FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

CellPathway

a Simulation Tool for Illustrative Visualization of Biochemical Networks

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Matthias Reisacher
Matrikelnummer 1125358

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ivan Viola
Mitwirkung: Mathieu Le Muzic

Wien, 25. März 2016

Matthias Reisacher

Ivan Viola

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

CellPathway

a Simulation Tool for Illustrative Visualization of Biochemical Networks

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Matthias Reisacher
Registration Number 1125358

to the Faculty of Informatics

at the TU Wien

Advisor: Ivan Viola
Assistance: Mathieu Le Muzic

Vienna, 25th March, 2016

Matthias Reisacher

Ivan Viola

Erklärung zur Verfassung der Arbeit

Matthias Reisacher

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.

Wien, 25. März 2016

Matthias Reisacher

Acknowledgements

This work was supported through grants from the Vienna Science and Technology Fund (WWTF) through project VRG11-010 and by the EC Marie Curie Career Integration Grant through project PCIG13-GA-2013-618680.

Kurzfassung

In der Biochemie ist Wissen über komplexe biochemische Reaktionsnetzwerke essentiell. Das Gebiet hat in letzter Zeit viel Aufmerksamkeit von Softwareentwicklern bekommen. Daher wurden bereits mehrere Visualisierungsprogramme zur Erzeugung von illustrativen Darstellungen der Struktur einer Zelle erstellt. Jedoch finden pro Sekunde Millionen von chemischen Reaktionen in einer realen Zelle statt die notwendig sind um das Leben eines Organismus zu ermöglichen. Daher ist es notwendig den physiologischen Aspekt in anatomischen Illustrationen einzubinden um Wissen über das Verhalten einer Zelle besser kommunizieren zu können. In dieser These stelle ich ein Reaktionssystem vor, das mit Hilfe eines Algorithmus zu Erkennung von Kollisionen auf atomarer Ebene in der Lage ist, molekulare Interaktionen zu simulieren. Um das Verhalten von Molekülen während des Simulationsprozesses visuell kommunizieren zu können, wurden ein Real-time Glow Effekt in Kombination mit Clipping Objekten implementiert. Da intrazelluläre Prozesse durch mehrere chemische Transformationen durchgeführt werden, wird eine hierarchische Struktur verwendet um den Einfluss einer Reaktion auf die gesamte Simulation darzustellen. Das CellPathway-System verwendet mehrere Optimierungstechniken die das Rendern von großen Datensätzen mit Millionen von Atomen in Echtzeit ermöglichen. Weiters wird das Reaktionssystem direkt auf der GPU durchgeführt damit mehr als 1000 Moleküle in den Simulationsprozess inkludiert werden können. Schlussendlich wurde ein grafisches Benutzerinterface implementiert das erlaubt, Einstellungsparameter während der Simulation interaktiv zu verändern.

Abstract

The molecular knowledge about complex biochemical reaction networks in biotechnology is crucial and has received a lot of attention lately. As a consequence, multiple visualization programs have been already developed to illustrate the anatomy of a cell. However, since a real cell performs millions of reactions every second to sustain live, it is necessary to move from anatomical to physiological illustrations to communicate knowledge about the behavior of a cell more accurately. In this thesis I propose a reaction system including a collision detection algorithm, which is able to work at the level of single atoms, to enable precise simulation of molecular interactions. To visually explain molecular activities during the simulation process, a real-time glow effect in combination with a clipping object have been implemented. Since intracellular processes are performed with a set of chemical transformations, a hierarchical structure is used to illustrate the impact of one reaction on the entire simulation. The CellPathway system integrates acceleration techniques to render large datasets containing millions of atoms in real-time, while the reaction system is processed directly on the GPU to enable simulation with more than 1000 molecules. Furthermore, a graphical user interface has been implemented to allow the user to control parameters during simulation interactively.

Contents

CHAPTER 1

Introduction

The usage of illustrative tools is an established approach to communicate knowledge of complex biochemical processes in cells to a broad audience. In the beginning, illustration artists had to create time consuming handmade animations combined with sophisticated visualization techniques to tell a structured story. The next step was to use software tools to create images showing complex molecular structures. While at the beginning, render processes took hours or days to complete, with increasing processing power it was possible to explore large scenes containing millions of atoms in real-time. However, millions of chemical reactions are performed every second in real cells to allow intercellular communication and to sustain living organisms. To communicate knowledge about complex intracellular processes which keep the cell alive, it is necessary to move from anatomical to physiological illustrations. Therefore, the next logical step is to use molecular reaction systems to simulate large scale reaction networks which are describing the physiology of a cell.

While this area has received a lot of attention lately, many tools to simulate and visualize molecules and reactions inside of a cell have been proposed in the last few years. Lately, particle-based simulators got more popular to imitate a realistic behavior of the molecules. Their general approach is to postpone most of the calculations and operations from the central processing unit to the graphics card through a GPU first approach by, for example, enabling GPU-to-GPU data flow. This is possible due to the modern, freely programmable GPUs. General-purpose GPU programming accelerates the performance of those systems immensely. That enables the simulation and visualization of large-scale scenes containing billions of atoms on an average computer. However, most of those approaches do not take global collision detection into account. This improves performance but leads to visible artifacts during the animation.

My goal is to extend a modern particle-based illustration tool with a basic molecular simulator and a collision detection system. Additionally, a visualization system to improve the user's awareness of biochemical processes and to display reaction networks inside of

a specified area is implemented. Besides, the user should be able to interact with the system to optimize the learning effect. To implement those goals, more calculations per frame have to be executed, which has a significant impact on the performance. Especially the collision detection needs multiple processing steps. For load balancing, the simulation is only executed in a specific part of the scene, whereby the user can determine the size and the position of this area. Therefore, the program's requirements can be scaled down manually by the user to enable the simulation also on weaker computers.

The system is based on the technique proposed by Le Muzic et al. [20]. While almost every aspect of the core visualization techniques are inherited, a simple reaction system has been implemented. The quantitative simulation itself is calculated by the COPASI [18] API and the reaction system is working with an omniscient intelligence while using passive agents for dynamic simulation given by Kubera et al. [19]. To enable a fast and easy change of the simulation system, the user is provided with a simple UI, whose implementation is inspired by the paper of Daniel Gehrer [27].

Current techniques in mesoscale visualization of biochemical processes are including collision detection only partially or they are ignoring it at all for the sake of performance. Tools like ZigCell3D [25] or MegaMol [34] are great for visualization but because the molecular participants do not collide and therefor don't interact with each other beside during a reaction, they are not able to showcase realistic animation of molecular crowding. Furthermore, visible artifacts occur. The main contribution of this work is to implement a three dimensional collision detection system which is able to detect the intersections of two or more objects at the level of single atoms. Additionally, an illustration technique using two adjustable cone-cut-objects in combination with a real-time glow effect is implemented to make complex processes visible even in dense scenes but without losing the impression of depth. Further, a hierarchical structure is used to illustrate intracellular process, by showing the impact one reaction has on the entire simulation. Since this project is based on the work proposed by Le Muzic et al., it also uses the Unity3D [22] engine. Unity is a cross-platform game engine which is also available for free in a limited, but still functional, scope. The provided user interface is also implemented in Unity. Simulated molecules can be downloaded from the public PDB database [23] and afterwards imported through the user interface. A screenshot of a simulation in CellPathway is shown in Figure 1.1.

The rest of the paper is organized as follows: Chapter 2 describes the basics of biochemistry. The related work is analyzed in Chapter 3. Chapter 4 gives an conceptional overview of the whole system. While the methods are outlined in Chapter 5, their implementation is described in detail in Chapter 6. Finally, the results are discussed in Chapter 7.

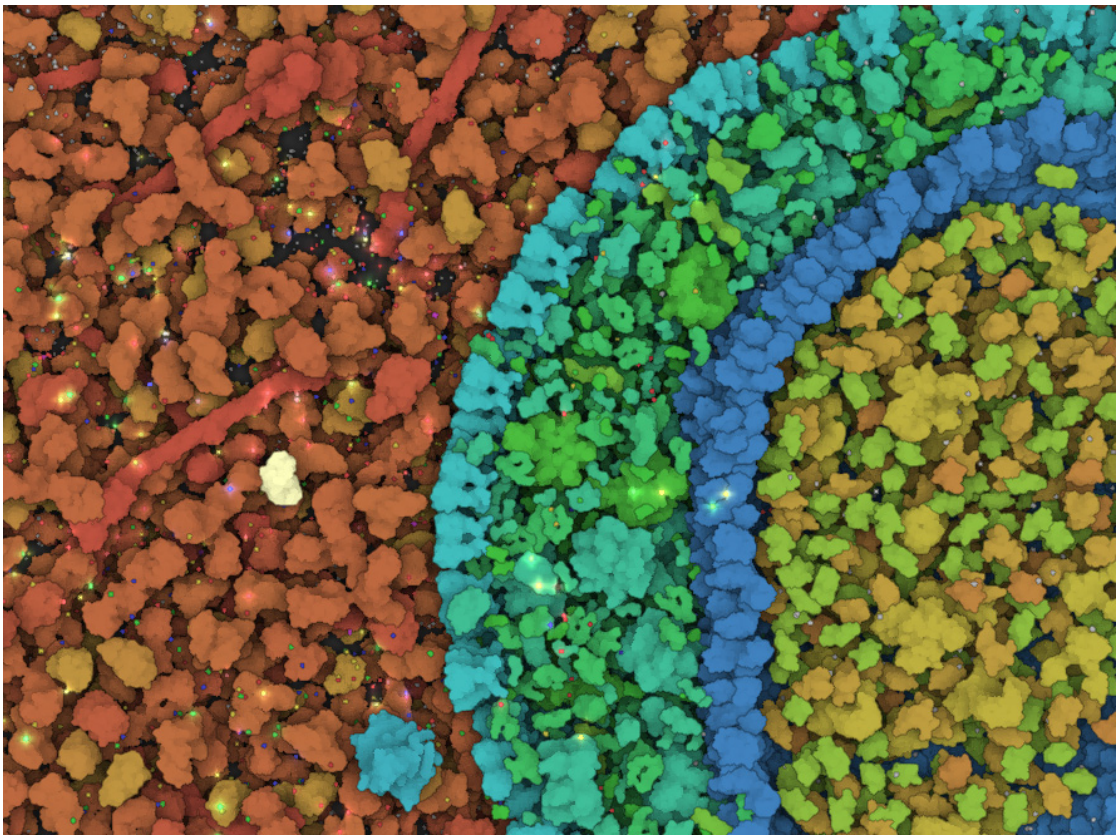


Figure 1.1: A screenshot of a simulation including 1640 reactants with a dataset illustrating a cross-section of human blood serum surrounding HIV virus. The products of currently processed reactions are illustrated with a glowing effect.

CHAPTER 2

Biochemistry 101

The main task of this project is to simulate and visualize biochemical reactions inside of living cells to allow a better understanding of those complex processes. Since proteins are taking part in essentially every structure and activity of life, basic knowledge of their origination process and their functional principles are required.
Biochemistry is a section of organic chemistry and denotes the study of chemical processes in living organisms. By collecting and controlling information about the energy flow during metabolism in cells and intercellular signaling in chemical processes, biochemistry tries to describe the complexity of living organisms. Therefore, the fundamentals of macromolecules, protein synthesis and intrinsic biological networks are described in this chapter.

2.1 Macromolecules

In biology macromolecules refer to a collection of four large molecular types: proteins, lipids, carbohydrates and nucleic acids. Many large molecules in living organisms can be divided into these groups and they have a wide range of responsibilities. For example, proteins are included in intercellular communication and intracellular reactions, some lipids are structural components of cell membrane, carbohydrates are needed to store energy and nucleic acids are responsible to store and transfer genetic information.
While proteins, carbohydrates and nucleic acids are often found as polymers, most lipids are much smaller and form generally monomers. Polymers are long chains consisting of many different small molecules, called monomers, and are created during polymer synthesis. Some examples of monomers, which are important for the synthesis process are glycerol and fatty acids (lipids), amino acids (proteins), nucleotides (nucleic acids) and monosaccharides (carbohydrates).
Since proteins have perhaps the broadest range of functions and are used in CellPathway

to illustrate the basic structure of a cell, the synthesis of proteins is described in more detail in the next section.

2.2 Protein Synthesis

The creation of various types of proteins is one of the most important tasks of a cell because they form structural components of the cell and facilitate essential life functions. Protein synthesis is a complex process which requires the understanding of genetics. Thus, the basics of chromosomes and DNA structures are described as well as the single steps of protein creation.

