FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Visualisierung von molekularen Maschinen mittels agentenbasierter Animation

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Medizinische Informatik

eingereicht von

Daniel Gehrer, BSc
Matrikelnummer 1125229

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Associate Prof. Dipl.-Ing. Dr.techn. Ivan Viola
Mitwirkung: Univ.Ass. Dipl.-Ing. Tobias Daniel Klein

Wien, 21. April 2017

Daniel Gehrer	Ivan Viola

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Visualization of molecular machinery using agent-based animation

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Medical Informatics

by

Daniel Gehrer, BSc
Registration Number 1125229

to the Faculty of Informatics

at the TU Wien

Advisor: Associate Prof. Dipl.-Ing. Dr.techn. Ivan Viola
Assistance: Univ.Ass. Dipl.-Ing. Tobias Daniel Klein

Vienna, 21st April, 2017

Daniel Gehrer	Ivan Viola

Erklärung zur Verfassung der Arbeit

Daniel Gehrer, BSc
Spengergasse 27 /2.602, 1050 Wien

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, 21. April 2017

Daniel Gehrer

Acknowledgements

I would like to thank everyone who supported me throughout the thesis in one way or another. Many thanks to my advisor Ivan Viola, who contributed with numerous ideas and brought me in contact with some renowned researches in the field of molecular visualization. I also want to thank the whole Computer Graphics Institute for welcoming me and providing me a place to work. Finally, I want to thank my family who supported me during the entire course of my studies, and without whom this would not have been possible.

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

Kurzfassung

Diese Arbeit stellt ein agentenbasiertes Modell vor, das zur Animation von molekularen Maschinen verwendet werden kann. In der Regel werden molekulare Maschinen mittels Schlüsselbild-Animationen visualisiert. Das Animieren von großen molekularen Strukturen mittels Schlüsselbild-Animation in Standard 3D-Programmen kann sehr mühsam sein, da Hunderte oder Tausende molekulare Teilchen, unter Berücksichtigung verschiedener biologischer Phänomene, von Hand animiert werden müssen. Im Zuge der Arbeit wurde ein prototypisches Framework erstellt, das mit Hilfe von unabhängigen Agenten das repetitive Animieren von molekularen Teilchen vermeidet. Anstatt die molekularen Teilchen direkt zu animieren, verwendet das Framework Verhaltensbeschreibungen für jede Art von Molekularteilchen. Die Animation entsteht durch die Interaktion der molekularen Teilchen untereinander, die durch ihr Verhalten definiert wird. Das Framework implementiert ein abstraktes Modell, welches die Interaktion zwischen molekularen Teilchen ermöglicht. Die Methodik für die Erstellung des Frameworks war getrieben von Beispielen. Drei molekulare Maschinen wurden mit dem Framework visualisiert. Während dieses Prozesses wurde das Framework iterativ verbessert, um die Anforderungen für jede neue molekulare Maschine zu erfüllen. Die daraus resultierenden Animationen zeigen, dass die agentenbasierte Animation eine praktikable Vorgehensweise für molekulare Maschinen ist.

Abstract

This work proposes an agent-based model for animating molecular machines. Usually molecular machines are visualized using key-frame animation. Creating large molecular assemblies with key-frame animation in standard 3D software can be a tedious task, because hundreds or thousands of molecular particles have to be animated by hand, considering various biological phenomena. To avoid repetitive animation of molecular particles, a prototypic framework is implemented, that employs an agent-based approach. Instead of animating the molecular particles directly, the framework utilizes behavior descriptions for each type of molecular particle. The animation results from the molecular particles interacting with each other as defined by their behavior. Interaction between molecular particles is enabled by an abstract model that is implemented by the framework. The methodology for creating the framework was driven through learning by example. Three molecular machines are visualized using the framework. During this process, the framework was iteratively improved, to meet the requirements for each new molecular machine. The resulted animations demonstrate that agent-based animation is a viable option for molecular machines.

Contents

CHAPTER 1

Introduction

Every second, an overwhelming number of molecular processes take place in our cells. A great variety of macromolecular complexes are involved in these processes by performing specific tasks. These complexes have some analogies to machines in the macroscopic world, therefore they are often called molecular machines [BBC+01].

