CellUnity is a tool for interactive visualization of molecular reactions using the Unity game engine. Current mesoscale visualizations commonly utilize the results of particle-based simulations, which account for spatial information of each single particle and are supposed to mimic a realistic behavior of the metabolites. However, this approach employs stochastic simulation methods which do not offer any control over the visualized output. CellUnity, on the other hand, exploits the results of deterministic simulations which are purely quantitative and in that way offering full user control over the spatial locations of the reactions. The user is able to trigger reactions on demand instead of having to wait or search for a specific type of reaction event, while the quantities of displayed molecules would still be in accordance with real scientific data. CellUnity exploits the simulation results in real time and allows the user to freely modify simulation parameters while the system is running. The tool was realized in Unity, a cross-platform game engine that also comprises a free version with adequate functionality and therefore enables easy deployment of the project.

CellUnity and its source code is freely available on GitHub.
`https://github.com/UnityDevTeam/CellUnity`

Contents

CHAPTER 1

Introduction

Biochemistry allows a deep insight into cells and the synergy of molecular processes. Without any visual explanation, biochemistry can be difficult to understand [1]. Hence, it is necessary to visualize these processes to gain a better and more intuitive understanding of what is happening inside a cell [2]. For learning and comprehension purposes it is also important to provide an interactive, game-like environment in order that students can immediately experience the impact of modifications in a cellular environment [3]. Scientific illustrators usually utilize animated storytelling principles to visually explain molecular activities, e.g. a metabolic pathway. To achieve this, corresponding particles and reaction events have to be shown in a story-structured manner [1]. Available mesoscale visualization tools commonly utilize the results of particle-based simulations to generate illustrations depicting reactions of a given biochemical process. Particle-based simulations determine spatial information of each single particle and are supposed to mimic a realistic behavior of the metabolites. However, this approach employs stochastic simulation methods which do not offer any control over the visualized output. [1]

CellUnity is a tool for interactive visualization of molecular reactions. The functionality and the implementation is inspired by the paper of Le Muzic et al. [1]. CellUnity exploits the results of deterministic simulations which are purely quantitative, and therefore offering, in contrast to the existing approaches, full user control over the spatial locations of the reactions. In particle-based simulations it is extremely difficult to track a specific particle due to the chaotic motion. Also reactions cannot easily be observed in the complex environment [2]. Due to this problem, it is challenging to comprehend reactions describing a biochemical process. Even when single particles are tracked and brought to focus, there is still no guarantee that a desired or an interesting event will happen [1]. In CellUnity, however, the user is able to trigger reactions on demand instead of having to wait or search for a specific type of reaction event, while the quantities of the displayed molecules would still be in accordance with real scientific data. This makes it possible for the user to follow a specific reaction chain in a realistic environment, which is greatly valuable for the user's comprehension and for illustration purposes. CellUnity exploits the simulation results in real time. This means that the user can also freely modify simulation

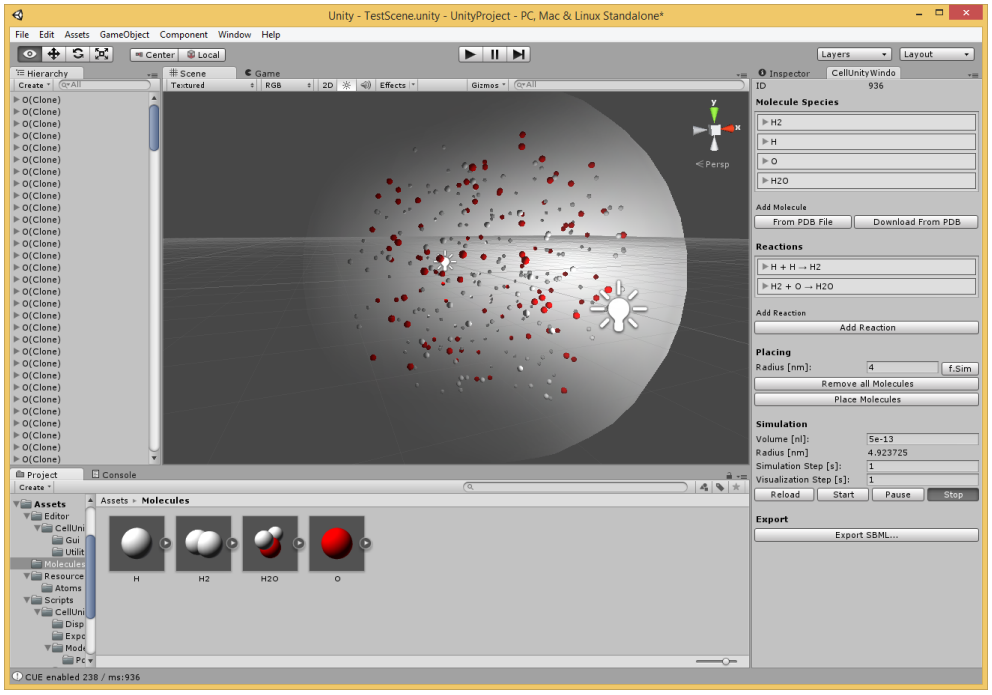
parameters while the system is running. Parameters such as reaction rates and particle quantities can be changed and the results of these changes are immediately visualized.

Often, existing visual simulation environments like ZigCell3D are implemented as proprietary research prototypes that cannot be freely deployed on any machine [4]. Other tools like Molecular Maya or BioBlender are great for visualization but the created environments cannot be animated using a simulator [5] [6]. The main contribution of this work is the implementation of such a visualization and simulation tool in Unity to enable easy deployment. Unity is a cross-platform game engine that also comprises a free version with adequate functionality [7]. CellUnity provides a user interface to create simple bio-molecular environments. It is possible to import molecular structures available from public databases, define molecule quantities, reaction rates, and even to locate individual particles in the environment. The settings can also be exported to bioinformatics standard formats for the usage in external applications. The project is intended as a platform to create biochemically correct and highly detailed simulations and visualizations. CellUnity can be easily extended and is, due to the open platform and the use of a common programming language, ready for further development.

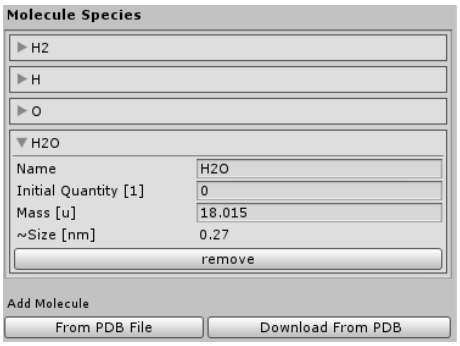
CellUnity is integrated in Unity and provides features to support the user in editor mode as well as in play mode. A screenshot of the CellUnity window is shown in figure 1.1a.

- Environment Editor: CellUnity offers the user a set of tools to create a cellular environment.
  - Molecule Species Editor: The user can specify the molecule species available in the environment. It is possible to either import PDB files or download species directly from the RCSB Protein Data Bank [8]. The species editor is shown in figure 1.1b.
  - Reaction Editor: In the reaction editor, the user can define the possible reactions in this environment. It is possible to set one or more reactants and none or more products. The reaction editor is shown in figure 1.1c.
  - Molecule Placing: The molecules can be instantiated individually or can be automatically and randomly placed in the environment according to the defined initial quantity.