Chromosomes

To create a new protein, the blueprint of the specific protein type is needed. This genetic information is stored in the DNA, the cell's hereditary material. In the nucleus of each cell, the DNA molecules are packed into thread-like structures called chromosomes. Humans are diploid, having two copies of each chromosome, whereby each cell contains 23 pairs for a total of 46 chromosomes. 23 of those chromosomes originate from the mother and 23 originate form the father. Those chromosomes contain all the generic code in form of double stranded DNA.

DNA

DNA is short for deoxyribonucleic acid and is stored as code made up from four chemical bases, adenine (A), thymine (T), guanine (G) and cytosine (C). The DNA molecule is constructed in a spiral form called double helix, where DNA bases pair up with each other, whereby only the pairs A-T and G-C are possible. The formed units are called base pairs and are also connected to a monosaccharide sugar and a phosphate group. Together, a base, sugar, and phosphate are called nucleotide. The order of the nucleotides determine the information available for building and maintaining an organism. Since every base can only connect with exactly one other base, the sequence of bases on one strand of DNA provides exact information about the sequence of the second strand. For example, if the sequence of one strand is ATGCCGTACGAT, the second strand has to have the structure TACGGCATGCTA. An example of this DNA sequence is shown in Figure 2.1.

From DNA to Protein

The central dogma of molecular biology explains how proteins are created from DNA. As already mentioned above, specific DNA-sequences are the blueprints for the respective proteins. Since each cell possesses only one genome, it has to be protected against damage. Otherwise, the cell would not be able to create new, correctly working molecules any more. Therefore, the DNA is stored in the nucleus, where it is well protected against dangerous influences in the cytoplasm. Cytoplasm is a thick fluid which surrounds the

nucleus and is enclosed by the cell membrane, in which many vital biological reactions take place. Thus, proteins are not created directly from DNA. Instead, an interim stage is carried out. This splits the creation process in two steps: transcription and translation.

During the first step, the DNA is transcribed into ribonucleic acid (RNA). Chemically, RNA resembles DNA with the minor differences that instead of the nucleotide T, the functionally equivalent uracil (U) is used and that RNA consists of only one strand, instead of two.
The development process of a random DNA sequence is shown in Figure 2.1. An RNA message is created by processing the read DNA string. The information contained in the DNA is processed and for each base, the respective counterpart is used. Thus, the RNA message corresponds to the blue DNA string, except for the U-T substitution.
During the transcription preparation phase, various transcription factors are gathered around DNA, each having different tasks and responsibilities. First, the chromosomes are unfolded and one of the transcription factors finds a transcription initiation site on DNA and docks. Another transcription factor breaks the double helical structure of the DNA apart to enable information processing. The enzyme most responsible for the synthesis of RNA is RNA polymerase II (Pol II). A multitude of transcription factors assemble at the transcription initiation site to recruit and properly place Pol II. Together, this large complex of proteins is called the preinitiation complex. This completes the preparation for transcription.

The DNA opens up, Pol II begins to read information and serially transcribes from DNA to RNA. Nucleotides are used to create the RNA sequence, whereby their bases

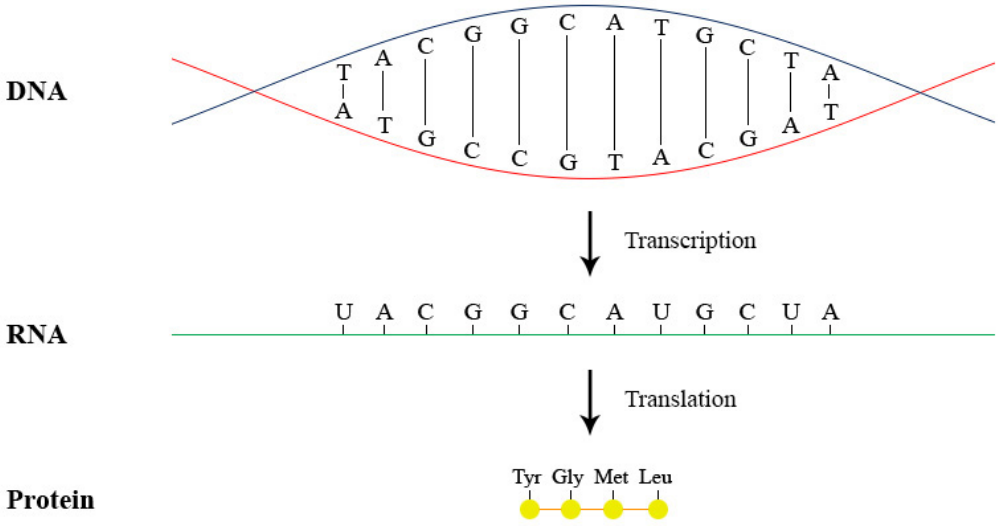


Figure 2.1: An example of a DNA sequence and the respective mRNA and protein structure after transcription and translation [16].

and order are determined by the DNA strand. A simplified representation of this process is shown in Figure 2.2, whereby RNAP is used as an abbreviation for RNA polymerase II. In Figure 2.2(a), multiple transcription factors are docked to the initiation site and are placing RNAP. The separated DNA strings are shown in Figure 2.2(b). RNAP is processing the DNA's information while it is sliding along the bottom string. This way, a RNA copy of the upper string is created.
During the creation of the RNA sequence the already created part is subjected to various processes. Because only information for protein synthesis is needed, the unused areas are eliminated form the RNA and the necessary areas are linked together to form the completed messenger RNA (mRNA). After all information is transcribed, Pol II leaves the DNA and the mRNA is transported outside the nucleus into the cytoplasm.

The second step is the translation of genomic information to create proteins from a mRNA strand. First, mRNA forms a ring to be translated. The ribosome, a complex molecular machine found within all living cells, slides along the mRNA and synthesizes a new protein with amino acids. The information is encoded by units of three mRNA nucleotides. Each set of three bases is called a codon. Individual amino acids are transported by transfer RNA (tRNA) molecules, whereby each tRNA recognizes a certain codon and leaves the corresponding amino acid. So a certain combination of three nucleotides always gives the same amino acid. The protein is folded sterically while being

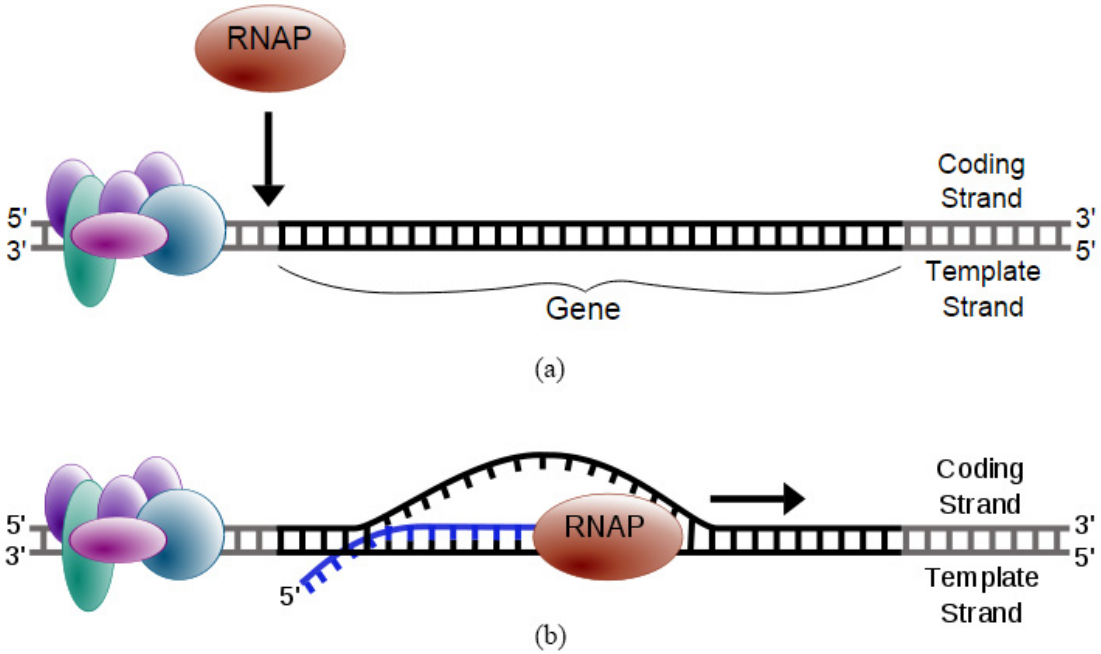


Figure 2.2: Simplified representation of of transcription initiation and elongation. RNAP = RNA polymerase II. [13]

formed. The thus created chain of amino acids, called polypeptide, may be processed by a number of other chemical reactions to form the mature protein.
Figure 2.3 shows the interaction of tRNA molecules and the mRNA string during translation. The ribosome has two sites where the transfer RNA molecules can bind, a peptide site (p-site) and an acceptor site (a-site). Additionally, processed tRNAs are released at the exit site (e-site). In Figure 2.3, a tRNA transporting the amino acid lysine (Lys) is bound on the p-site and another transfer RNA molecule carrying aspartic acid (Asp) has just entered the a-site by a peptide bond. The ribosome now advances a distance of one codon. Thus, the amino acids are joined and the transfer RNA from the p-Site is released at the exit-site. Now a new tRNA, where the anti-codon on the transfer RNA is matching the codon on the messenger RNA, can access the free a-site. Translation occurs concurrently to produce multiple copies of the same protein from one mRNA. The mRNA ring and the ribosome are broken down when they have completed their roll.

2.3 Cellular Functions of Proteins

Proteins are the chief actors within the cell and are traditionally categorized on the basis of their individual actions as enzymes, signaling molecules or structural proteins. They make up half the dry weight of an Escherichia coli cell and can bind to other proteins, so called protein-protein interactions, as well as to smaller molecular substrates. Due to the

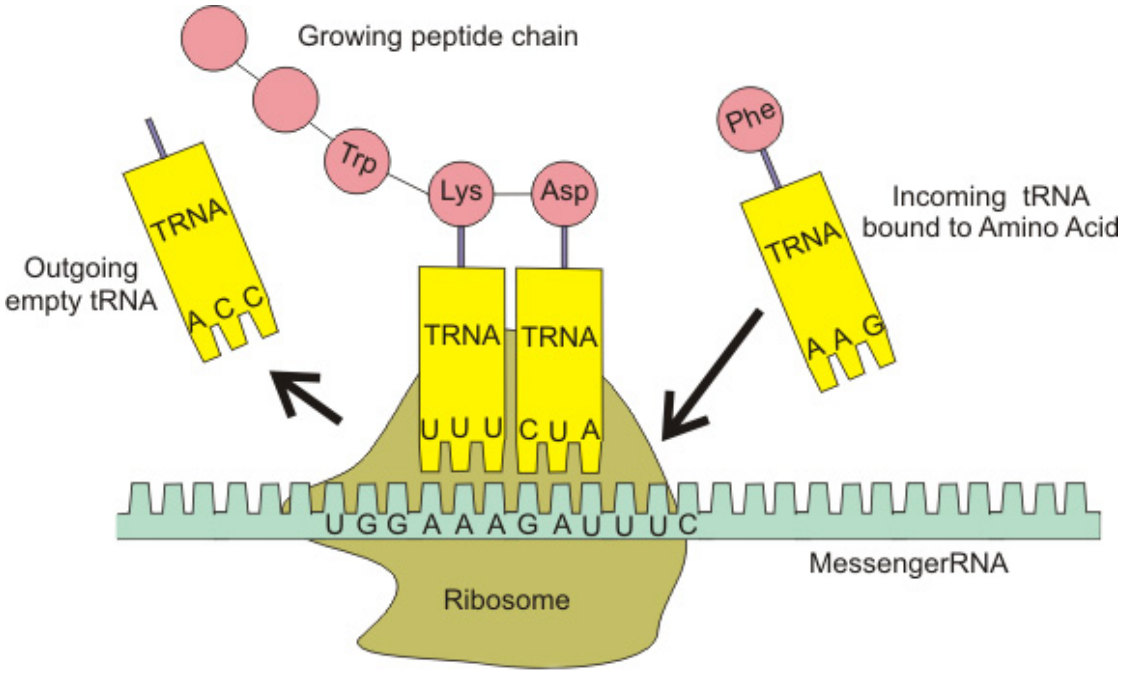


Figure 2.3: The interaction of tRNA and mRNA in protein synthesis. [14]

variety and complexity of interactions between genes, proteins and metabolites, those processes are best represented by various biological networks. In this section, the two major rolls of proteins are described.