Since molecular machines are much smaller than the wavelengths of visible light, optical microscopes cannot be used to analyze their structures and functionalities [Goo04]. Instead, biologists examine the structure of molecular machines by using X-ray crystallography, NMR spectroscopy and cryo-electron microscopy [Goo04, Cal15]. Since the functionality cannot be observed directly, the biologists develop models on how such machines may work, based on the spectroscopy data. The model then have to be conveyed to fellow scientists and students. Textual descriptions are hard to imagine. Simple two-dimensional (2D) visualizations provide better insights, but fail to convey the richness of a three-dimensional (3D) environment [Iwa10]. Animated 3D visualizations proved to be powerful tools for describing complex molecular processes to diverse audiences. This kind of visualizations are not only an impactful way to communicate complicated molecular processes, but are also widely appreciated for their beauty [FHFH15]. Furthermore, creating an animation can also help the biologist to reflect on the model, detect flaws and find missing parts [Rot16]. This, again, can help to refine hypotheses and design new experiments [FHFH15].

1.1 Problem Statement

For creating 3D animations, usually standard 3D software, like Cinema4D [MAX17] or Maya [Aut17], is used. The drawback of using standard 3D software for animating molecular machines, is that this kind of software often requires weeks to months of training and regular use to create only a basic molecular animation [Iwa15]. Sometimes biologists have to hire programmers and animators to create models and animations.

This is not only expensive, but also provides a lot of opportunities of miscommunication between the biologist and the animators [WTHT14]. Another challenge for animators is to model large molecular assemblies which can include thousands of molecules. Animating such a large number of objects can be very time consuming and tedious, especially when every object has to be animated manually [Iwa15].

Great support for molecular visualization is provided by the Protein Data Bank (PDB) [BWF+00]. PDB is a public database which contains hundreds of thousands molecular structures [BWF+00]. There are plug-ins for most well-known 3D software that simplify the import of molecular structures from the PDB into the 3D software [JAG+11, Too17]. But still, the 3D software must be mastered by the biologist or a hired animator [WTHT14]. Still, the animation of each object has to be done by hand [Iwa15].

Due to the problems involved by using standard 3D software, a few tools emerged that are dedicated to simplify the process of creating molecular animations [IMS+17, WTHT14]. However, these tools are either meant for very simple animations only [IMS+17] or use a similar key-framing approach that is also used by standard 3D software [WTHT14]. So, the needed effort for animating a large number of objects remains the same.

1.2 Aim of the Work

The aim of this work is to simplify and accelerate the animation process of molecular machines in their environment by avoiding repetitive and automatable tasks. This is achieved by creating a framework that supports the biologist in the animation process.

The result is a prototypic framework that animates molecular environments using an agent-based model. The model represents an abstract molecular environment. It can be extended and customized to adopt the desired behavior. A lot of animation tasks are abstracted away by the framework. Instead of animating a scene using key-frames, the framework requires the animator to define the behavior of molecular particles, using a formal behavior description. These molecular particles act as independent agents that interact with the environment according to their behavior description. They can respond to changing environmental conditions, like the concentration of specific ligands, and can also actively participate by initiating or releasing molecular bonds. A molecular machine is built of molecules [Goo04]. By describing the behavior of each molecule, the functionality of the whole machine can be animated.

This type of animation scales very well, because not every molecular particle has to be animated separately. Instead, only the behavior of each type of molecular particle has to be described. Thus, the animation effort is independent from the total number of molecular particles, but only dependent on the number of distinct types.

1.3 Methodological Approach

To learn about the visualization and animation process, three molecular machines were visualized and animated from scratch. For this, a framework was created that supports the animator. The three molecular machines differ in complexity and the simplest one was chosen to start with. The framework was extended with features in order to being able to animate the molecular machines. Simultaneously, the features were added in an abstract manner, so that they can be reused for other animations. The framework was iteratively improved with each machine.