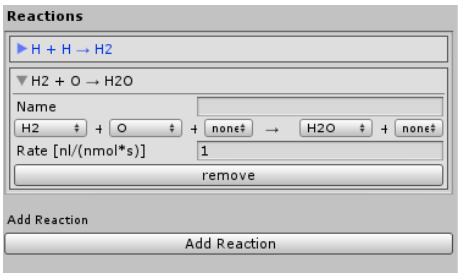
    After the molecules are created, they can be moved and rotated freely using Unity's tool set.
- Simulation: CellUnity is linked to a quantitative simulator and exploits the simulation results in real time.
  - Simulation Control: The user can start or pause the simulation any time.
  - Simulation Update: Simulation parameters can be changed while the simulation is running and can be applied immediately.
  - Spatial Navigation: It is possible to move freely in the environment such as in a first person game, even when the simulation is running.



(a) Unity with CellUnity-Editor



(b) Molecule Species Editor



(c) Reaction Editor

Figure 1.1: CellUnity Editor

  - Particle Tracking: By clicking, the user can select and follow any particle.
  - Reaction Initiation: The user can choose in which reaction the selected particle should be part of. CellUnity will arrange the desired reaction and will make sure the quantity of the displayed particles in the environment remains correct.

- Export: The created environment can be exported for the usage in other applications.
  - SBML Export: The cellular environment can be exported to a SBML-file, a bioinformatics standard format [9].

- Public: CellUnity is openly available, and can be easily installed on a user's computer.

CHAPTER 2

State of the Art

There are already tools for visualizing molecular reactions depicting a biological pathway. All of them have been designed for a slightly different purpose. Yet they all share the goal to provide insight into biological processes by visualizing a cellular environment at mesoscale levels. In this chapter various available tools are examined in respect to their visualization capabilities for scientific correct mesoscopic storytelling to explain cellular activities.

2.1 Tools for Molecular Visualization only

Molecular Maya Toolkit

Molecular Maya is a free plugin for the 3D computer graphics software Maya [10] that allows to import, build and animate molecular structures. The user can open PDB files or download them directly from the RCSB Protein Data Bank [8]. Molecular Maya supports various representation forms and also enables the user to easily extend structures, for example creating surface meshes and biological units [5].

The Toolkit can be used to illustrate molecules but the models do not convey information about its function. To illustrate a cellular environment, the illustrator has to model the molecular processes manually using Maya's default tool set, which is a time-consuming and expensive task, taking up to months or years [1].

BioBlender

BioBlender is an engine built in Blender for the visualization of bio-molecules. The engine integrates in Blender's user interface and allows the import of molecule data and provides a large set of visualization and render options. BioBlender can animate transitions of conformations and visualize various molecular features, e.g. the electrostatic potential (EP) and the molecular lipophilicity potential (MLP). This kind of representation makes the nano scale world more

understandable and is making it easier to conceive invisible phenomena such as hydropathy or charge [6].

BioBlender gives insight into special molecular properties, which makes it great for examining individual molecules. However, it is unsuitable for the visualization of crowded molecular environments.

Embedded Python Molecular Viewer

The embedded Python Molecular Viewer (ePMV) is an open-source plugin for common 3D software, which allows the import of molecule structures to depict them in the context of a cell. It was created to facilitate the production of visual illustrations for publications and for educational purposes. There are viewers for all kinds of scientific data but mostly the viewers lack in putting multiple structures into a common context, especially when data of multiple interdisciplinary researchers are combined [11]. The plugin is available for Blender [12], Cinema4D [13] and Maya [10] [11].

ePMV does not only import molecules from different file formats but also keeps the link between structure file and the model. That way changes that are applied to the structure file after the import, are also applied to the model. The generated model is not just a static structure but can also be manipulated by the 3D host program or by python scripts that interact with ePMV. That way a protein e.g. can change its conformation, or the visualization can be linked to a simulator [11].

ePMV is very flexible and optimized for visual illustration. It is even possible to create interactive applications. Tasks like simulations are currently hard to use and for this reason still under development. The use as potential research tool is further examined [11].

2.2 Tools for Molecular Visualization and Simulation

Visualization of Signal Transduction Processes

Falk et al. developed a visualization framework to explore simulation data of a virtual cellular environment [2]. The goal of the work was to highlight events of interest in the confusing environment. It especially helps Biologists to follow signaling molecules through the cell. The user can interactively select individual molecules and zoom into the virtual cell. It is possible to visualize individual molecules, their tracks, or reactions. They can be selected and brought into focus in order to highlight the signal transduction pathway. A simulation may include thousands of proteins, obstacles, and filaments which are schematically visualized as three dimensional glyphs. The work is suitable for detailed, realistic, spatial simulations, where each molecule is an independent agent. A simulation usually covers several hundred frames. The user can step through each frame by keystrokes. To smooth the motion of the particles in the visualization, the molecule position is interpolated between two simulation frames. An animation over all frames

is also provided. The work also includes a virtual microscope to create images which can be compared with results from wet lab experiments [2].

The work presented by Falk et al. provides advanced analysis tools for simulation results. It also allows to visualize, analyze and follow the trajectories of each particle which is valuable to comprehend signal transduction processes. While the analyzing tool is interactive, the simulation is not [2]. The user has to perform the simulation again before it is possible to see the effects of the changes made. Also the tool is not openly available and therefore only used by a small set of users.

MCell and CellBlender

MCell (Monte Carlo cell) is referring itself as micro physiological simulator [14]. It is a program to simulate the movements and reactions of molecules within and between cellular regions. For simulation, MCell uses spatially realistic 3D models and specialized Monte Carlo algorithms. It is intended to realize as realistic simulations as possible. Diffusing chemical species in solutions, as well as in membranes, are supported. The model can contain multiple compartments, which represent enclosed parts, e.g. organelles [14]. Creating the great numbers of initial mesh objects for spatially complex models is one of the most challenging parts. The meshes can be obtained from segmented volumetric imaging data or from CAD (computer-aided design) software [15]. MCell is free of charge and available for Linux, OS X, and Windows [14]. MCell does not have a graphical user interface and can be exclusively run from the command line. The model and the simulation conditions are defined in modular, human-readable text files, using a model description language [15].

CellBlender is an add-on for the free and open-source 3D computer graphics software Blender [12] [16]. The add-on is closely linked to MCell and enables the user to perform integral modeling tasks in Blender. It is possible to create, edit and visualize cellular models for the use in MCell. The simulation results generated by MCell, again, can be visualized in Blender, including the locations and states of participating meshes and molecules. Additionally CellBlender offers basic support for SBML [14].

The simulation in CellBlender is not interactive and is very realistic, including the fast, stochastic movement of the molecules. This is great for creating a correct representation, but not for storytelling since it is challenging to locate molecular events in time and space.

ZigCell3D

ZigCell3D is a software for modeling, simulating and visualizing an entire cell. The visualization covers several orders of magnitudes, the full range from the cell surface to organelles and molecules down to the atomic level. The movement of the molecules as well as their interactions in the cell are also visualized and show how cellular processes emerge from the molecular level. The system also includes a virtual fluorescence microscope. That way the user can verify the simulated model against a real experimental fluorescence microscopy image. For simulation

two different approaches are used. On the one hand particle-based Brownian dynamics simulation, and on the other hand simulations based on Reaction Diffusion Master Equations (RDME), which have less spatial resolution but better performance. The application also supports PDB and SBML files [4].