Enzymes

Enzymes are macromolecular biological catalysts and they accelerate chemical reactions. Most enzymes are proteins and they speed up the rate of chemical reactions in a cell without being consumed in the reaction. Without them, many biochemical reactions of life would not be controllable or would take too long to sustain complex life. For example, enzymes and catalytic RNA molecules are an important element in transcription and translation of protein synthesis. Even tough they chemically resemble any catalyst, enzymes have a much more specific behavior. Other molecules can affect their activity, which has an impact on the whole cell. While inhibitors decrease the enzymes activity, activators increase their functionality.

Almost all metabolic processes need enzymes to catalyze biochemical reactions. Metabolism is the set of all chemical transformations within a cell, which are involved in maintaining the organism alive. Usually those molecular transformations require intermediate reaction steps to accomplish. This series of stages is called metabolic pathway. In general, a pathway can be anabolic or catabolic. Usually, anabolic reactions require energy input to synthesis new macromolecules while energy is released during the breakdown of complex molecules via catabolic pathways.

Cell Signaling Molecules

Cellular activities are coordinated through a complex communication system. The process of a cell perceiving and responding to their microenvironment with chemical signals, is called cell signaling, whereby proteins are used during both of those cell activities. When a cell releases a chemical stimulus, an extracellular protein like insulin transmits a signal from the cell in which it was synthesized to target cells and alters their activity. Since cells are enclosed within a lipophilic plasma membrane, which has to be crossed by incoming signals, membrane proteins act as receptors to enable communication. These receptors bind signaling molecules and induce biochemical responses in the cell to produce intracellular messengers. A variety of information transfer mechanisms are utilized by complex diffusion, protein-protein interactions and covalent modifications to stimulate sensors and effectors that brings a change in cellular responses. Different signaling pathways are combined and adapted to control a diverse array of cellular processes.

All information described in this chapter were gathered from various sources, including websites, papers and books [1-16].

CHAPTER 3

Related Work

Illustration and simulation of molecular interactions received a lot of attention lately. As a consequence, multiple visualization programs have been already developed. All of them share the same goal, to enable deeper insights in biochemical reactions inside of a cell and to offer a tool which improves the communication of knowledge about these complex processes to a broad audience. In this chapter various tools are examined regarding their simulation capabilities, visualization techniques and area of deployment.

3.1 Agent-based Simulations in Game-like Environments

CellView

cellView [20] is a real-time visualization tool to illustrate the anatomy of a cell in a multiscale approach and is based on the techniques presented by Muzic et al. [17]. The used macromolecular datasets are modeled with cellPACK [21], a tool to generate large molecular structures. The anatomy of a cell is defined by proteins, whereby the structure of individual molecules is specified with a PDB[23] file. Acceleration techniques are used to render scenes containing billions of atoms. The GPU driver overhead is reduced with a hierarchical Z-buffer occlusion culling technique, as well as a twofold level-of-detail approach.
Molecular data is stored directly on the GPU to minimize data transfer. The atoms are rendered via 2D sphere impostors, using the tessellation stage, to decrease the number of vertices per object. To include depth, the individual molecules are processed in a fragment shader to mimic a spherical volume. Furthermore, the dynamic generation of DNA strands on the GPU is supported as well. Although, the main focus of DNA is the visualization aspect and not biomolecular accuracy, even large nucleic acid strands with a double helix structure can be rendered.

cellView is a powerful cross-plattform illustration tool for large molecular landscapes. It has been implemented inside of the Unity3D [22] game engine to reduce the development work-load. But since no molecular dynamics are supported, cellView can not be used to illustrate the physiology of a cell.

ZigCell3D

ZigCell3D [25] is an agent-based simulation tool to visualize a whole cell and its biochemical reactions. To handle the complexity of those chemical processes, the user can set various zoom levels and can use multiple illustration techniques to gain a better understanding of the cell at molecular levels. Since their goal was to support the creation of new drugs, they are using a very accurate simulation system which is adjusted by an interactive and game-like 3D environment. The cellular reactions are modeled with a GUI and are represented as an SBGN [26] network diagram. This enables the user to create complex processes with simple geometric modules like reaction smart boxes and multiple reaction arrows. Additionally, a virtual fluorescence microscope is implemented to benchmark the simulation against real life experimental data. This way the user can add a fluorescence tag to each species to visually highlight their instantiation. The cell itself and the molecules are visualized with complete iso-surface models.

Before the simulation starts the defined reaction network is validated. The simulation can either run with particle-based Brownian dynamics or with the Reaction Diffusion Master Equation (RDME) approach, whereby the latter has less spatial resolution but better performance. However, the rendering module does not use GPU programming and therefore, no real-time processing is possible.

Molecular Reactions using Omniscient Intelligence and Passive Agents

To enable real-time rendering with billions of molecules in the scene, the necessary calculations have to be done in parallel using GPGPU programming. Such a visual explanation tool for difficult biochemical processes inside of a single compartment has been proposed by Le Muzic et al. [17]. A particle-based simulation system is used in combination with passive agents and an omniscient intelligence to enable real-time rendering and interactive exploration of the simulated reactions.
The individual molecules are represented as a three dimensional van der Waals surface with a given radii and the user can follow any one of them. The molecule structure is defined in PDB files [23]. Once an element is brought into the focus, the camera follows the actor and the omniscient intelligence prioritizes the molecule for future reactions. This way the user is able to force specific reactions at a defined location while being able to inspect the whole process. Furthermore, the user is able to build a molecular story by chaining reactions.
The system is designed to simulate multiple reactions simultaneously with a large number of molecules. Therefore the quantitative simulation engine COPASI [18] is used, which is working with biological network files in the SBML [28] format. To reduce the calculation

time during simulation, the necessary computations are executed in parallel on the GPU, using CUDA. Additionally, a simple and very limited collision detection is implemented which is only applied on reaction partners. The reaction is triggered once a collision is detected. To create the impression of chaotic behavior, the molecular movement during a reaction is created by interpolating direct motion with Brownian motion.
Because it is not necessary to draw every molecule in full detail, a molecular Level-of-Detail technique is implemented to decrease the performance requirements during rendering. The number of rendered atoms depends on the distance from the molecule to the camera. To keep the basic structure of the molecule constant, the radii of the remaining atoms are scaled accordingly.

Through the interactive visualization and story-telling structure, the user is able to generate illustrative visualizations of biochemical processes. However, due to the trivial collision detection, animation of realistic trajectories of crowded molecules is not possible.

CellUnity

CellUnity [27] is a molecular simulation and visualization tool based on the techniques proposed by Le Muzic et al. [17] and uses a complete collision detection system to enable realistic trajectories. The cell is mimicked by a spherical compartment and contains all participants of the simulation. Molecular structures are defined through a PDB [23] file and are represented with the van der Waals surface model. To add new molecules, the specific files can either be imported or directly downloaded from the public PDB database.
The simulation is implemented entirely on the CPU and is therefore only deployable for smaller scenes containing a few hundred molecules. An omniscient intelligence is tightly coupled to the COPASI API [18] which is used to simulate biochemical processes. Molecules are based on the passive agents principle and are moved per frame through the compartment. The project is implemented inside of the Unity3D [22] framework. This allows to use Unity's built-in physics engine for collision detection. A reaction is processed when a physical connection is established between all participants. To emulate the impression of chaotic behavior inside of the compartment, Brownian motion is applied partially on all molecules.
Reactions are shown in a story-telling manner, while the user has full control over their spatial locations. The reactions can either be executed randomly or the user can trigger a specific reaction by selecting a molecule in the scene, whereby the camera follows the selected actor. Additionally, the visualization and the simulation speed can be changed by the user to avoid a cluttered visualization in dense scenes.

CellUnity is an easy to use illustration tool for biochemical reactions. The fully applied collision detection enables realistic simulation while complex chemical processes are told in a story-telling structure. However, since the project is implemented entirely on the CPU, only small scenes containing a few hundred molecules can be used for simulation.

3.2 Monte Carlo based Simulation Systems

MCell

MCell (Monte Carlo cell) [29] is a program to simulate movements and reactions of molecules within and between cells. The simulation can contain multiple compartments and supports molecular interactions of multiple objects, while using realistic 3-D models of synaptic microphysiology and specialized Monte Carlo algorithms. Molecules are created form polygonal meshes. They are described together with the general simulation preferences with the MCell Model Description Language (MDL) and are saved in human-readable text files. Since meshes have to be modeled by hand, this tool is not suitable for high density scenes because of the immense initial effort. To increase the execution speed, spatial partitions and subvolumes are used for collision detection between ligand molecules and mesh elements. MCell has no initial support of a graphical user interface and is executed instead form the command line.
All decisions made during the simulation process depend on a seed value. This value has to be set by the user at the beginning, cannot be changed afterwards and is used to generate random numbers. As a consequence, no interactive visual steering is supported and the user can only follow the previously adjusted simulation. This complicates a storytelling approach extremely, because it is very hard for a user to follow a single molecule.
Another problem are the fixed timesteps during simulation, which simplifies the program design considerably but also leads to several computation and visualization problems. An optimization of the algorithm is proposed by Kerr et al. [30] which extends the system by a dynamic timestep algorithm and some performance optimizations.

CellBlender [31] is another extension of MCell and enables an easier and faster way to model and edit the molecule designs of a simulation. It is an addon for Blender 2.6-2.7x [32] that allows users to create cells and their inner structure with a professional, free and open-source 3D computer graphics software. While the simulation is still based on MCell, the results are visualized in Blender. Although, the simulation settings can be changed directly in Blender, still no interactive storytelling approach can be used to present the outcomes.

Illustrative Timelapse

Another tool which is attending to the problem of fixed timesteps during simulation was proposed by Le Muzic et al. [33]. Illustrative timelapse is a system to create illustrative mesoscale visualizations automatically while the user is being able to move freely through the scene and to explore the simulation on multiple temporal scales. Since many biochemical reactions take only a few nanoseconds to accomplish, different visualization techniques have to be used to allow the user to follow individual elements and their story. Therefore, a combination of temporal zooming and visual abstraction is used. This allows the user to change the temporal resolution interactively, which also has a direct impact on the particle speed. Additionally, the participants of important

reactions are emphasized for a limited duration and a lens effect is implemented to preserve a more realistic way of molecule motion.

MCell is used to model and simulate biochemical processes while the visualization part is implemented in Unity3D. Because computations in MCell are quite slow, real-time rendering was implemented based on the technique proposed by Le Muzic et al. [17].

Illustrative timelapse is a cross-platform simulation tool with the focus on multi-scale temporal illustrative visualization techniques. Even though no collision detection between molecules is taken into account, the user is still able to follow a story.

3.3 Molecular Rendering Systems

MegaMol

MegaMol [34] is a visualization software for large generic particle datasets. Their goal was to create an easily extensible and therefore very flexible visualization software which is able to render millions of particles and can be adopted to fit special requirements in various areas. The visualizations in general are defined through module graphs generated with the MegaMol-Configurator. Due to the flexible architecture and data flow, interactive image-based post-processing steps can be implemented and added to the project. By default, the OpenGL graphics API is used for rendering, but the project can easily be adopted to use Direct3D instead. MegaMol does not contain a user interface, but several specialized front ends exist. Because the core library is published as a slim C API, it can easily be included in other languages like C# and Java. Therefore, multiple UI technologies like Windows Forms [35] or Qt [36] can be used.

MegaMol is a powerful visualization tool which supports multiple file formats like PDB [23] to define the molecular structure. The system is freely available, open source and runs on Microsoft Windows and Linux, both in 32 bit and 64 bit.

Illustrative Molecular Rendering

Hermosilla et al. [37] proposed a illustrative visualization tool for all-atom simulations, which is able to render even very large molecules with more than a million atoms in real-time. The goal of the work was to enable accurate simulation of the reactions between a complex protein and a simpler molecule. This especially helps in Pharmacology to understand protein interactions to develop improved drugs for healthcare.
To implement their objectives they had to use accurate data for the molecule's behavior. Therefore, all participants are moving along actual calculated positions during the molecular dynamics simulation. Through the simulation an occupancy pyramid, a hierarchical data structure, is generated on-the-fly on the GPU and is used to create high and low frequency ambient occlusion shadows as well as halos for the highlight visualization and is stored as a three-dimensional texture with different mipmap levels. To help the user identify the ligand and to seperate it form the protein, an object-space

halo surrounds either the protein or the ligand. Furthermore, temporal halos capture the evolution of the interaction. This way the ligand's positions in the last 10 seconds are highlighted and fade out gradually. Additionally, the user can stop the simulation at any time and move the camera around in the scene to get a better view on interesting areas during the reaction.
The software supports three different pure structural molecular representation modes. The user can choose between the van der Waals surface, ball-and-stick and the more abstract licorice illustration.