The result is an abstract model of a molecular environment. Since the model is abstract, it is not only usable for the three animated molecular machines, but is also applicable to a variety of other molecular machines. The model is motivated by biological concepts. It is used in behavior descriptions for molecular particles to interact with the molecular environment.

1.4 Structure of the Work

The following Chapter 2 introduces the reader to the biological basics of molecular machines. At the end of the chapter, three molecular machines are discussed, that are used as examples throughout this work.

Chapter 3 discusses related tools that pursue the same or a similar goal. Moreover, different animation methods, valuable data sources and specialized rendering approaches that are related to molecular machines, are discussed and compared.

The abstract machinery model proposed by this work is explained in Chapter 4. It provides a detailed description of each model component, including their relation to biology. The emergence and the incremental improvements of the model are discussed at the end of the chapter.

In Chapter 5 the implementation of the animation framework is discussed. It explains how existing tools are used and how the model is integrated in framework.

The molecular machines animated with the framework are presented in Chapter 6. The single components of the molecular machines are explained and it is shown how the framework was used to animate them.

The final Chapter 7 discusses the advantages and flaws of the model and the framework, and compares the framework to existing solutions. The chapter also provides some insights and thoughts about this work.

Starting at page 91, there is a glossary that explains the biological terms used in this work. The used acronyms are also listed there.

CHAPTER 2

Biological Background

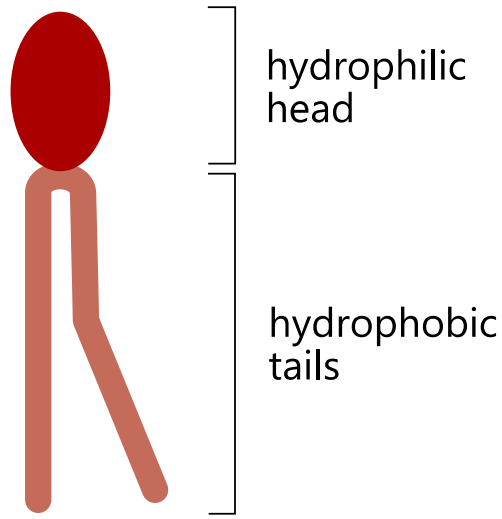
This chapter is dedicated to readers who are unfamiliar with the biological basics of molecular machines. In the following sections, a few important biological terms are defined and explained. The first section presents some vital components of the cell. In the second section, the focus is moved to the building blocks of molecular machines and the cell: the macromolecules. Then special properties of molecular machines are discussed. The final section discusses three examples of molecular machines. Later chapters will refer to those examples to provide a better insight.

2.1 Cells

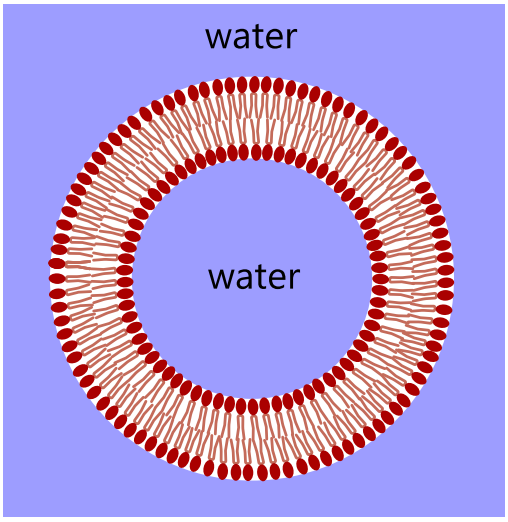
Cells are small, membrane-enclosed units filled with water and other chemicals. They form the fundamental units of life. The simplest forms are solitary cells, as most bacteria and algae. But also higher organisms, including plants, animals and humans, are basically just communities of individual cells. Despite the great variety of existing cells, all of them share a very important feature: they store their genetic instructions in deoxyribonucleic acid (DNA). Every cell has the capability to synthesize protein molecules out of the blueprints that are encoded in the DNA. Proteins are vital components of the cell. They are involved in chemical reactions, give the cell its shape and also control its behavior [AJL+14]. As proteins are also an integral part of molecular machines, they are discusses later in this chapter in more detail.