ZigCell3D provides a real time interactive environment, where model parameters can be changed and the resulting effect can be analyzed in the 3D visualization of the cell. As a consequence complex biophysics research obtains creative and game-like elements. The user can select particles out of the visualization. The application then displays the underlying rules and mathematical expressions that are responsible for the creation of the selected component or molecule. That frees the user from guessing the possible origin and bridges the divide between quantitative sciences and math-free wet-lab biology [4].

A great magnitude from the nano to the macro scale is covered by ZigCell3D. The simulations are realistic and very detailed. ZigCell3D is a research prototype that cannot be freely deployed on any machine. Also the use of a spatial simulation technique could make it hard to use the results for structured storytelling.

Visual Molecular Dynamics

Visual Molecular Dynamics (VMD) is the visualization program of the Interactive Molecular Dynamics (IMD) system. VMD is linked to a molecular dynamics program (NAMD). NAMD can run on a single or on multiple machines. Coupled with a haptic feedback device the system allows manipulation of molecules in a molecular dynamics simulation. The visualization and the force feedback are responding in real time. VMD was developed as front-end for IMD and is designed to work with 3D spatial trackers and other non-traditional input devices. The molecular structures in VMD are not static like in most applications, but also visualizes dynamics inside the molecule. The system was designed to handle large, complex biomolecular systems. The scalability of NAMD enables to examine small biological systems as well as large ones, given enough computational power [17].

VMD as part of IDM is a very extensive tool which simulates biomolecular systems down to intra molecular phenomena. Moreover it is free of charge and is available for MacOS X, Unix and Windows [18]. Because of the advanced attention to detail and the associated high computational power it may become impractical for cellular illustrations on personal computers.

CHAPTER 3

Methods

The aim of this work is to create a tool for visualizing an illustrative molecular environment. The environment is animated and coupled to a simulator to provide a scientifically accurate visualization. For better comprehension of cellular processes, the tool is interactive, so that the user is able to explore and immerse into a virtual 3D cellular environment, track molecular compounds and also trigger reactions manually, while respecting quantities obtained via scientific simulation. Moreover, simulation and visualization occur in real time, to enable the user to observe the effects of changes made in the environment immediately. To fulfill all these requirements, several tools and concepts are used, which are introduced in this chapter.

3.1 Development Environment

The project is built in Unity, a cross-platform game engine. Unity is not only a game engine but also includes an integrated development environment (IDE). The reasons why Unity was chosen as framework are the following: it is easy to use, quick to learn and there is also a free version with adequate functionality to realize this tool. Moreover, this enables the project to be easily shared and deployed, and allows the user to modify or extend the project with little effort. Additionally, Unity provides built-in methods for visualizing 3D objects and has a built-in physics engine. These features speed up development and avoid that the project has to be created from scratch. Furthermore, the Unity editor can be extended easily to include custom plugins, which seamlessly integrate into the Unity interface. Another beneficial feature is Unity's ability to build projects for a lot of different platforms [7].

3.2 Visualization of Molecules

Molecules are visualized at atomic resolution. CellUnity can import molecule species from the file system using PDB files or can download the structure information automatically from the PDB webserver using a given PDB ID [8]. The representation of a molecule is automatically

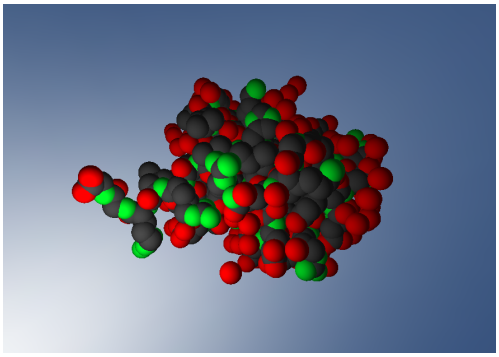


Figure 3.1: Visualization of an ubiquitin [19] molecule in CellUnity

created using the atom definitions in that PDB file. The imported molecules are displayed as bunch of spheres, each representing an atom. The locations of the atoms are in accordance with the PDB file. The size of each atom corresponds with the Van der Waals radius of the associated element. This representation is often referred as Van der Waals representation [1]. An example is shown in figure 3.1.

3.3 Simulation

CellUnity is coupled to a simulator to mimic a realistic behavior in the visualization. For simulation the biochemical simulator COPASI is used. COPASI provides an application programming interface (API) for C# that is used to communicate with CellUnity [20]. Via this interface, model data is exchanged. Since only the number of reactions occurred are needed by CellUnity, the simulation is purely quantitative. The simulation is performed step by step. After each step, the results are transmitted back to CellUnity and the reactions are performed in the visualization. It is also possible to modify simulation parameters after each step. The duration of such a step is adjustable by the user.

3.4 Molecular Dynamics

The motion of molecules are controlled by Unity's built-in physics engine. Random forces are applied to each molecule to mimic diffusion. It is important to precise that the molecular diffusion constants are not realistic in CellUnity, because it would create too much visual clutter. Instead an arbitrary, random walk motion is applied that simply illustrates the real diffusion mechanism.

Reaction events are solely triggered by an omniscient intelligence (OI), like proposed by Le Muzic et al. [1]. Molecules in CellUnity are unable to initiate reactions but can only receive reaction orders from the OI. In this system, molecules are passive agents, according to the definition by Kubera et al. [21]. The OI is influenced by the simulator and controls the molecules accordingly. Thus reactions in CellUnity do not just happen but are actively forced. The OI

uses the current simulation state and takes action to achieve the same state in the visualization. Concretely, the OI reads out the quantity of reactions that occurred in the simulation for each reaction type, and forces the same quantity of reactions to happen in the visualization. Therefore the simulation and the visualization are quantitatively synchronized [1].

When a reaction is initiated, the OI selects candidates according to the species of the reactants and applies mutual attraction forces to force reaction partners to meet. As soon as they collide, the reaction is performed and the reactants are replaced by the reaction products. Collisions are also detected by the physics engine. Depending on if colliding molecules should react, the reaction takes place, if not, they repel each other via bouncing motion [1].

3.5 Navigation and Storytelling

To allow the user to navigate through the molecular environment, a navigation similar to a first-person-game is enabled. The user can turn around using the mouse and move with the W, A, S, D keys. Clicking on a molecule selects it. The user can adjust the view to automatically follow the selected molecule in the space or tag it for a reaction. If a molecule is tagged for a reaction, its priority will be set to react first when the OI initiates a new reaction. This allows the user to easily follow a reaction chain along a metabolic pathway. This is an easy way to comprehend reaction chains without having to wait until reactions happen on themselves, and is useful for storytelling.

3.6 Data Persistence

Unity usually serializes game data into so-called assets to persistently save them [7]. CellUnity uses this feature to save environmental data like molecule species and reactions. Molecules are implemented as GameObjects and therefore can be saved and restored in scenes when Unity is in edit mode. The position of every molecule is also preserved that way. In game mode the scene cannot be saved but the current state can be exported to an SBML file. The export functionality is available in edit mode as well. The SBML functionality is acquired by an external library.

CHAPTER 4

Implementation

CellUnity is implemented as a project inside Unity. Custom editors are used to allow the user to configure the CellUnity environment. When custom editors are used, Unity demands to split the project into an editor part and a project part. Classes in the editor part can extend the Unity editor and seamlessly integrate into the Unity interface. CellUnity's implementation is divided into individual classes. It is heeded that responsibilities of each class is well defined, to ensure coherent program modules that are as independent as possible. The CellUnity environment implements the model of the cell, the custom editor serves as controller for this model and Unity provides the visualization. Together these components form a model-view-controller. Figure 4.1 shows the structure of the project.