At about the same time Staib et al. [38] presented an extension of their system using the same approach but with a different implementation. However, their system does not include visual effects like halo rings and only support the van der Waals surface representation.

This software gives very accurate insights into the interaction of a protein and a drug-molecule. An all-atom simulation can be rendered in real-time for even very large molecule structures. However, the system is unsuitable for large scale scenes with many molecules and multiple simultaneous reaction.

CHAPTER 4

Conceptual Overview

Presented algorithm consists of three basic elements, which are shown in Figure 4.1. The first element, a two-parted data structure is used to store object properties and has been implemented to enable efficient data access and to minimize the needed amount of storage. The simulation system is responsible to move reactants, process collisions and to perform molecular reactions. The last part, the visualization system, uses three techniques to communicate the proceedings during the simulation to the user. A conceptual overview of CellPathway and the three main elements are described in this chapter.

4.1 System Overview

Since the manipulation of the properties from individual objects per frame is crucial for the functionality of both, the simulation and the visualization, the system overview is given in respect of the data change during runtime. The three main parts of the abstracted program structure regarding the information flow are shown in Figure 4.1: data initiation, simulation and visualization.

Since this project is based on the visualization system proposed by Le Muzic et al. [20], the used data structure is inherited and extended. Instead of using spherical meshes to represent molecules, the data is stored in a generated texture buffer and only one vertex per molecule is used as an input during rendering, while the object is constructed on the fly using the tessellation stage [20].
However, Data objects are stored in a two-parted data structure to enable efficient data access and to minimize the needed amount of storage. Passive data contains all information needed to render molecules and is uploaded to the GPU when a scene is loaded and when the reactants are placed. Structural details of the molecular types, such as the atom count, the position of single atoms and the color, as well as the position and rotation of every single molecule, are stored as passive data. On the other hand, active data stores additional properties for every participant of the simulation process, such as

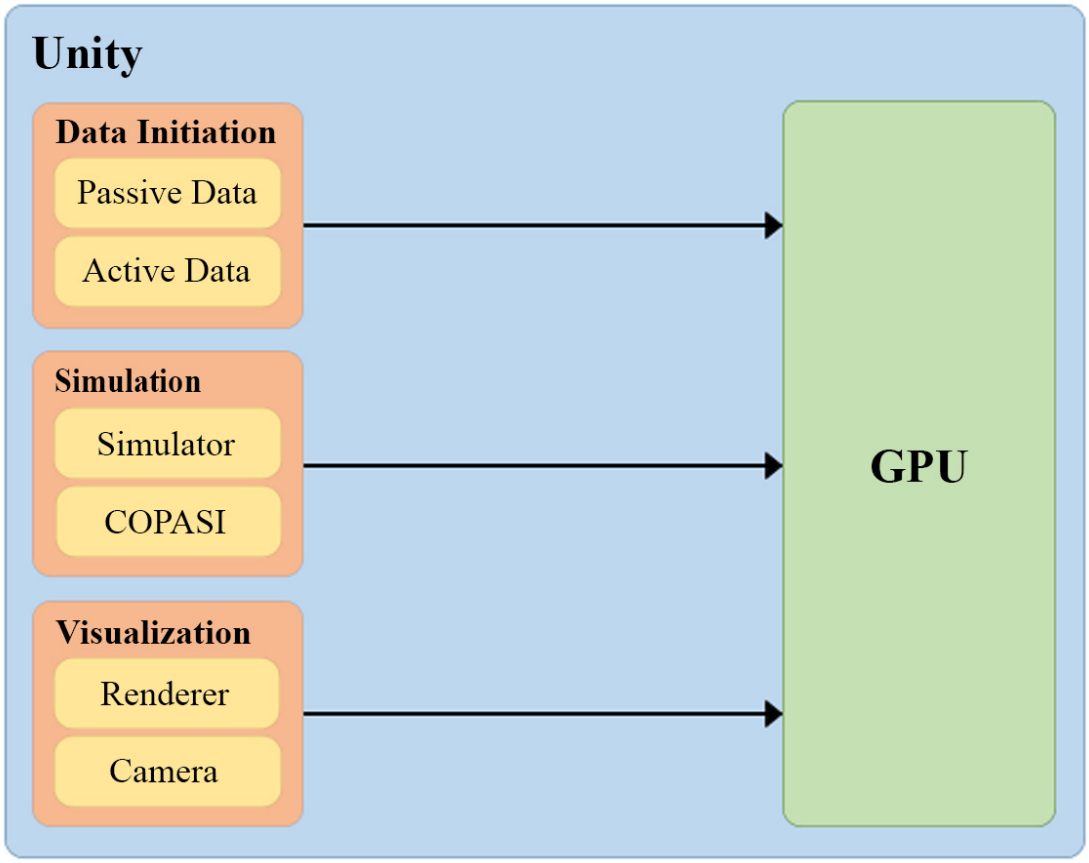


Figure 4.1: The abstracted program structure regarding the information flow with the three main parts: data initiation, simulation and visualization.

reference IDs and flags to manage the mutual coherence of individual objects. Since this information is only needed during simulation, active data is gathered and uploaded to the GPU when a simulation is started. A detailed description of active data is given in Section 6.1.2.

While the data initiation, which corresponds to an object creation and initialization step, is only executed when needed, simulation and visualization are done every frame. To increase the performance and to enable a large scale simulation, the CPU to GPU traffic has to be minimized. Therefore, the reaction system and the collision detection is processed directly on the GPU. This way, no data transfer per frame between the processing units is needed.
To simulate biochemical networks and their dynamics, the COPASI API [18] is integrated and runs in parallel on the CPU. While COPASI indicates, which reactions in which quantity should be initiated, the simulator is responsible to execute programs on the GPU, so called shaders, to apply the required actions on the objects. Additionally,

the simulator moves all simulation participants each frame and executes the collision detection system.

Since no meshes are used, the render pipeline in Unity3D returns an empty scene. Instead the rendering is done in a post-processing step, as described by Le Muzic et al. [20]. The renderer is responsible to synthesise the scene and to visualize the three illustrative effects, real-time glow, cone clipping and reaction tree. A detailed description of the real-time glow technique is given in Chapter 5.4. With cone clipping, the user is able to remove occluding objects between the camera and the simulation center. Reaction tree is a technique to visualize biological networks in cells, where based on reactions, an interactive tree structure is created. This way, the temporal impact of one reaction on the entire simulation is illustrated.

4.2 Simulation

The simulation process is one of the main parts of this work and consists of a molecular reaction system and a collision detection algorithm. To control the simulation process and to enable the user to interact with the system, a graphical user interface has been implemented as well, which is based on the GUI proposed in CellUnity [27]. The molecular type and the quantity of the reactants, as well as the processes reactions are defined with the UI. Structural information about molecules are stored in PDB [23] files and can be imported. Every reaction can be linked with a protein. Thus, the reactants need a random protein to perform the specific reaction.

The molecular reaction system is based on the technique proposed by Le Muzic et al. [17]. While the individual molecules who are participating in the simulation process are implemented as passive agents, an omniscient intelligence (OI) is used to control molecular interactions. Passive agents are unable to start reactions autonomously, instead they can only receive reaction orders from an OI, which is tightly coupled with the quantitative simulation [17]. The system uses the COPASI API [18] as simulation engine, which runs in parallel [17].

To minimize the CPU to GPU traffic, simulation steps manipulating object data are executed on the GPU using the DirectCompute technology [43]. While the OI is responsible to initiate new reactions, the simulator executes those initiation orders by manipulating object data accordingly. Furthermore, the simulator moves all reactants which are included in the simulation system and processes collision between objects. Since this requires to execute many calculations per frame, the workload on the GPU is significant. To enable simulation in scenes with thousands of reactants, the reaction system does not include all molecules. Instead, the simulation area is reduced to a spherical compartment. Only reactants inside of the compartment are included in the simulation process, while the others are moved by random walk. While the center of the sphere is set by selecting an arbitrary protein inside of the scene, the compartment radius is set in the UI. This way, the user can adjust the workload of the simulation to the available resources.

4.3 Visualization

Illustrative tools are used to communicate knowledge of complex biochemical processes in cells. With various visualization techniques, the user is able to get insights in molecular interactions and biological networks. While the technique to represent molecules in a level-of-detail manner proposed by Le Muzic et al.[17] is inherited, three additional visualization techniques, called real-time glow, cone clipping and reaction tree, are implemented to improve the way the user perceives information of biochemical processes.

Since the reactions are distributed throughout the compartment and can occur simultaneously, it is difficult for the user to realize when and where a reaction is completed. The purpose of the glow effect is to draw the user's attention to areas where reactions are processed during the simulation. Every created product of a completed reaction is highlighted for approximately one second. In addition, proteins, which are included in a reaction, are highlighted as well. This enables the user to recognize ongoing reactions easily, while observing the simulation compartment. The real-time glow technique is described in more detail in Section 5.4.

When illustrating dense scenes with millions of atoms placed near each other, reaction processes are easily covered by larger protein structures. With the cone clipping method, the user is able to remove disturbing molecular structures to get a better view of the ongoing reactions and molecular interactions during simulation. Since the reactants, consisting of only a few atoms, are much smaller than the proteins, they are not removed. Proteins, which are located between the camera and the center of the simulation compartment are clipped away. To allow the user to change the amount of clipped objects, the cone angle can be set interactively to a value between 1 and 89 degrees.
While increasing the visibility of simulation participants is the main goal of this visualization technique, the three dimensional spatial depth impression should be retained. Therefore, a semi-transparent area is used to create a continuous transition between the clipped area inside of the cone and the shown objects outside of it. Thus, a second cone is implemented and placed at the same position as the clipping cone, whereby the angle of the second cone is twice as large as the angle of the clipping cone. Objects located between the inner and outer cone are represented partially transparent. To create a continuous transition between the clipped area and the opaque objects, the amount of transparency for a specific object depends on the location. While molecules located next to opaque objects are rendered with less transparency, the value increases when located closer to the clipping cone. Although, semi-transparency is faster to calculate than a real transparent effect, artifacts can occur when no opaque objects are located behind a transparent molecule. In this case, the objects color is interpolated with the background color black and objects with a larger alpha value are darker than more opaque objects.

In this project, a hierarchical structure is used to illustrate biological networks created by complex interactions between different molecules and proteins in a particular time span. Starting with only one reaction, a tree structure is build by illustrating the path of the reaction products with lines. When those molecules are included in another reaction,

a node at the location of the reaction is included in the structure, while the outgoing branches are connected with the new products. Therefore, the tree structure is growing over time, showing the influence of the starting reaction on the whole simulation.
To increase the visibility of the hierarchical structure, the user can switch between two representations. By default, the color of individual branches corresponds to the color of the respective molecule. Since the molecule's color depends on the molecular type, the user is able to determine, which reactions where performed. Alternatively, an arbitrary line color can be used to optimize the contrast between the reaction tree and the scene.

CHAPTER 5

Methodology

During the creation of this project, several algorithms and techniques have been used to create visual effects and to enable molecular interactions. Since both, the visualization and the simulation are running in real-time, performance is an important factor. In particular, because simulation of more than 1000 molecules is desired to run at least with 30 frames per second. But components like the collision detection system, which is working at the level of single atoms, require many calculations per frame. Additionally, information such as molecular interactions, biological networks and chemical reactions are illustrated. Therefore, several methods have been implemented to reduce the number of overall calculations and to create visualization effects in a dynamic system.
In this chapter, four implemented techniques used during simulation and visualization are introduced.

5.1 Spatial Subdivision

Spatial subdivision is an approach where objects in a three dimensional space are ordered by their position. This improves the speed of further processes, which are using the spatial relation of individual objects, immensely. The space is partitioned in a uniform grid, such that a cell is at least as large as the largest object [40].