2.1.1 Cell Membrane

All cells are enclosed by a membrane that separates the inside of the cell from the outside environment. The membrane consists of lipid molecules that have a head that likes to interact with water (hydrophilic) and a tails that avoid contact with water (hydrophobic). A simple illustration of a lipid molecule is shown in Figure 2.1a. When placed in water,



(a) Illustration of a lipid molecule with its hydrophilic head and its hydrophobic tails [AJL+14].



(b) Lipids hiding their hydrophobic tails from the water while exposing the hydrophilic heads, which results in a lipid bilayer [AJL+14].

Figure 2.1: Lipids forming a lipid bilayer.

such molecules arrange spontaneously to hide their hydrophobic tails from the water, while exposing their hydrophilic heads. This arrangement is assumed because it is energetically favorable. The arrangement results in small vesicles that are enclosed by two layers of lipid molecules, a so-called lipid bilayer [AJL+14]. Multiple lipid molecules forming a vesicle are displayed in Figure 2.1b. Such lipid bilayers form the membrane of cells.

Lipid bilayers are not rigid but fluid. Since the lipid molecules are not bound together chemically, they are free to move as long as the hydrophobic tails are isolated from the water [AJL+14]. Therefore, lipid bilayers form dynamic structures where the lipids can move within the plane of the bilayer. Due to this freedom of movement, lipids are constantly flowing relative to one another. Sometimes the lipids even flip from one face of the bilayer to the other. Because lipid bilayers are fluid, they are capable of spontaneously healing damage [Goo04].

A cell is not a closed system, but actively interacts with its environment by exchanging molecules and ions. This exchange is necessary for the cell to obtain nutrients and also to get rid of waste produced inside the cell. Therefore, the cell membrane must be permeable for certain substances. Specialized proteins that are embedded in the membrane allow this permeability. These proteins are present in every cell and transport specific substances across the membrane. They determine which substances enter the cell and which substances leave and in this way influence the chemistry of the cell [AJL+14].

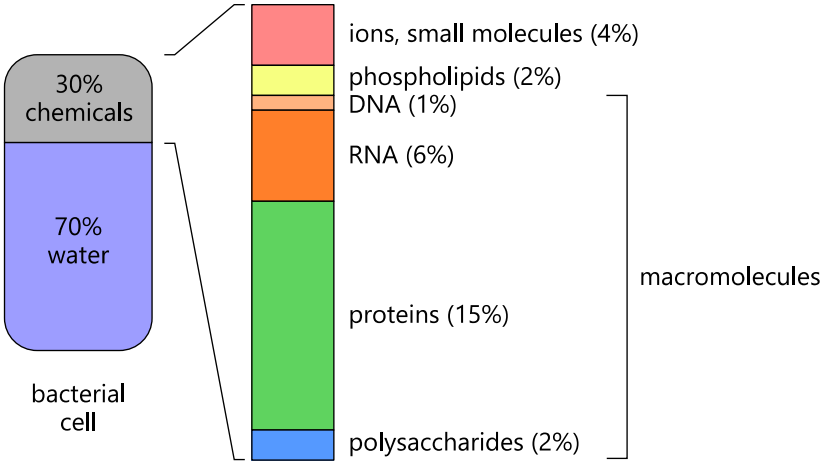


Figure 2.2: The approximate composition of a bacterial cell [AJL+14].

2.1.2 Cytoplasm

The cytoplasm is the content of the cell. It includes everything that is enclosed by the membrane. Inside the cytoplasm there are other membrane enclosed subunits called organelles. Organelles are separated subunits with specialized chemical functions. The cytoplasm consists mainly of water with a mixture of large and small molecules that carry out many of the cell's chemical processes. In an approximate bacterial cell, water makes up 70 % of the cytoplasm. Macromolecules like DNA, proteins and polysaccharides make up 24 %. The remaining 6 % are lipids, small molecules and ions. The ratio of the cytoplasm content is shown in Figure 2.2. The composition of a human cell is similar [AJL+14].