One custom editor window is implemented for CellUnity. Via this editor the user can model and modify the environment. It is possible to import molecule species from PDB, to add and remove reactions from the system, configure simulation properties and export the environment to an SBML file. The target of the changes made in the editor is the CellUnity Environment (CUE), which holds all the environment properties and definitions. Furthermore, the CellUnity Environment manages all environmental events like reactions.

Another important aspect is the units in the system. Different modules are using different units that must be converted in order to interact with each other. A related issue is the scaling of the visualization. A reasonable scaling from the atomic level to the Unity view unit has to be applied. To consistently solve this problem, a converter and scaling class has been implemented.

4.1 CellUnity Environment

The CellUnity Environment (CUE) is the class that holds all environmental properties and definitions. The entire system can only contain one instance of a CUE, therefore it is implemented as singleton. The CUE contains the defined molecule species and reactions, the volume of the

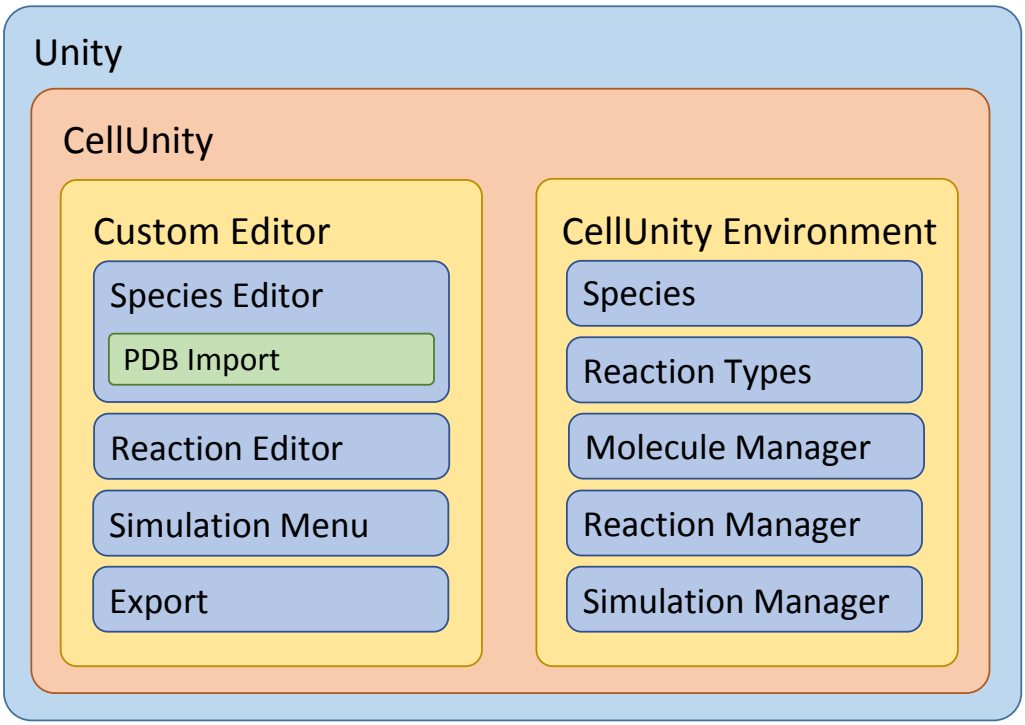


Figure 4.1: CellUnity: internal structure

compartment, a molecule manager, a reaction manager and a simulation manager. Each manager focuses on a separate task. They are described in detail in later sections.

Saving

Because all the environmental information is stored in the CUE, it makes sense to simply serialize the instance to persistently save the entire model when Unity is closed. Unity already provides automatic serialization methods. However, a few specific characteristics must be considered when used. Multiple references to one instance of a class are serialized multiple times, therefore, for every reference a new instance is created after restoring. This behavior is not satisfactory for species and reaction instances. Therefore these classes are derived from `ScriptableObject`. `ScriptableObjects` are serialized only once and multiple references are restored correctly [7].

Compartment

The CUE currently only supports one compartment and it has to be in the shape of a sphere. To keep the molecules inside the compartment, collisions with the compartment wall must be detected and the particles must bounce off. The physics engine of Unity does not support in-

verted colliders, which would be required. Therefore the desired behavior must be implemented by user code [7]. For each molecule, the distance to the compartment center is calculated. If the distance exceeds the radius, the distance is set to the value of the radius and the velocity of the molecule is inverted.

4.2 Molecules

In CellUnity, molecule representations are `GameObjects` with a `Molecule`-script attached. The `Molecule`-script applies molecular behavior to the `GameObject`. Each molecule is an instance of a specific molecule species. To make it easy to insert new molecules, each species has a prefab asset. A prefab is a Unity asset type that allows to store a GameObject with all components and properties. It acts as a template from which it is possible to create new instances in the environment [7]. The prefab is automatically created when a new species is imported.

PDB Import

CellUnity can either import PDB files from the file system or download them directly from the PDB website. PDB files are text files which contain 3D structure information of biological macromolecules [22]. For the molecule representation, only the atom positions in the file are considered. To gather this information, a simple PDB parser was written. The locations of the atoms are specified in Angstroms and have to be scaled before using for the visualization.

When a new molecule species is imported, at first, a new `MoleculeSpecies`-instance is created and added to the CUE. Then an empty `GameObject` is created, which serves as the main object of the molecule. The main object gets the `Molecule`-script attached and the newly created species-instance assigned. All atoms defined in the PDB file are now created as sphere-primitives and are added as sub-objects to the main object. To get a Van der Waals representation of the molecule, the size, location and color of each sphere is set accordingly. By default each sphere-primitive has a collider attached and is therefore considered by the physics engine [7]. A biomolecule usually consists of thousands of atoms. With each having its own collider would result in an immense computational expense. Hence, all colliders are removed from the atoms and only one spherical collider is added to the main object that contains all atoms. The newly created molecule is then saved to a new prefab asset and assigned to the new species as the template for the molecules.

Molecule Manager

The assignment of the `MoleculeManager` is to keep track of all molecules in the system. When the play mode in Unity is activated, each molecule registers itself to the `MoleculeManager` of the CUE. In the manager, all molecules are organized in separate lists, depending on their species and whether they are free or already in use for a reaction. These lists are implemented as double linked lists. One molecule can only be in one list at a time. This enables to find free molecules quickly and efficient, which is important for the preparation of reactions. The organization of the molecule manager is depicted in figure 4.2.