During the first step, the number of cells has to be calculated, which only has to be executed once at the beginning. Since the cell size corresponds to the size of the largest object, the number of cells is determined by dividing the measurements of the space by the size of one cell. Afterwards, the objects can be sorted by using the counting sort algorithm described in section 5.1. Additionally, an array called Bin-Counter is used to keep track of the number of objects inside of every single cell. The Bin-Counter in combination with the list of ordered objects allows to identify all objects contained in a specific cell by the cell index.

5.2 Counting Sort

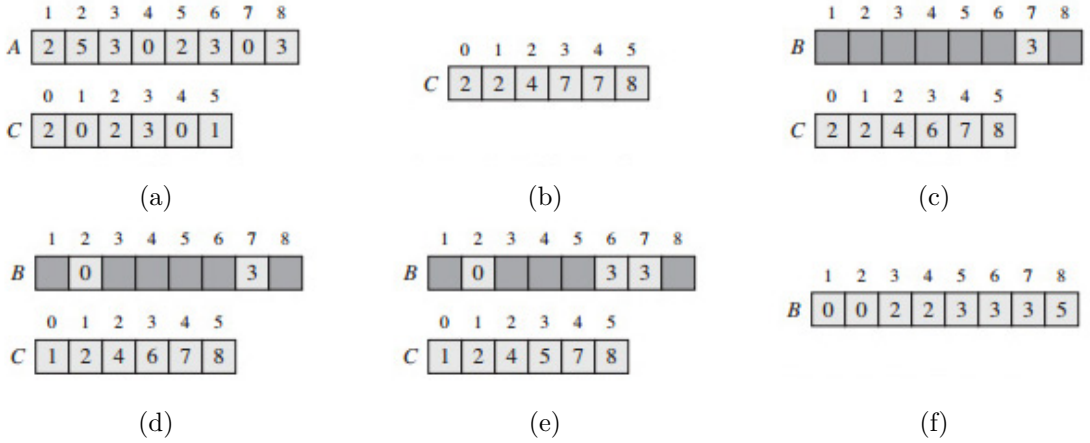


Figure 5.1: The operation of the counting sort algorithm on an input array A[1-8], where each element of A is a non-negative integer no larger than $k = 5$ [39].

With spatial subdivision it is possible to find all molecules, which are assigned to a specific cell. To enable efficient data access all objects are sorted by their cell index, using the counting sort algorithm.
Counting sort is a method for sorting a collection of objects based on keys in a specific range. Each of the n input elements is a non-negative integer with a value between 0 and k. The algorithm has a running time of $\Theta(k)$ and is effective when the number of objects is significantly greater than the number of possible input values.

The sorting process is split into multiple steps. During the procedure, three arrays are used to store information. Figure 5.1 illustrates an example of the algorithm's procedure on 8 unsorted integer values. Array A contains the input values while the resulting values, sorted in ascending order, are stored in array B. Both have the same length $n = 8$. C is an auxiliary array to store temporary data, is initialized with 0 and has the length $k = 5$. The first step is shown in Figure 5.1(a), where the number of appearances of each value is counted and stored in the auxiliary array. Then subsequently, the single values in C are summed up so that each index represents the last position of each value in the sorted array. The resulting content is shown in Figure 5.1(b).
During the last step, each element in A is placed into its correct position in the output array B [39]. It is started with the last element in A, in this example with the value 3 stored at position 8. The element's new position is contained in C, while the element's value corresponds with the specific index. Since $C[3] = 7$, the element is copied into B at index 7. Afterwards, the counter value in $C[3]$ is decreased by one. The resulting array content is shown in Figure 5.1(c). The procedure of the last step is repeated with the next element in A, until all elements are processed.
Figure 5.1(d) and (e) shows the results after processing the next to values, while the sorted input values are shown in (f).

An important property of counting sort is that it is stable: numbers with the same value appear in the output array in the same order as they do in the input array [39].

5.3 Fast Fixed-Radius Nearest Neighbors

Fast fixed-radius nearest neighbors (NNS) is an algorithm to find all objects inside of a sphere with the radius R, centered at the position of a specific object. Since the time complexity of a brute force attempt to find all neighbors of all objects is $O = (n^2)$, the spatial partitioning method, described in Section 5.2, is applied first. To minimize the number of cells who are overlapping with the sphere without having too many objects per cell, an additional requirement is established, which states that the minimum cell size during spacial partitioning has to be at least as large as the radius R. This way, only objects in neighboring cells have to be searched, which reduces the average complexity to $O(n * log(n))$ [41]. Those objects can easily be found by combining the spatially sorted objects and the Bin-Counter values.

A two dimensional example of the fast fixed-radius NNS including spatial partitioning is shown in Figure 5.2. As can be seen, no matter where the selected object is located inside of the cell, only the adjacent cells have to be searched to find the neighbors. When the object's position inside of the cell is ignored, 9 cells and their corresponding objects have to be checked. This applies to the three dimensional approach also, in which case 27 instead of 9 cells have to be tested. This reduces the number of overall tests immensely.

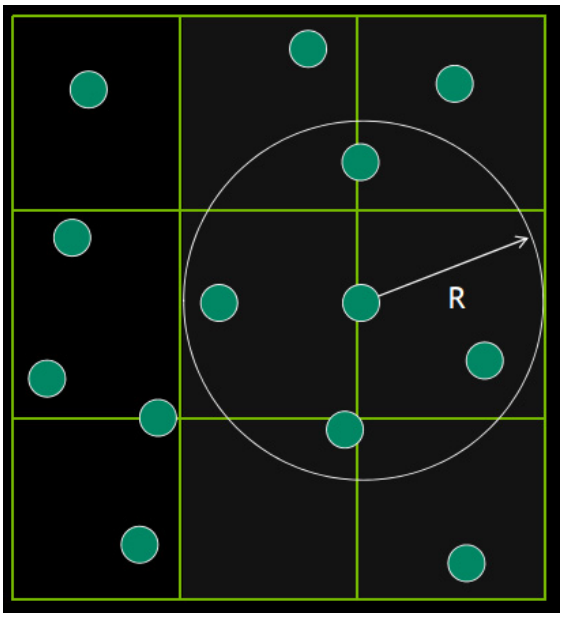


Figure 5.2: Shows an example of the fast fixed-radius NNS method in a 2D space with spatial partitioning [41].

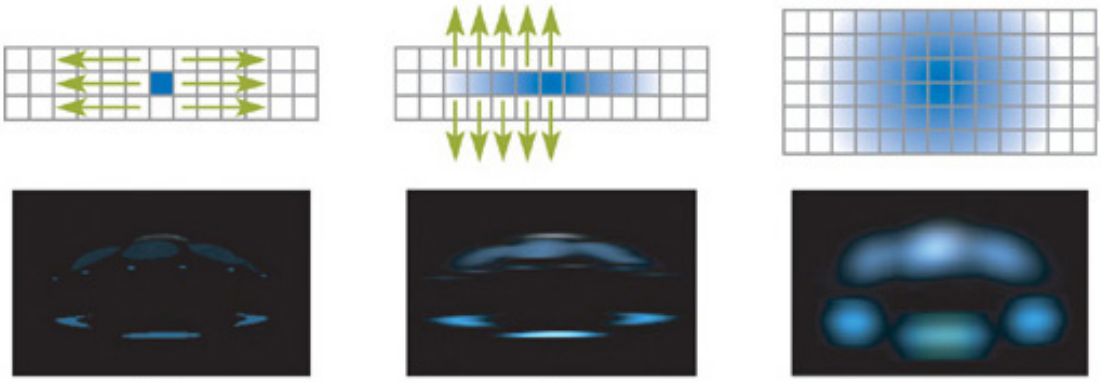
5.4 Real-Time Glow

Glow, also called bloom, is an effect where an artifact of a real-world camera is reproduced to create the illusion of extremely bright areas. There are several different approaches to create a glow effect. However, in this project a post-processing method of a 2D rendering of the scene is used [42].

The rendered image is passed to a shader and processed on the GPU, where all objects which should be highlighted are copied in a separate texture. This way, when a glowing object is covered fully or partially by another object, only the visible areas are copied into the glow texture.
In the second step, the glow effect is created by blurring the rendered object. This is done with a compute shader using a two-step operation called a *separable convolution* [42]. The technique of separable convolution splits the two-dimensional convolution kernel in two separate one-dimensional convolutions, one in each axis, which greatly reduces the computation costs [42]. To perform the blurring convolution operation, the color of each pixel in the separated texture is spread amongst the local neighborhood of pixels. This is done first along the horizontal axis and then along the vertical axis.
The blurring process is shown in Figure 5.3. While the original image is shown in Figure 5.3(a), the result of horizontal blurring is shown in Figure 5.3(b). The color of a single pixel is spread out on the neighboring pixels to create a continuous transition. This effect is enhanced by repeating the operation along the vertical axis. The final result is shown in Figure 5.3(c).

Finally, the original texture and the blurred texture are combined. This is done by simply summing up the two pixel values on every position.

The range of the glow effect can be increased or decreased by changing the convolution kernel size respectively. On the other hand, the glow intensity depends on the kernel weight. To create a bright glow effect, a convolution kernel with a weight larger than one is used.



(a) (b) (c)

Figure 5.3: The Two-Step Separable Approach for Creating Blurs Efficiently [42].

CHAPTER 6

Implementation

CellPathway is implemented seamlessly in the Unity3D game engine. Since Unity3D supports advanced GPU programming, most parts of the simulation system as well as the entire collision detection algorithm, have been implemented with compute shader programs. This enables to manipulate object data directly on the GPU, which eliminates data transfer from the CPU to the GPU per frame. In this chapter, a detailed description of the implementation of the simulation system and the collision detection algorithm is given.

6.1 Simulation

The implemented simulation system consists of two parts: preparation and simulation step. Since object data is stored exclusively in texture buffers on the graphics card, all parts of the simulation system which are manipulating object data are implemented with general-purpose GPU programming, using multiple compute shaders. In this section, both parts are explained in detail.

6.1.1 Preparation

The preparation phase of the simulation only needs to be executed once, at the beginning of every simulation. The main purpose of this stage is to identify all participants of the simulation. Since the information about the objects in the scene is located on the GPU, the entire preparation phase is executed with shaders. The simulation system distinguishes between proteins and reactants. While reactants are consumed in the course of a reaction, proteins are used to illustrate the structure of a cell and are not changed by a chemical reaction. Both, the reactants and the proteins are checked, if they are inside of the compartment sphere or not. A molecule is considered to be inside of the compartment, when part of a molecule's bounding sphere intersects with the

compartment sphere. Because reactants and proteins have to be handled differently, the simulation preparation is implemented in two compute shader kernels, which are executed consecutively.

For every reactant inside of the compartment, a molecule object, consisting of flags and reference IDs to store information about coherence between objects, is created. With the reference IDs a molecule can be assigned to a specific reaction, a collision and the reaction tree structure. On the other hand, the protein objects, which are created for every protein participating in the simulation, contain only a reference to the passive protein data. Additionally, a counter keeps track of the number of reactants and proteins inside of the compartment, as well as the number of reactants per molecular type. The determined values are sent subsequently to the CPU. This way, the exact number of needed work groups of each compute shader kernel call during a simulation step can be specified.

6.1.2 Simulation Step

A simulation step contains all processes which are done once per frame during simulation. Since the effectiveness of a single step is crucial for the overall performance, special methods are used to increase the processing speed. This section describes the single procedures of a simulation step.

Reaction Generation

Reactions are initiated by the omniscient intelligence. In regular time intervals, the OI calls the simulator to initiate multiple reactions. The length of the time intervals can be set by the user. Since each reaction has to be processed separately on the GPU, the number of created reactions per frame is limited to 20 to increase the performance. If more reactions are supposed to be initiated, the remaining reactions are queued. The simulator checks every frame if the reaction limit is already reached and initiates queued reactions if possible.

For every initiated reaction, a shader searches for appropriate reactants which are not already included in an open reaction. The selection of molecules is partially randomized. While the first reactant is picked randomly, all other molecules of the specific reaction are selected by their distance to the first molecule. Only the molecules closest to the first reactant are assigned to the reaction. When all required molecules could be found, a reaction object is created. A reaction object contains the following information:

- *Reactants and products:* The reaction stores the number of needed reactants and products, the molecular type of the products and a reference to the included reactants. This allows bidirectional access between the molecules and the reactions, which is needed in the further simulation steps.
- *Position:* The position where the reaction will take place is needed during the molecular movement step and when the products are created. The position is

calculated by adding up the selected reactant's position and dividing the sum by the number of reactants. Additionally, the reaction position is controlled with the collision detection system to avoid a location inside of an arbitrary protein.