The small organic molecules are usually floating free in the cytoplasm and play important roles in a lot of different processes. Some are processed and used as subunits to build larger molecules like proteins or nucleic acids. Others serve as energy sources when broken down and transformed into smaller molecules. Many small molecules serve multiple purposes. In a typical cell, there are about thousand different kinds of these small molecules [AJL+14].

2.2 Macromolecules

A molecule is group of two or more atoms that are held together by covalent bonding, which is the strongest chemical bonding type. Covalent bonding is 10-100 times stronger than other attractive forces between atoms. Thus, it is possible to define clear boundaries of a molecule. Macromolecules are molecules which are typically composed of several thousands of atoms. Most macromolecules are constructed by linking smaller subunits together. These macromolecules are called polymers and their subunits are called

monomers [AJL+14].

In a cell, various types of macromolecules are synthesized and processed. Often polymers are synthesized by adding one monomer at a time to a long chain. Step by step, a large and complex macromolecule is built. Most macromolecules are composed of a set of monomers that are slightly different from one another. The monomers are not added to the chain randomly, but are added in a particular order [AJL+14]. The two most important macromolecules in respect to molecular machines are nucleic acids and proteins [Goo04]. Thus, both of them are further discussed in the following sections.

2.2.1 Nucleic Acids

Nucleic acids are long molecular chains that are present in every cell. The most famous nucleic acid is DNA, which is used to store genetic information. The other nucleic acid is ribonucleic acid (RNA). RNA molecules have a variety of functions, but mostly they are used to transfer information of single genes across the cell [AJL+14].

Nucleic acids are polymers of subunits that are called nucleotides. They form long linear strands and can contain millions of nucleotides. Each nucleotide has one of four bases attached. The sequence of those bases encodes the information contained in the nucleic acid. DNA uses the bases cytosine, guanine, adenine and thymine for encoding the information. The encoding in RNA is very similar, but it uses uracil instead of thymine. The bases are often abbreviated by their initial letters: C, G, A, T and U. The encoded information is often written as a sequence of those letters [AJL+14].

DNA is usually paired with a partner strand. The two stands wrap around each other, thus form the familiar double helix structure. The bases of the nucleotides of the two strands form base-pairs, where the nucleotide of the partner strand has the complementary base attached. The complementary base of C is G, and the complementary of A is T (or U in RNA). RNA is most often single-stranded and folded onto itself [AJL+14]. The schematic structure of DNA and RNA is illustrated in Figure 2.3.

The DNA contains the entire genetic information of the organism. The sequence is divided in segments called genes. On demand, the cell copies certain genes from the long DNA sequence to shorter RNA strands. These RNA strands, which contain the information of a single gene, are then transported to other regions of the cell, where they serve as blueprints for protein production [AJL+14].

2.2.2 Proteins

Proteins play a crucial role in cells as they are versatile and can perform thousands of distinct functions. They are involved in countless cellular processes. The reason for this versatileness is that a protein is made of a chain of amino acids. Amino acids are a class of chemical substances with each having unique physical and chemical properties. Some amino acids are negatively or positively charged, some are chemically reactive, some are hydrophobic, etc. Depending on the sequence of the amino acids in the chain, the protein