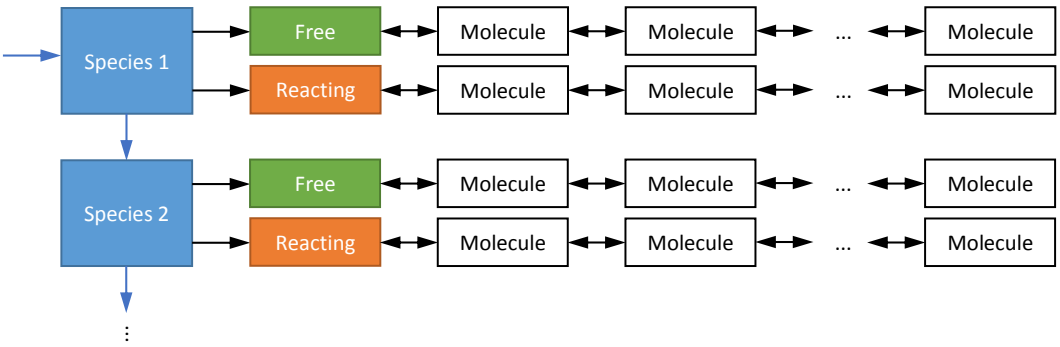


Figure 4.2: Schematic Diagram of the Molecule Manager

When a new reaction is initiated, the reaction manager asks the molecule manager to find a set of molecules of specific species that are near together. Due to the organization in the molecule manager, the nearest free molecule of a specific species can be found in $O(n + m)$ where n is the number of species, which is usually very small, and m is the number of molecules which are "free" and of the defined species, meaning only a fraction of all molecules. When a molecule's state changes from "free" to "reacting", it has to be removed from the "free" list and added to the "reacting" list. This state change can be performed in $O(1)$.

4.3 Automatic Molecule Placement

Since it is impractical to place each molecule manually, CellUnity offers to place a defined initial quantity of molecules automatically and randomly. The initial quantity can be set in the species editor, and placed in the placing menu. A problem that can possibly occur is that due to the randomness two or more molecules are placed too near together so that their colliders intersect. When this happens in Unity, particles repel each other with an unusually strong collision response which we ought to avoid. In CellUnity the problem is solved using the physics engine itself. The initial drag of the molecules is set to a very high value. As a result, colliding molecules repel each other gently until they do not intersect each other any longer and then remain steady next to each other. After 2 seconds, the drag is reset to its normal value. This procedure is only performed once, when the molecules are placed.

4.4 Reactions

Reactions that are possible in the system must be defined to the CUE. They are stored as `ReactionType`-instances. A reaction type consists of one or more reactants, zero or more products and a reaction rate. Reactants are the molecule species that are needed to perform a reaction. Products are molecule species that are produced when a reaction was performed. Whereas stoichiometric coefficients cannot be set directly, it is possible to add the same species multiple times to the reactants or products list, which has the same effect. The rate defines how

many reactions should happen in a given amount of time. The reaction law is by default “Mass action (irreversible)” and cannot be changed in the current version of CellUnity. A reaction is performed when all reactants collide due to steering forces, just after receiving a reaction order from the OI.

Reaction Manager

The assignment of the `ReactionManager` is to initiate reactions and to perform them when all reactants collide. Reactions are usually initiated by the simulation manager. When a reaction of a specific reaction type is started, a new `ReactionPrep` (short for reaction preparation) object is created, which stores all important information about the reaction. Then the reaction manager asks the molecule manager to choose some free molecules of the species of the reactants for the reaction. The user can influence which molecules are chosen by selecting them. A selected molecule is preferably used as reactant. If no molecule is selected, the reactants are picked randomly. If not enough molecules are available, the reaction is noted and delayed until enough molecules are available. This is important to guarantee correct molecule quantities on the long run. Such a delay can happen when the simulator is faster than the visualization, for example when the reacting molecules are located far apart, or when they hit obstacles which slow them down before reacting. However, if enough molecules are available, they are linked with the `ReactionPrep` object. Every molecule can only be linked to one `ReactionPrep` object at a time. Molecules linked to the same `ReactionPrep` instance and therefore are reactants of the same reaction, attract each other. This ensures that they will collide. When two molecules collide, they inform the reaction manager. The reaction manager checks if they belong to the same reaction, if yes, they are both tagged as “ready”. As soon as all reactants are ready, the reaction is performed. This means that the reactants molecules are replaced by new product molecules. For visual guidance purposes a short flashing light is displayed during the reaction. An example of the whole reaction process is illustrated in figure 4.3.

When a reaction is initiated, the reactants attract each other and accelerate towards each other. Due to their physical properties they possibly do not collide immediately but start to orbit the common barycenter. This can result in an endless circulation with the molecules never collide. To avoid this, a drag is set in the environment for all molecules. As a consequence orbiting molecules slow down and collide after a short time.

4.5 Simulation

CellUnity is coupled with COPASI, a tool for quantitative modeling and simulation. COPASI is used to simulate the user defined molecular environment. The communication is enabled via the C# application programming interface (API) provided by COPASI [20]. The simulation is started and administered by the simulation manager. The manager is also responsible for the data transfer with the simulator as well as the utilization of the simulation results.

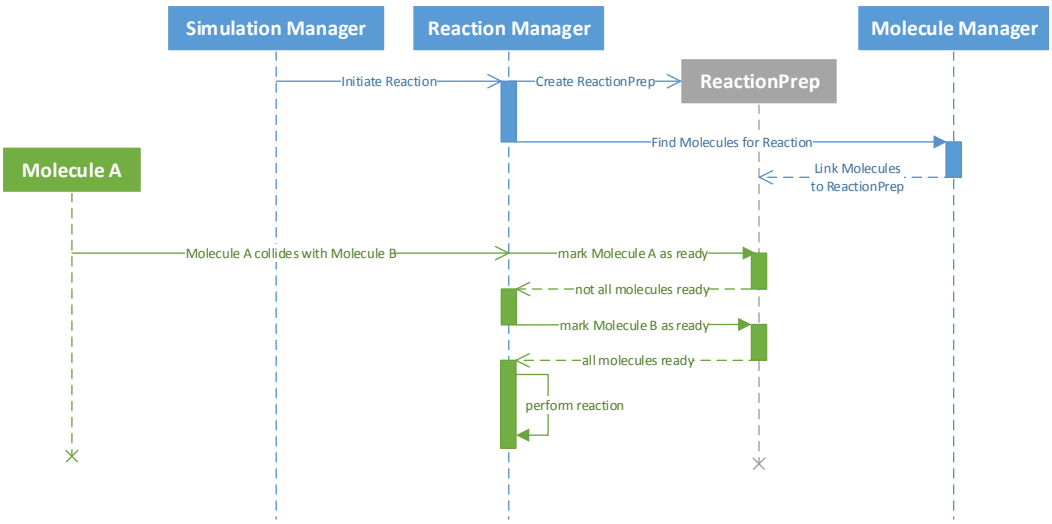


Figure 4.3: Sequence diagram of a reaction $A + B \rightarrow C$ that is initiated by the Simulation Manager, assuming the environment contains enough molecules.

Simulation Manager

Prior to the real-time simulation of the environment, the CellUnity model has to be transferred to COPASI. The compartment, the species and the reactions from the CUE are added to COPASI via the API. Since COPASI uses its own entity classes, a link between the COPASI entities and the CellUnity entities is established, in order to associate COPASI results correctly with CellUnity resources. The initial quantities of the species in COPASI are set to the count of the species currently located in the CUE. Because CellUnity only pursuits of quantitative correctness, everything needed from the simulator are the number and types of reactions performed in the simulation. To gather this value for each reaction type, a "global quantity" model value is added. The value is defined as the ParticleFlux of the particular reaction. The type of the model value is set to "ode", so the value is the total value of performed reactions of this type.

After the model is transferred and compiled, COPASI is ready to use. The simulation is performed in steps. In CellUnity, there is a time for the "visualization step" and a time for the "simulation step" that the user can define. The "visualization step" is the real time interval of a step. The "simulation step" is the time simulated in such a step. That means that a visualization step of 1 second, and a simulation step of 2 seconds, would result in a double-speed visualization, because for every second that passes in real time, 2 seconds are simulated. After each simulation step, the ParticleFlux of each reaction is compared with the value before that step. The difference is the number of reactions performed during this step. The simulation is run in an extra thread, so it does not disturb or interrupt the main thread which is used for the visualization. To transfer

data between these two threads, the reaction manager provides a thread-safe queue, where the simulator enqueues its results and the main thread dequeues the results and initiates reactions according to the simulation results.