- *Reaction protein:* The user is able to assign proteins to a specific reaction type. This way, every reaction of this type needs an arbitrary protein to be carried out. Therefore, the reaction object stores a reference to a randomly chosen protein inside of the compartment if a protein has to be included. Additionally, the reaction position is moved to the center of the respective protein.

- *Reaction Tree:* When a reaction tree is initialized or when one or more reactants of the reaction are already part of the reaction tree, a flag is set in the reaction object. After the reaction is executed and the flag is set, the products are added to the visualization tree.

So called critical reactions can be regarded as a special case and additional steps are needed to prevent errors. All reaction, having more products than reactants, are called critical reactions. Normally, during the creation of products, the reactants consumed by the reaction are simply overwritten. However, the excess products in critical reactions have to be appended to the compute buffer. Therefore, the number of reactants inside of the compartment has to be increased when a critical reaction is processed. Otherwise, the created products are ignored in the movement and reaction system.

Movement and Collision

Molecules in the scene are moved in three steps. First, Brownian motion is applied to all reactants and proteins. This is important to create the impression of frantic interactions and chaos in the scene. While this step is done independently of the simulation, the next steps are part of the reaction system.

Since chaotic interactions are also desirable during reactions, the new molecule locations calculated previously are further processed. Reactants included in a reaction have to reach a specific location. Otherwise, no reaction can be carried out. Therefore, the actual trajectory is calculated by blending the reaction steering force with Brownian motion. This prevents linear pathways and enables the simulation of molecular trajectories more realistically. Since a collision detection system is included in the movement procedure, the calculated locations are stored as movement vectors to enable further processing. Whenever a reactant's new position is located outside of the compartment sphere, the movement vector is reversed instead.

The actual movement of the reactants inside of the compartment is done in the third step. In this step, collision detection is also included. First, the molecule's position, rotation and movement vector are passed to the collision detection algorithm, to find a collision with a protein. Since proteins are much larger than the reactants, a collision has no influence on their movement. On the other hand, the movement vector of a

reactant is shorten in case of a collision. When a reactant needs to enter a specific protein to perform a reaction, collision between the protein and the reactant is ignored. The collision detection system is described in Section 6.2 in more detail.
Afterwards, collision between reactants is checked. Since reactants can interfere with each other, a collision affects both participants. Therefore, a collision object is created, which stores the two reactants of a collision, their movement vectors and the ratio of the movement vectors, calculated by the collision detection method. When no collision was found, the location of the molecule is updated.
In a further step, the collision objects are processed and the reactants are moved until they collide. When the collided molecules are assigned to the same reaction, a flag is set in both molecule objects to indicate that they are ready for execution. When all molecules included in a reaction are ready, a perform-reaction object is created.

Reaction Execution

The last step of the simulation system is the reaction execution step, where perform-reaction objects are processed. Since every reaction stores information about the participants, the involved reactants can be accessed easily. During the first step, the reactants are deleted. To minimize the number of molecules stored in a compute buffer, the buffer positions of deleted molecules are saved. This way, whenever a product is created, deleted molecules can be overwritten.
In the second step, new products are created. The collision detection algorithm is used to find a free position in the scene, as close as possible at the reaction position, for each product. Additionally, a flag to enable the glow visualization effect is set for each product and the included protein.

6.2 Collision Detection

Collision detection among many 3D objects is an important component of physics simulation and most efficient implementations are structured in a two-phase approach: a *broad phase* followed by a *narrow phase* [40]. During the broad phase, collision between objects is determined by using basic bounding volumes, like minimal bounding spheres. Those tests are fast and cheap to calculate. However, since approximated bounding volumes are too imprecise for more complex shapes, errors occur in the collision tests. Therefore, further processing with the precise shape of objects is necessary, which is done in the narrow phase. Since those tests are much more costly and require more calculations per collision test, they are only executed for potentially colliding objects, found during the broad phase.

In this project, collisions between molecules are calculated. Since each molecule consists of multiple atoms, which are represented as spheres, and each object is enclosed by a spherical bounding volume, the same collision algorithm is used during the broad and narrow phase. The only difference between the two phases is that the collision test is executed once per object during the broad phase and once per atom during the narrow

phase.
Due to the difference in size between reactants and proteins, two steps have to be executed during collision detection. Since the number of reactants inside of the simulation compartment is much larger than the number of included proteins, the fast fixed-radius nearest neighbor algorithm combined with the spatial subdivision technique, both introduced in Chapter 5, are used to minimize the amount of collision test between reactants. On the other hand, proteins are excluded of those optimization methods and collision between reactants and proteins is calculated by brute force. Each spatial cell is as least as large as the biggest participant. Since proteins are excluded, the cell size is much smaller, which results in less required collision tests per object.

In the first step, collision between proteins and reactants is tested. Since proteins are much larger and heavier than reactants, the collision has no impact on a proteins movement. On the other hand, the movement vector of the reactant has to be adjusted. Therefore, each reactant is tested for collision with each protein inside of the simulation compartment. Whenever a collision is found, a new movement vector for the reactant is calculated.
During the second step, collision between reactants is tested, whereby collision for a specific object is only tested with reactants in neighboring cells. The moved reactant is tested in a broad phase manner first, followed by the precise collision tests during the narrow phase.

Since precise collision detection requires many calculations and is costly to compute on a per frame basis, a second collision detection algorithm has been implemented. Since molecules are moving fast and a chaotic interaction is simulated, exact collision calculation is not always necessary. In many cases, an approximated approach is sufficient for the simulation purpose. Therefore, the user can decide if the exact or the simplified collision detection algorithm should be used during the simulation process. During the approximated approach, the narrow phase is not executed. This increases the overall performance immensely.

Description of the Collision Detection Algorithm

The collision detection algorithm returns a new movement vector if a collision was detected. The returned vector has the same direction as the original movement vector but its length is reduced to the maximum possible length the object can move without colliding. How this new movement vector is created is shown in Algorithm 6.1. The mathematical approach is based on the concept proposed by Heuvel and Jackson [44]

The spatial difference between two objects, measured from their center, the original movement vector and the sum of the object's radii are passed to the algorithm. It should be noticed, that the algorithm can be used for either partially static or dynamic collision tests. In a partially static collision test, the reactant is moving while the protein is seen as stationary object and the movement vector corresponds to the movement of the reactant. Whereas in a dynamic collision test, both participants are moving and both movements

have to be used. In this case, the movement vector corresponds to the movement of one reactant in relation to the movement of the second reactant, as shown in equation 6.1.

$$realtiveMovement = movement1 - movement2. \tag{6.1}$$

In Line 3 of the Algorithm 6.1, the closest distance *cd* to the second reactant is calculated. If this value is not larger than zero, then the objects are moving in different directions. This early escape test is done in line 4.
The squared distance from *cd* to the center of the second sphere is calculated in line 7 and is called f. This value is needed for the second escape test. When the closest distance between the spheres during the movement, which is stored in the variable f, is larger than the sum of their radii, no collision can occur. This test is done in line 12.
At this point it is certain that the two spheres will collide. Therefore, the movement vector has to be shortened. The amount of adoption is calculated in line 15 and is stored in t. The new movement length can be calculated by using the Pythagorean theorem, which is done in line 16. Finally, the shortened length of the shortened movement vector is returned.

Algorithm 6.1: Collision Detection

Input: Two float3 named $diff$ and *movement*, and a float value called *sumRadii*.
Output: The new vector length or -1.

```
1  float movementLength = length(movement);
2  float dist = length(diff);
3  float cd = dot(normalize(movement), diff);
   /* Early escape test                                    */
4  if cd <= 0 then
5  |  return -1;
6  end
7  float f = dist * dist - cd * cd;
   /* Eliminate precision errors                           */
8  if f < 0.0001 then
9  |  return f = 0.0f;
10 end
11 float sumRadiiSquared = sumRadii * sumRadii;
   /* Early escape test                                    */
12 if f >= sumRadiiSquared then
13 |  return -1;
14 end
15 float t = sumRadiiSquared - f;
16 float newMovementLength = cd - sqrt(t);
17 return newMovementLength;
```

CHAPTER 7

Results

A reaction system and a collision detection algorithm have been implemented to simulate biochemical reaction networks. Additionally, three visualization techniques, a real-time glow effect, cone clipping and a reaction tree are used in order to communicate the happening during the simulation. In a real cell, biological networks are created by multiple complex biochemical reactions, which consist of a large number of reactants. To simulate those processes and to illustrate large hierarchical structures, it is required to include many reactants in the reaction procedure. But the overall number of calculations executed during the simulation process increases with every additional reactant taking part in the reaction process. Therefore, the single steps of the simulation system must be performance efficient to enable the simulation of life inside of a whole cell.
In this chapter, the visualization of the described reaction system, the collision detection and the three visualization methods are demonstrated. Furthermore, the performance analysis of the simulation process is discussed.

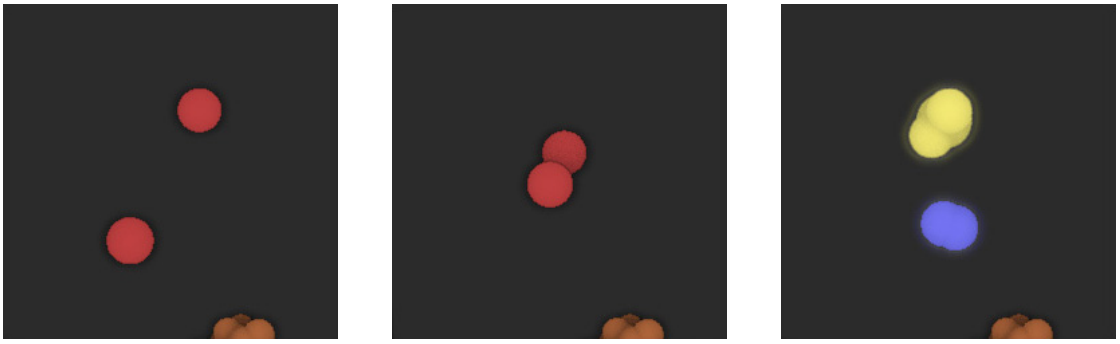
7.1 Simulation and Visualization

The simulation system is tested with a dataset created with the cellPACK [21] modeling tool, showing a human blood serum surrounding HIV virus. Additionally, 50000 reactants of five different molecular types have been placed throughout the scene. By using a compartment radius of 300, abut 350 reactants have been included in the reaction system, processing four reaction types.

Figure 7.1 shows snapshots from the reaction process of the type:

$$A + A \rightarrow B + C \tag{7.1}$$

In Figure 7.1(a) the two reactants of type A, which are included in the reaction, are moving to the calculated reaction location. The collision of the molecules is shown in



(a) (b) (c)

Figure 7.1: Snapshots from the processed reaction $A+A \rightarrow B+C$. (a) The two reactants approach each other. (b) The objects are colliding and the reaction is processed. (c) The reactants have been removed and the products are placed.

Figure 7.1(b). Since reactions are triggered by contact, the reaction system removes both reactants and creates the products B and C. Figure 7.1(c) shows the placement of the newly created products. To avoid overlapping molecules, the collision detection algorithm is used to find free areas around the reaction location.
Furthermore, it can be seen in Figure 7.1 that every molecular type has a separate color. This allows the users to imply the reactants and the ongoing reaction without considering the molecular structure.

A screenshot from the reaction tree visualization technique where the reactants are processed by four reaction types is shown in Figure 7.2. After initiating the starting reaction it took approximately 20 seconds of the simulation time until the tree structure has reached the illustrated size. It can be seen that created products are included in further reactions, which are initiated afterwards. By using the color of a molecular type for single lines it is possible for the user to determine which reaction has been processed at a specific location and which products have been included.