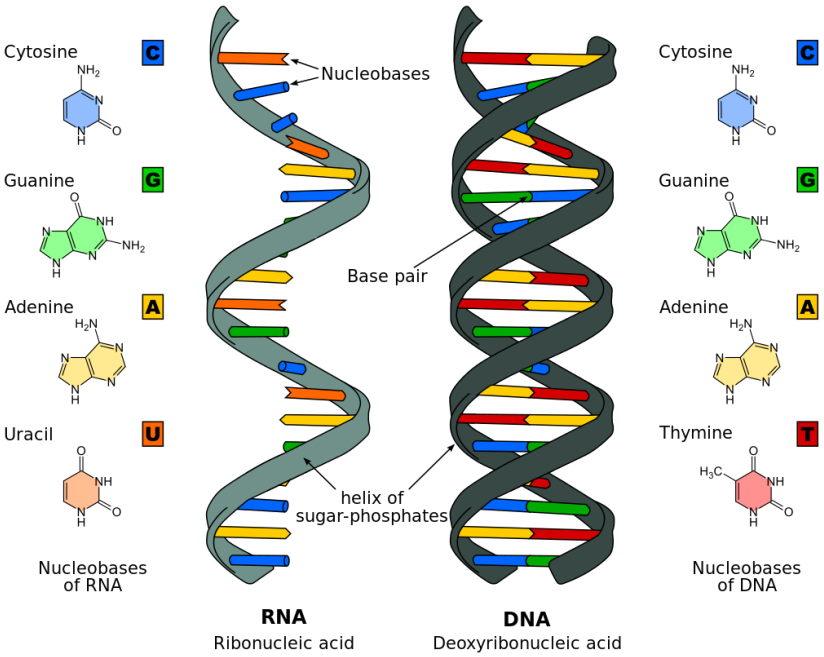


Figure 2.3: Comparison of a single-stranded RNA and a double-stranded DNA [Spo10].

is suitable for different tasks. The number of possible amino acid chains is a tremendous. Depending on the sequence and the number of amino acids in the chain, the resulting protein can be of very different size and shape. Proteins control the behavior of the cell. They serve as support structures, chemical catalysts, membrane channels, etc. [AJL+14]

The cell has capabilities to produce proteins on demand. Proteins are built from a set of 20 amino acids. The sequence of amino acids that make up a protein are encoded in genes. When a certain protein is needed in the cell, the corresponding gene is copied from the DNA to a RNA strand. The RNA strand is then transported to molecular machines called ribosomes. Ribosomes can build new proteins using the information encoded in the RNA strand as blueprint. The bases of the nucleotides of the RNA strand encode the amino acids that have to be linked together to form a certain protein. The ribosomes read the RNA strand sequentially, and link the amino acids together, one after another [AJL+14].

Most of bonds between the amino acids in the chain are flexible. Thus, theoretically allowing them to adopt a large number of shapes. However, some amino acids attract or repel each other, which causes the chain to fold. Usually, the chain preferably adopts one particular conformation that is determined by the amino acid sequence. Most proteins and many of the RNA strands fold tightly into one highly preferred conformation in this way, which represents a state of lowest energy [AJL+14]. This process is illustrated in Figure 2.4.

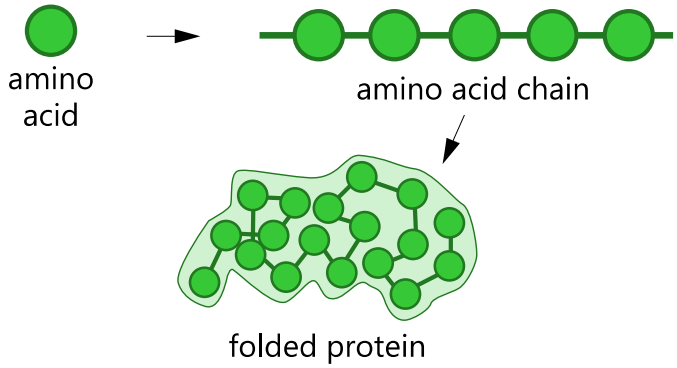


Figure 2.4: A Protein consists of a chain of linked together amino acids. In a watery environment the chain usually folds into a stable conformation [AJL+14].

2.3 Interaction of Molecules

Atoms in molecules are held together by covalent bonding. Not only atoms can bind together, but also whole molecules to form larger molecular complexes. Bonds between multiple molecules are not held together by covalent bonding but by electrical attraction. The electrical bonding is common at hydrogen atoms of a molecule, therefore this bonding type is called hydrogen bonding. The attraction is much weaker than covalent bonding. Thus, a lot of the weak hydrogen bonds are necessary to strongly bind two molecules together. To allow a great number of hydrogen bonds to establish between two molecules, these molecules must fit together very closely and have an exact complementary shape and charge profile on their surface. A simple illustration of this effect is shown in Figure 2.5 [AJL+14]. If the shape or charge on the surface does not match exactly, the two molecules cannot bind because only a few hydrogen bonds are not strong enough to hold the molecules together. Therefore, this kind of binding provides great specificity. Often it is only possible for exactly one molecule from the many thousands to bind to a certain site. Because the strength of the binding depends on the number of hydrogen bonds that are established, bonds of almost any strength are possible. The area where other molecules can bind to is often called binding site [Goo04].