The CellUnity model is interactive. It is possible to change parameters, even during simulation. This is achieved by re-transferring the model to COPASI when something changed. Instead of the original initial values, the initial species quantity of COPASI is set in accordance to the current model state. In experiments, the COPASI model was synchronized with the CellUnity model instead of recreating it. But the synchronization process became quickly very complex and did not have any real advantage. Therefore this simple solution was implemented.

4.6 Export

CellUnity is able to export the created model to an SBML file [9]. Because COPASI is already integrated in CellUnity and because the COPASI API provides SBML functionality by default, COPASI is not only used for simulation, but also for SBML export [20]. For exporting, the CellUnity model is transferred to COPASI like in the simulation. The only difference is that no global quantity model values are added. Firstly, because they are not necessary for the model, and secondly, because SBML cannot express the mathematical expression that is used for determining the particle flux. After the SBML file has been written, the COPASI model is released.

CHAPTER 5

Discussion

In this chapter CellUnity is compared with the related work presented in chapter 2. Differences, advantages and disadvantages of the tools are summarized. Later on open issues in the current CellUnity version are discussed.

5.1 Relation to Prior Work

CellUnity and almost every related tool provides support for PDB files and also direct download from the online Protein Data Bank [11] [5] [4] [6] [18]. Tools like MCell combined with CellBlender and ZigCell3D strive to maximize realism and therefore make use of spatial simulation methods [16] [4]. This kind of simulation results are unpredictable due to high randomness and the chaotic particle movement. This makes it hard for the user to follow a specific reaction chain [2]. CellUnity on the other hand uses the quantitative result of deterministic simulations and forces the reactions to happen in the visualization instead of waiting for them to happen randomly as technically described in the paper by Le Muzic et al. [1]. Furthermore, CellUnity allows the user to follow particles and to perform reactions in the sight of the user. ZigCell3D and the tool of Falk et al. can depict the pathway of a particle, but the user cannot influence the reactions like in CellUnity [4] [2]. ZigCell3D as well as CellUnity use real-time visualization and allow the user to modify simulation parameters while the simulation is running [4]. MCell and ZigCell3D also allow to simulate systems with multiple compartments, whereas CellUnity only supports one [14] [4]. A lot of tools are realized as plugins for host applications. For example CellBlender is integrated in Blender and allows the user to use Blender's rich tool set to create the molecular environment, including custom compartments [16]. CellUnity uses Unity as host application but only supports imported molecules and only a sphere-shaped compartment. CellUnity solely provides the Van der Waal representation of molecules. Other tools, especially the tools which are tailored only for the production of educational and scientific illustrations, like ePMV and Molecular Maya Toolkit provide a wide range of representation methods [11] [5]. BioBlender even implements innovative visualization options like depicting

the molecular lipophilicity potential in addition to the usual options. ePMV and BioBlender can visualize different conformations of a molecule [11] [6]. However, these tools do not automatically animate the environment [11] [5] [6]. They rely on the host application's tool set to animate the environment manually. The manual animation tasks are very time-consuming because of the high number of participating particles and the complex dynamics of molecular systems [1]. CellUnity, CellBlender, ZigCell3D, VMD and the work of Falk et al. rely on simulation data which avoids the tedious manual animation tasks [16] [4] [17] [2]. ZigCell3D supports multiple levels of magnitude which makes it necessary to switch the visualization of molecules from atomic resolution to simple geometric objects, depending on the current magnitude level. The tool of Falk et al. and ZigCell3D both additionally allow to display the cell through a virtual microscope [2] [4]. Because CellUnity is hosted in a game development platform, it is easy to modify or extend its functionality. Tools based on Blender offer similar possibilities. MCell, CellBlender, ePMV, Molecular Maya Toolkit, BioBlender, VMD and CellUnity are available for free, often for multiple platforms [14] [16] [11] [5] [6] [18]. ZigCell3D and the work of Falk et al. are research projects which are not available for the public [4] [2].

5.2 Open Issues

CellUnity has a few issues that should be considered. The currently used visualization method is not very efficient. For atom representation a default Unity sphere primitive is used with each having more than 500 vertices. Already several hundred molecules consisting of a few hundreds atoms reduce the frame rate significantly.

The reactions in CellUnity are indeed initiated by the simulation, but the time needed until the reaction is performed and completed is not well defined. Depending on distance and obstacles between the reactants, the reaction can take a long time until it completes. In extreme cases this can lead the visualization to lose synchronization with the simulation, because the reactions are taking place too slowly. This problem is partly avoided by delaying currently unfeasible reactions, but if the delay queue grows faster than reactions are initiated, the latency of the visualization compared to the simulation will grow as the simulation proceeds.

The deflection of the molecules on the compartment wall is currently achieved by user code. The location of each molecule is checked in every frame and reset to the compartment wall, if a molecule has tried to escape. Further research should focus on finding a way to solve this issue more efficiently by the physics engine.

CHAPTER 6

Summary and Future Work

This thesis presents an interactive tool for illustrative visualization of molecular reactions. It enables the user to build a simple molecular environment and simulate it in real time. It is possible to import molecular structures available from public databases, define reactions, and locate molecules in a compartment. Existing visualization tools commonly utilize particle-based simulations to generate illustrations depicting reactions. This approach provides highly realistic visualization, however, it does not offer any control about the visual output. Due to this, it can be hard to follow a specific chain of reactions, because reactions occur randomly and it is not guaranteed that anything interesting will happen in the user's sight. CellUnity, on the other hand, allows the user to trigger reactions and can automatically follow molecular reactions along a metabolic pathway. Even though the user interacts with the environment, the visualization remains in accordance with real scientific data. This enables the user to experience and comprehend metabolic processes. The model created in CellUnity can be exported as SBML file and used in other applications. Another advantage is that only free software is used to develop CellUnity. Hence, CellUnity can be easily deployed, modified and extended by everyone.

Extension possibilities of CellUnity are almost unlimited. At the moment an inefficient molecule representation method is used (see section 5.2). It is planned to implement the rendering technique proposed by Le Muzic et al. [1]. Further work could also include additional representations, so that the user could switch for example between the 3D animation and a 2D annotated graph. Also more detailed reaction animations would be desirable. The reactants could dock together spatial correctly dependent on their conformation and charge. Also Unity's game engine functionality could be used to design more realistic biological environments. Meshes that are modelled from real biological structures and textures designed by artists could be imported. These models could be used for alternative representations or for visualizing currently invisible cell structures like cytoskeletons. Unity itself is a cross-platform development environment [7]. CellUnity, however, can only be executed on Windows computers because the Windows COPASI DLL is used to communicate with the simulator. In further versions this could be adapted

to support other operating systems as well. Another valuable feature would be the ability to produce and export illustrative videos directly inside CellUnity.