Screenshots of the last two implemented visualization techniques are shown in Figure 7.3. An example of the real-time glow effect is given in Figure 7.3(a). The camera is positioned in such a way that the entire compartment is shown. By highlighting the created products, the user is able to determine when and where a reaction is processed during the simulation. In Figure 7.3(b), a close-up of individual glowing molecules is shown. By comparing the glow radius in Figure 7.3(a) and (b), it can be seen that the size of the glowing effect depends on the distance of the camera to the specific object. When the camera is close to the highlighted product, the size of the glow effect around the molecule is reduced to avoid superposition of molecular structures.
A screenshot of the cone clipping effect is given in Figure 7.3(c). In this example, a clipping cone with a 15° angle is used to remove interfering proteins, which are located between the selected protein and the camera. A simplified transparency effect is applied on molecules around the clipping cone to prevent losing the impression of depth in

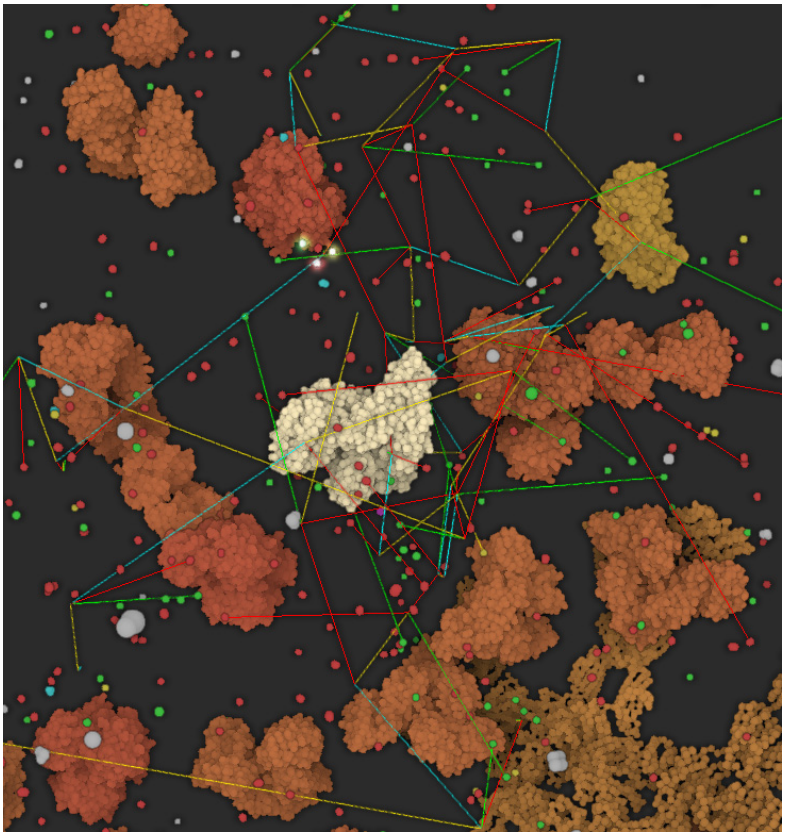


Figure 7.2: Snapshots from a reaction tree structure while simulating four different reaction types with approximately 350 reactants.

the scene. The color of a translucent object is combined with the color of concealed opaque objects or the background color, but not with other transparent object. Although, excluding translucent objects during the alpha blending process increases the rendering performance, artifacts are created when molecules are located at the edge of the scene. An example of those artifacts is shown in the top right corner of Figure 7.3(c). When no opaque objects are located behind a transparent protein, the object color is mixed with the background color black. This leads to a wrong perception of depth, because transparent objects which are located closer to the clipping cone are shown darker than the objects behind it, which are positioned further away.

7.2 Performance Analysis

Since the performance of the simulation system depends on the number of reactants participating in the reaction process, two performance tests with different numbers of reactants are discussed. The performance of both tests were measured on an Intel Core i7-3930 CPU 3.20 GHz coupled with a GeForce GTX Titan X graphics card using the

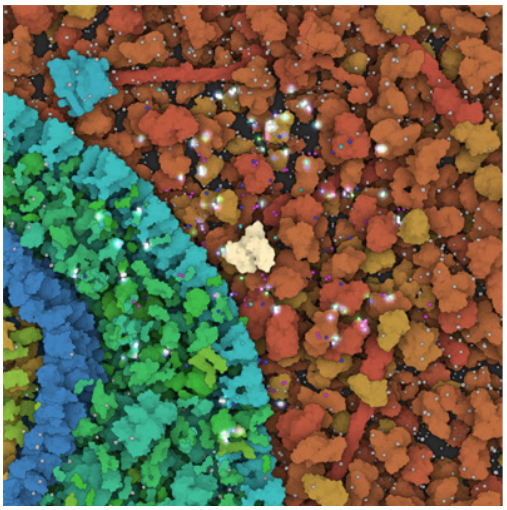
Unity3D profiler. Because the implemented visualization techniques have only a minor impact of approximately $1ms$ on the overall performance, the performance analysis discussed in this section concentrates on the single steps of the reaction system.

Simulation Steps	Test 1 [ms]	Test 2 [ms]
Reaction Generation	3	12
Movement	13	33
Reaction Execution	1	7

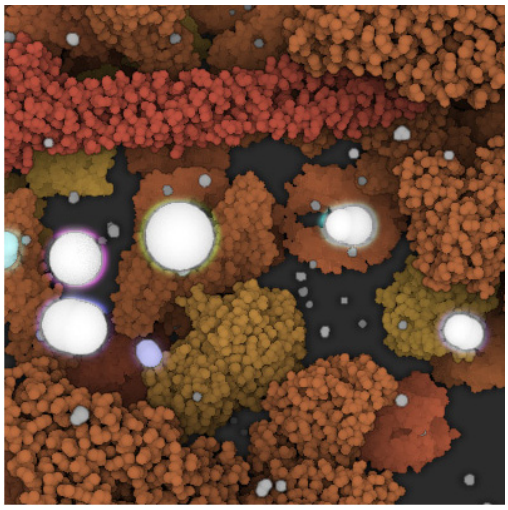
Table 7.1: Average performance results of the single steps of the reaction system during approximately 130 frames. The first test included 322 reactants and was executed at 45 frames per second, while the second test contained 1138 reactants and was performed at 18 frames per second.

Table 7.1 shows the performance results of the three main steps of the reaction system in two tests containing 322 and 1138 reactants. While reactions are generated and executed when needed, only the movement step is processed per frame. Therefore, the average processing time during approximately 130 frames was measured in both tests to give a more realistic representation of the overall processing time.
This time interval of about 130 frames corresponds with the adjusted reaction cycle of the COPASI API, in which new reactions are initiated. Since the reaction system stores the list of new reactions given by COPASI and processes them over time by initiating only 20 reactions per frame, the processing time of the reaction generation step fluctuates between $0ms$ and approximately $160ms$. While the processing time of the movement step was consistent in both tests, the reaction execution step fluctuated as well. During the first test the performance to execute completed reactions was in a range between $0ms$ and $2ms$. Since more reactions have to be executed over time with an increased number of reactants participating in the simulation system, the processing time of the reaction execution step increased and fluctuated between $0ms$ and $15ms$ during the second test. Those immense fluctuations of the processing time during reaction generation and execution lead to non-stable frame rates and possible stuttering during the simulation.

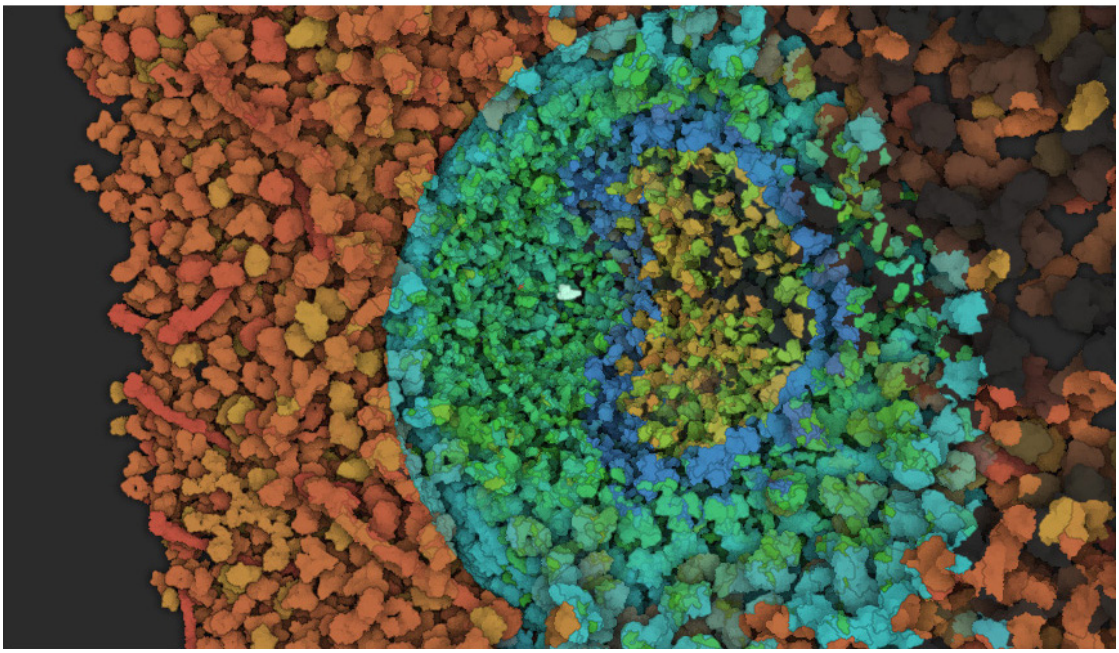
By comparing both tests it can be seen, that the movement step needs the most processing time. With $13ms$ in the first tests and $33ms$ in the second test, approximately 60% of the overall processing time is consumed by moving the reactants and applying the collision detection algorithm. This shows the importance of including optimization methods, such as the spatial subdivision and fast fixed-radius nearest neighbor algorithm, to decrease the overall number of calculations per frame.



(a) The glow effect during simulation.



(b) Close up of the highlighted reactants.



(c) Cone clipping with a 15° angle.

Figure 7.3: Snapshots of the visualization techniques real-time glow effect and cone clipping.

CHAPTER 8

Conclusion and Future Work

I have introduced a tool to simulate and visualize biochemical reactions and biological networks in a large and complex multiscale structural model. Due to the collision detection algorithm, which is able to work at the level of single atoms, the implemented reaction system is able to simulate molecular interactions in a realistic way. For load balancing, advanced GPU programming was used for data manipulation as well as optimization algorithms to minimize the number of calculations per frame and to enable simulation with more than 1000 reactants participating in the reaction process. Due to the size and complexity of cells and their inner life, containing billions of atoms, it is necessary to visually communicate the proceedings during the simulation to the user. Therefore, three visualization techniques have been implemented. A real-time glow effect in combination with a conical clipping object are used to point out interesting areas where reactions occur. The third implemented visualization technique, a hierarchical structure called reaction tree, is used to illustrate a biological network, by illustrating the impact of one reaction on the entire reaction system.

Although this reaction system is able simulate more realistic molecular behavior it also encloses some limitations: Firstly, due to the complexity and large amount of calculations per frame, the simulation process is reduced to a spherical compartment. To increase the maximum number of reactants included in the reaction process, further optimization algorithms for movement and collision detection could be implemented. The second limitation refers to the limited reaction animation, which are triggered by contact and do not include the spatial position of single atoms. In future work, molecular structure of the participants could be broken apart in real-time to illustrate how atoms dock together and how new bonds are created at the correct spatial position. Another limitation is about the simplicity of the reaction process. In CellPathway, an arbitrary protein can be assigned to single reactions. Thus, the reaction position is moved to the location of the specific protein, whereby the molecular type of the protein is not taken into account. Furthermore, proteins are not synthesized or broken apart during the reaction process.

In further versions, a more advanced reaction system could be implemented to simulate changes of the cell structure. Finally, the explanatory visualization of biochemical processes is limited by the overall amount of ongoing reactions during the simulation. Currently, it is not possible that the user can follow a specific molecule during the reaction process. Therefore, a combination of a leaded camera and a slow motion technique could be implemented to improve the way how the processing of individual reactions are communicated to the user. Furthermore, this approach would allow the user to follow a specific molecule through out the scene and to illustrate the molecule's reaction pathway.

APPENDIX A

Download and Installation

1. Download the current version of Unity3D from the webpage `https://unity3d.com/get-unity/download?ref=personal` and install the program.
2. Clone the git repository from `https://github.com/UnityDevTeam/CellPathway`. Alternatively, you can download the ZIP-file with a browser of your choice and extract the package afterwards.
3. Start Unity3D and open the project by accessing its root folder.
4. Open the scene "hiv+blood" located in "Assets/Scenes".
5. If the graphical user interface is not already open, click on "Windows → CellPathway". CellPathway is now ready to use.

APPENDIX B

Configure and Run a Simulation

1. **Import and Place Molecules**: First, molecule species have to be imported. This is done with the Molecular Species Editor shown in Figure B.1(a). By clicking the button "From PDB File", a window opens and a file describing the molecular structure must be selected. A new PDB-file can be downloaded from the RCSB protein data bank [23] or a existing file can be used, which are located in the folder "TestMolecules". All imported molecules are displayed in the Editor. Next, the quantity of the molecular species must be set, since the default quantity is 0. After all desired molecules have been set and customized, they can be placed inside of the scene.