Small molecules serve as subunits for macromolecules. Multiple macromolecules, again, can bind together on binding sites of complementary shape and charges to form large complexes with multiple moving parts. These macromolecular complexes often perform necessary cellular processes. Thus, they are called molecular machines [BBC+01]. The hierarchy of subunits to macromolecules, to macromolecular complexes is shown in Figure 2.6.

When a protein interacts with other molecules in the cell, its conformation often slightly changes. This change in shape is called allostery and is crucial to the function of the protein. Many proteins are allosteric. They can adopt two or more slightly different conformations. The change of the conformation regulates the activity of the protein.

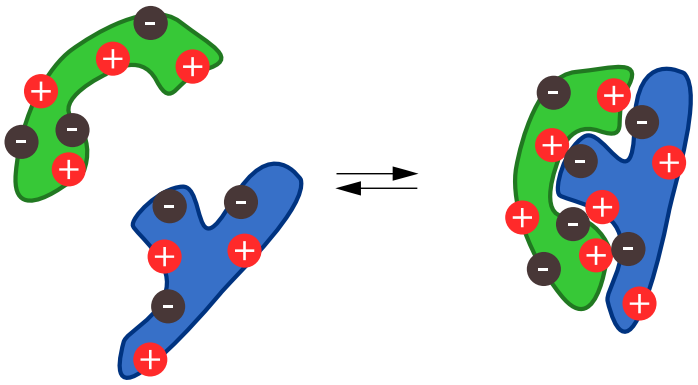


Figure 2.5: Binding of two molecules due to complementary shape and charges on their surface [AJL+14].

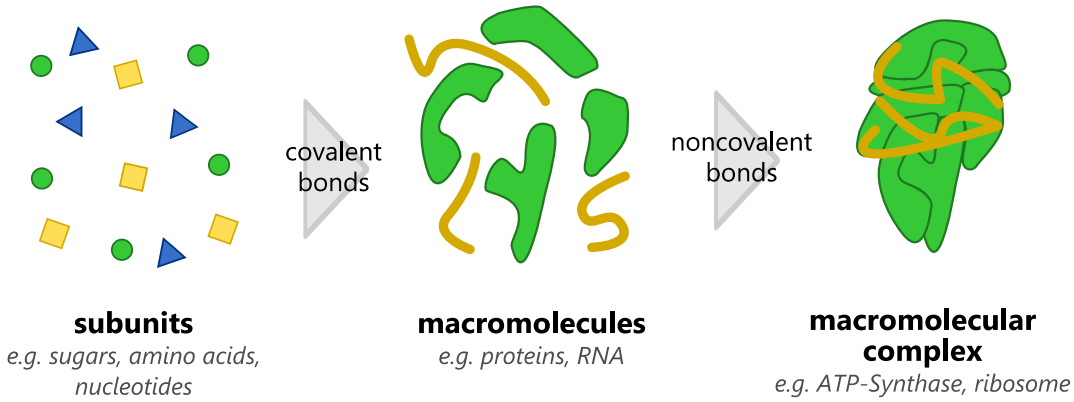


Figure 2.6: Small molecules join together to form macromolecules, which can assemble into large macromolecular complexes [AJL+14].

The reason for this is that conformation changes also affect the surface of the protein. Thus, other molecules the protein usually interacts with, may not bind tightly, because the shapes are not exactly complementary anymore. The weak bonds are not strong enough to hold the two molecules together. Often proteins have a separate binding site for regulatory molecules that can activate or deactivate the function of the protein. This is especially common for enzymes. Most enzymes are special proteins that accelerate chemical reactions with specific molecules (substrates). A simple example of allosteric inhibition and activation in an enzyme is shown in Figure 2.7.