APPENDIX A

Download and Installation

1. Navigate to `http://unity3d.com/unity/download` and download the current version of Unity.
2. Run the Setup file to install Unity.
3. Clone the git repository from `https://github.com/UnityDevTeam/CellUnity.git`. If you want to download CellUnity using your browser, open the URL in your browser and click on the "Download ZIP" button on the right hand side of the website. Then extract the ZIP to your file system.
4. Start Unity. Unity will ask you to open a project (see figure A.1). Click on the "Open Other..." button and select the "UnityProject" folder in the repository downloaded from GitHub.
5. CellUnity is now opened and ready to use. The Unity window should look similar to figure A.2. If the CellUnity menu on the right hand side is not shown, you can open it by clicking on "Window → CellUnity".

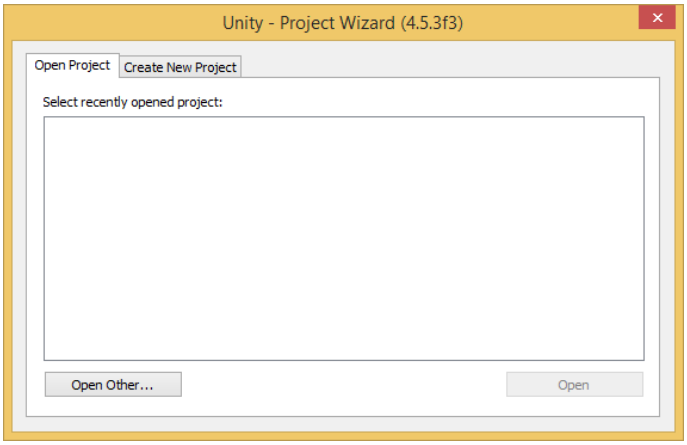


Figure A.1: Unity Project Wizard

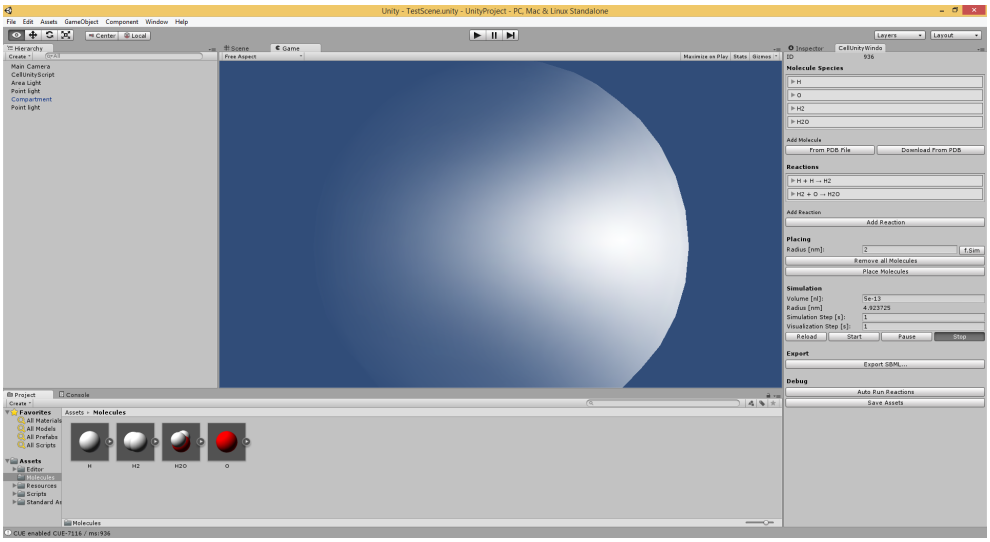


Figure A.2: CellUnity opened in Unity

APPENDIX B

Configure and Run a Simulation

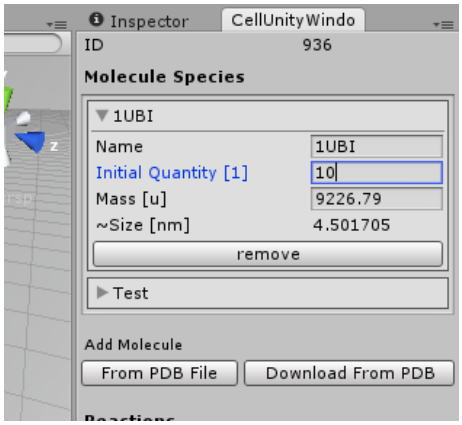
1. *Import the needed species:* In the CellUnity editor there is a "Molecule Species" section (see figure B.1a). All imported species are listed there and you can import more species by clicking the buttons "From PDB File" or "Download From PDB". When you click on "Download From PDB" a window opens where you have to enter the PDB ID of the species you want to import. Click on "Download Molecule" and the species is automatically downloaded and imported. When you select a species in the list, you can edit its properties or remove it.

2. *Define the reactions:* Beneath the species section, there is a "Reactions" section (see figure B.1b). By clicking on "Add Reaction" a new, empty reaction is added to the system. You can optionally enter a name for the reaction and create the reaction equation by selecting the species in the pop-up-list. You should also set an adequate reaction rate.

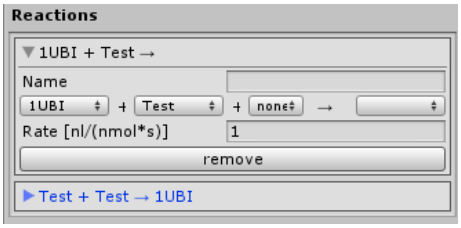
3. *Place the molecules:* When you open the "Molecule" asset folder (see figure B.1c), you can instantiate and place the molecules individually via drag and drop. You can also use the the "Placing" section in the CellUnity menu to randomly place the molecules (see figure B.1d). The number of molecules instantiated equals the "Initial Quantity" defined in the "Molecule Species" section. "Radius" defines the radius of the sphere in which the molecules are placed. By clicking on "f.Sim" the whole radius from the simulation compartment is used for placing.

Simulation

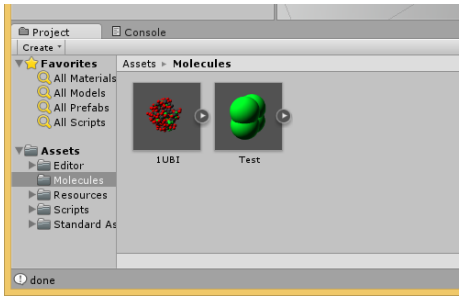
1. *Configure simulation:* In the "Simulation" section of the CellUnity editor, you can set up the compartment volume and duration of the simulation and visualization step, which are explained in section 4.5.