2. **Set Reactions**: All reactions which should be processed during the simulation must be defined in the Reaction Editor shown in Figure B.1(B). By clicking on the button "Add Reaction", a new empty reaction is added to the system. The reactants are defined through the listboxes on the left side of the arrow while the products are defined with the listboxes on the right. To include a random protein into the reaction process, the checkbox "React on Protein" must be activated. Optionally, a reaction name can be set.

3. **Simulation and Visualization**: The simulation can only be performed in the Unity3D playmode. To enable the playmode, the arrow-button in the toolbar must be pressed. Since the simulation is only performed inside of the compartment sphere, a protein must be selected by pressing the right mouse button. This places the center of the compartment at the proteins location. The selected protein is highlighted for visual feedback. Simulation settings can be changed in the Visualization and Simulation Editor shown in Figure B.1(c). The following settings can be changed:

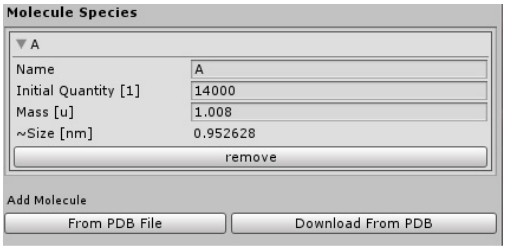
    - *Fast Collision Detection*: When the fast collision detection is enabled, the narrow phase of the collision detection algorithm is not executed during the simulation. This increases the frame rate significantly.

- *Show Copasi Output*: When enabled, the initiation orders send by the COPASI API are shown in the Unity3D console.
- *Compartment Checks*: Defines the time interval when the simulation system should check if new molecules have been moved inside of the compartment. Those molecules are included in the reaction process as well.
- *Compartment-Radius*: Defines the radius of the compartment sphere.
- *Speed*: Sets the speed of the reactants during simulation.
- *Volume*: Defines the volume of the simulation compartment. Is needed by the COPASI API when a new simulation is started.
- *Simulation Steps*: Sets the real time interval of a simulation step in the COPASI API [27].
- *Visualization Steps*: Sets the real time interval of a visualization step in the COPASI API [27].

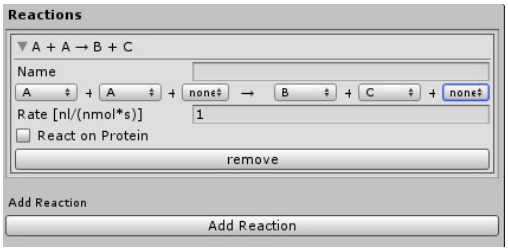
When the simulation settings are adjusted, the simulation process can be started by pressing the "Start Simulation" button and afterwards stopped by pressing the "Stop Simulation" button.

Two visualization settings can be changed: *Cone-Angle* and *Fixed-Color*. The cone angle defines the angle of the clipping cone, while fixed-color sets a static color for the reaction tree.

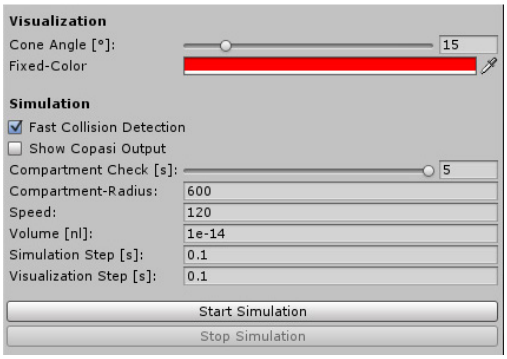
4. **Reaction Tree**: The Reaction Tree Editor is shown in Figure B.1(d). This window is only displayed when a simulation has been started and is located at the left corner in the Unity3D game view. The type of the initiating reaction can be set by selecting a reaction in the "Reactions" section. The linecolor of the reaction tree can either be set to the molecular color type or to a fixed color. By clicking on the "Initialize" button, the simulation is paused and the tree structure is initialized. After one click, the button text changes to "Play". By pressing the button again, the simulation process is resumed and the reaction tree is created over time. The "Pause" button allows to pause the simulation process at any time.



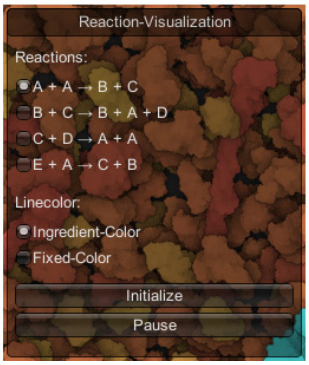
(a) Molecular Species Editor.



(b) Reaction Editor.



(c) Visualization and Simulation Editor.



(d) Reaction Tree Editor.

Figure B.1: Shows the graphical user interface.

Bibliography

[1] Laurent J. and Nightingale J. Darwinism and evolutionary economics. *Edward Elgar Publishing*, 2001.

[2] Yanoff M. and Duker J.S. Ophthalmology. *Saunders Elsevier*, 2013.

[3] KhanAcademy. https://www.khanacademy.org/science/biology/macromolecules. Accessed: 2016-02-11.

[4] Crick F. Central Dogma of Molecular Biology. *Nature*, Vol.227, No.5258, pp.561–563, 1970.

[5] Lodish H., Berk A., Zipursky S.L., Matsudaira P., Baltimore D., Darnell J. Molecular Cell Biology. *W. H. Freeman*, 4th Edition, 2000.

[6] Genetics Home Reference, a service of the U.S. National Library of Medicine. http://ghr.nlm.nih.gov/handbook. Accessed: 2016-02-11.

[7] Hahn S. Structure and mechanism of the RNA polymerase II transcription machinery. *Nature structural & molecular biology*, NIH Public Access, Vol.11, No.5, pp.394–403, 2004.

[8] Genetic Science Learning Center, a service of the University of Utha - Health Science. http://learn.genetics.utah.edu/. Accessed: 2016-02-11.

[9] Wuchty S., Ravasz E., Barabási A-L. The Architecture of Biological Networks. *Complex systems science in biomedicine*, Springer, pp.165–181, 2006.

[10] Phizicky E.M., Fields S. Protein-Protein Interactions: Methods for Detection and Analysis. *Microbiological reviews*, Am Soc Microbiol, Vol.59, No.1, pp.94–123, 1995.

[11] Hawoong J., Sean P.M, Barabási A-L., Oltvai Z.N. Lethality and centrality in protein networks. *Nature*, Nature Publishing Group, Vol.411, No.6833, pp.41–42, 2001.

[12] Berridge M.J. Cell signalling biology. *Portland Press Limited*, 2006.

[13] Wikipedia - Transcription. https://en.wikipedia.org/wiki/Transcription_(genetics). Accessed: 2016-02-11.

[14] Wikipedia - Transfer RNA. https://en.wikipedia.org/wiki/Transfer_RNA. Accessed: 2016-02-11.

[15] Riken - The Central Dogma. http://www.riken.jp/en/pr/videos/intro/20080622/. Accessed: 2016-02-11.

[16] Science Explained. http://science-explained.com/theory/dna-rna-and-protein/. Accessed: 2016-02-11.

[17] Le Muzic M., Parulek J., Stavrum A.K., Viola I. Illustrative visualization of molecular reactions using omniscient intelligence and passive agents. *Computer Graphics Forum*, Vol.33, No.3, pp.141–150, June 2014.

[18] Hoops S., Sahle S., Gauges R., Lee C., Pahle J., Simus N., Singhal M., Xu L., Mendes P., Kummer U. Copasi - a complex pathway simulator. *Bioinformatics*, Vol.22, No.24, pp.3067–3074, 2006.

[19] Kubera Y., Mathieu P., Picault S. Everything can be agent! *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, Vol.1, pp.1547—1548, 2010.

[20] Le Muzic M., Autin L., Parulek J. and Viola I. cellVIEW: a Tool for Illustrative and Multi-Scale Rendering of Large Biomolecular Datasets. *The Eurographics Association*, 2015.

[21] Johnson G. T., Autin L., Al-Alusi M., Goodsell D. S., Sanner M. F., Olson A. J. cellpack: a virtual mesoscope to model and visualize structural systems biology. *Nature methods*, Vol.12, No.1 , pp.85-–91, 2015.

[22] Unity Technologies. https://unity3d.com/. Accessed: 2016-02-11.

[23] PDB. 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. Accessed: 2016-02-11.

[24] Stryer L., Berg J.M., Tymoczko J.L. Biochemistry (5th ed.). *San Francisco: W.H. Freeman*, 2002.

[25] Ciechomski P.H., Klann M., Mange R., Koeppl H. From biochemical reaction networks to 3D dynamics in the cell: The ZigCell3D modeling, simulation and visualisation framework. *Biological Data Visualization (BioVis)*, IEEE Symposium, pp.41–48, 2013.

[26] SBGN wbsite. http://www.sbgn.org/Main_Page. Accessed: 2016-02-11.

[27] Gehrer Daniel. CellUnity - an Interactive Tool for Illustrative Visualization of Molecular Reactions. *Institute of Computer Graphics and Algorithms, University of Technology*, 2014.

[28] Hucka, Michael and Finney, Andrew and Sauro, Herbert M and Bolouri, Hamid and Doyle, John C and Kitano, Hiroaki and Arkin, Adam P and Bornstein, Benjamin J and Bray, Dennis and Cornish-Bowden, Athel and others. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, Vol.19, No.4, pp.524—531, 2003.

[29] Stiles, Joel R. and Bartol, Thomas M. and others. Monte Carlo methods for simulating realistic synaptic microphysiology using MCell. *Computational neuroscience: realistic modeling for experimentalists*, CRC Press, Boca Raton, FL, pp.87–127, 2001.

[30] Kerr, Rex A. and Bartol, Thomas M. and Kaminsky, Boris and Dittrich, Markus and Chang, Jen-Chien Jack and Baden, Scott B. and Sejnowski, Terrence J. and Stiles, Joel R. Fast Monte Carlo simulation methods for biological reaction-diffusion systems in solution and on surfaces. *SIAM Journal on Scientific Computing*, Vol.30, No.6, pp.3126–3149, 2008.

[31] CellBlender website. https://code.google.com/p/cellblender/. Accessed: 2016-02-11.

[32] Blender Online Community. Blender - a 3d modelling and rendering package, http://www.blender.org. Accessed: 2016-02-11.

[33] Le Muzic M., Waldner M., Parulek J., Viola I. Illustrative Timelapse: A Technique for Illustrative Visualization of Particle-Based Simulations. *Visualization Symposium (PacificVis)*, 2015 IEEE Pacific, Vol., pp.247–254, 2015.

[34] Grottel S., Krone M., Müller C., Reina G., Ertl T. MegaMol - A Prototyping Framework for Particle-Based Visualization. *Visualization and Computer Graphics*, IEEE Transaction, Vol.21, No.2, pp.201–214, 2014.

[35] Microsoft: Windows forms website. http://msdn.microsoft.com/library/dd30h2yb.aspx. Accessed: 2016-02-11.

[36] Qt website. http://www.qt.io/developers/. Accessed: 2016-02-11.

[37] Hermosillaa P., Guallarb V., Vinacuaa A., Vázqueza P.P. High quality illustrative effects for molecular rendering. *Computers & Graphics*, Vol.54, pp.113–120, 2016.

[38] Staib J., Grottel S., Gumhold S. Visualization of particle-based data with transparency and ambient occlusion. *Comput Graph Forum*, Vol.34, pp.151-–160, 2015.

[39] Cormen T.H. Introduction to algorithms, Chapter 8.2. *MIT press*, 2009.

[40] Nguyen H. GPU Gems 3, Chapter 32. *Addison-Wesley Professional*, 2007.

[41] Nvidia - GPU Technology Conference. http://on-demand.gputechconf.com/gtc/2014/presentations/S4117-fast-fixed-radius-nearest-neighbor-gpu.pdf. Accessed: 2016-02-11.

[42] Fernando R. GPU Gems, Chapter 21. *Addison-Wesley Professional*, 2004.

[43] Microsoft - Compute Shader Overview. https://msdn.microsoft.com/en-us/library/windows/desktop/ff476331(v=vs.85).aspx. Accessed: 2016-02-11.

[44] Gamasutra. http://www.gamasutra.com/view/feature/131424. Accessed: 2016-02-11.