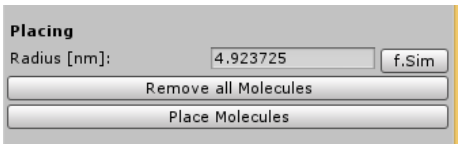
(a) Molecule Species Editor



(b) Reaction Editor



(c) Molecule Asset Folder



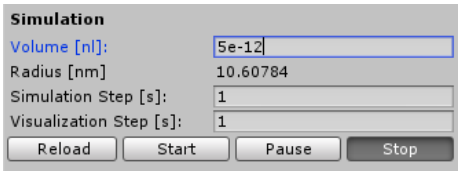
(d) Placing Editor

Figure B.1: CellUnity Editor Sections

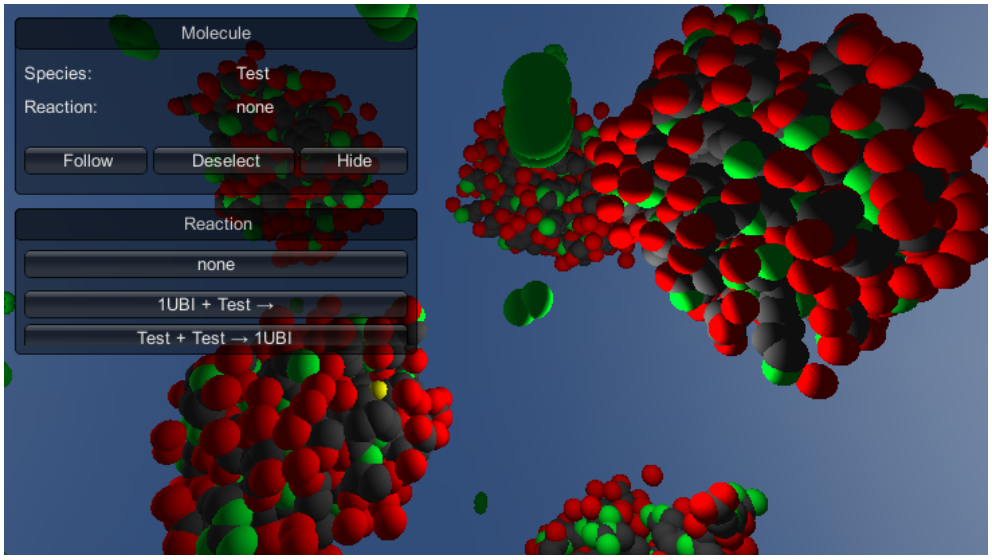
2. *Switch to game mode:* When everything is defined and placed, you are ready to run the simulation. Click on the play button on the top of Unity to switch to game mode. You can look around by pressing on the right mouse button and move the cursor. You can also move while holding the right mouse button pressed and using the keys W, A, S and D.

3. *Start the simulation:* Click on "Start" in the simulation menu and you can see the molecules start moving and reacting. You can press on "Pause" and "Stop" any time. If you changed the model (e.g. added a new reaction or changed the reaction rate), press "Reload" and the simulation is updated immediately.

4. *Initiate reactions:* By clicking on a molecule, an info menu shows up (see figure B.2b). By clicking on "Follow", the camera automatically follows the molecules when it moves. By clicking on a reaction in the shown list, CellUnity uses the selected molecule for the next reaction that occurs of that type.



(a) Simulation Editor



(b) CellUnity Info Menu

Figure B.2: CellUnity Simulation Menus

Bibliography

[1] Mathieu Le Muzic, Julius Parulek, Anne-Kristin Stavrum, and Ivan Viola. Illustrative visualization of molecular reactions using omniscient intelligence and passive agents. *Computer Graphics Forum*, 33(3):141–150, June 2014.

[2] Martin Falk, Michael Klann, Matthias Reuss, and Thomas Ertl. Visualization of signal transduction processes in the crowded environment of the cell. In *Proceedings of the 2009 IEEE Pacific Visualization Symposium*, PACIFICVIS '09, pages 169–176, Washington, DC, USA, 2009. IEEE Computer Society.

[3] Marc Prensky. Computer games and learning: Digital game-based learning. *Handbook of computer game studies*, 18:97–122, 2005.

[4] P. de Heras Ciechomski, M. Klann, R. Mange, and H. Koeppl. From biochemical reaction networks to 3d dynamics in the cell: The zigcell3d modeling, simulation and visualisation framework. In *Biological Data Visualization (BioVis), 2013 IEEE Symposium on*, pages 41–48, Oct 2013.

[5] Molecular Maya Toolkit website. http://www.molecularmovies.com/toolkit/. Accessed: 2014-08-21.

[6] Raluca Mihaela Andrei, Marco Callieri, Maria Francesca Zini, Tiziana Loni, Giuseppe Maraziti, Mike Chen Pan, and Monica Zoppè. Bioblender: A software for intuitive representation of surface properties of biomolecules. *CoRR*, 2010.

[7] Unity Technologies. http://www.unity3d.com/. Accessed: 2014-08-21.

[8] Helen M. Berman, John Westbrook, Zukang Feng, Gary Gilliland, T. N. Bhat, Helge Weissig, Ilya N. Shindyalov, and Philip E. Bourne. The protein data bank. *Nucleic Acids Research*, 28(1):235–242, 2000.

[9] M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, , the rest of the SBML Forum:, A. P. Arkin, B. J. Bornstein, D. Bray, A. Cornish-Bowden, A. A. Cuellar, S. Dronov, E. D. Gilles, M. Ginkel, V. Gor, I. I. Goryanin, W. J. Hedley, T. C. Hodgman, J.-H. Hofmeyr, P. J. Hunter, N. S. Juty, J. L. Kasberger, A. Kremling, U. Kummer, N. Le Novère, L. M. Loew, D. Lucio, P. Mendes, E. Minch, E. D. Mjolsness, Y. Nakayama, M. R. Nelson, P. F. Nielsen, T. Sakurada, J. C. Schaff, B. E. Shapiro, T. S. Shimizu, H. D.

Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, and J. Wang. The systems biology markup language (sbml): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, 2003.

[10] Autodesk Maya. http://www.autodesk.de/products/maya/. Accessed: 2014-08-21.

[11] Graham T. Johnson, Ludovic Autin, David S. Goodsell, Michel F. Sanner, and Arthur J. Olson. epmv embeds molecular modeling into professional animation software environments. *Structure*, 19(3):293–303, 2014.

[12] Blender Online Community. Blender - a 3d modelling and rendering package, http://www.blender.org. Accessed: 2014-08-21.

[13] Maxon Cinema4D. http://www.maxon.net/de/products/cinema-4d-studio.html. Accessed: 2014-08-21.

[14] MCell website. http://mcell.org/. Accessed: 2014-08-21.

[15] Rex A. Kerr, Thomas M. Bartol, Boris Kamubsky, Markus Dittrich, Jenchien Jack Chang, Scott B. Baden, Terrence J. Sejnowski, and Joel R. Stiles. Fast monte carlo simulation methods for biological reaction-diffusion systems in solution and on surfaces. *Nucleic Acids Research*, 2008.

[16] CellBlender website. https://code.google.com/p/cellblender/. Accessed: 2014-08-21.

[17] John E. Stone, Justin Gullingsrud, and Klaus Schulten. A system for interactive molecular dynamics simulation. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, I3D '01, pages 191–194, New York, NY, USA, 2001. ACM.

[18] Theoretical and Computational Biophysics Group, University of Illinois at Urbana-Champaign. http://www.ks.uiuc.edu/research/vmd/. Accessed: 2014-08-22.

[19] PDB ID: 1UBI
Ramage, R. and Green, J. and Muir, T.W. and Ogunjobi, O.M. and Love, S. and Shaw, K. Synthetic, structural and biological studies of the ubiquitin system: the total chemical synthesis of ubiquitin.

[20] Stefan Hoops, Sven Sahle, Ralph Gauges, Christine Lee, Jürgen Pahle, Natalia Simus, Mudita Singhal, Liang Xu, Pedro Mendes, and Ursula Kummer. Copasi—a complex pathway simulator. *Bioinformatics*, 22(24):3067–3074, 2006.

[21] Yoann Kubera, Philippe Mathieu, and Sébastien Picault. Everything can be agent! In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, AAMAS '10, pages 1547–1548, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.

[22] wwPDB. Protein data bank contents guide: Atomic coordinate entry format description version 3.30. ftp://ftp.wwpdb.org/pub/pdb/doc/format_descriptions/Format_v33_A4.pdf.