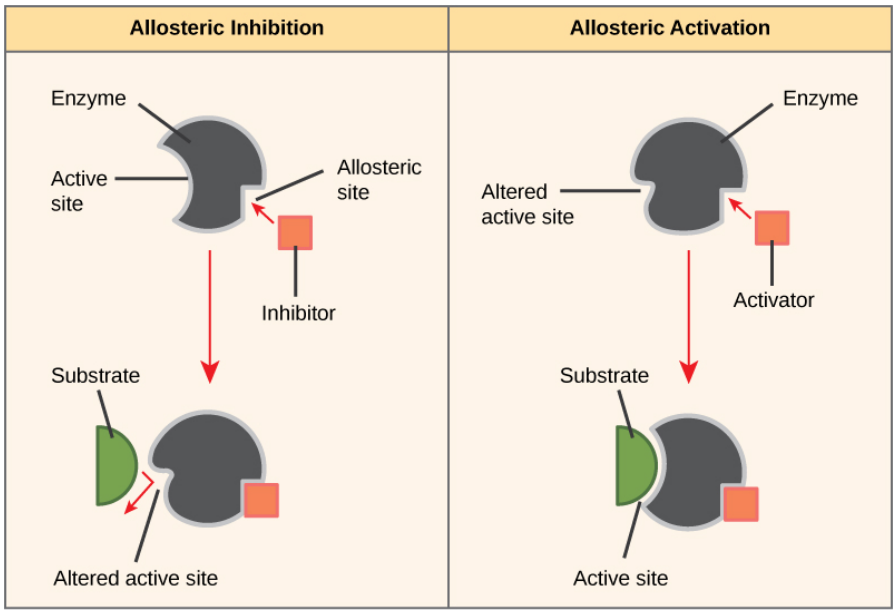


Figure 2.7: Allosteric inhibitors modify the active site of the enzyme so that substrate binding is reduced or prevented. In contrast, allosteric activators modify the active site of the enzyme so that the affinity for the substrate increases [Col13]

2.4 Molecular Machines

Molecular machines are molecules or complexes of molecules that carry out physical and chemical processes. Most molecular machines found in cells are assemblies of multiple proteins and specifically folded RNA molecules. There are a lot of differences to machines in our familiar world. Instead of being designed by humans, molecular machines in cells have been developed by the process of evolution. The environment in which molecular machines operate is also quite different to the macroscopic environment we know [Goo04].

2.4.1 Environment

Molecular machines are very small, typically they are only a few nano meters in size. At this scale, objects behave quite differently to machines in the macroscopic world. Mass increases with the cube of the size of an object. That means objects at the nanoscale have hardly any mass. Due to this reason, gravity and also inertia are negligible at the nanoscale. On the other hand, thermal motion is a significant force. Also the attraction forces between molecules are way stronger than gravity. The environment is very crowded and the molecules are in constant motion. This causes the molecule to collide with other molecules all the time and therefore undergo random Brownian motion. The random motion is quite fast. A molecule released in typical bacterial cell is equally likely found anywhere in the cell within 1/100 s. Any two molecules within a micrometer sized cell meet each other every second.

Molecular machines are built of a discrete number of atoms. Therefore, the typical continuous representations used for macroscopic machines is not appropriate. Moreover, there are no smooth transition from one state to another. Instead, the molecular machines jump from state to state when the appropriate chemical energy is applied. Also bulk properties such as viscosity and friction do not exist at nanoscale, instead individual atomic properties must be considered [Goo04].

For molecular machines to operate correctly, they require a water environment as it is available in cells. A lot of processes, e.g. protein folding, require to be surrounded by water.

2.4.2 Self-Assembly

Molecular machines are not built like macroscopic machines. As mentioned before, the macromolecules that make up molecular machines, are flexible chains, which spontaneously fold into specific structures. Due to the Brownian motion, all molecules interact with each other. When two or more molecules with complementary binding sites meet, they are held together by the attraction forces at the binding sites. In that way, macromolecules then assemble themselves to complexes, which ultimately build molecular machines. This process is termed self-assembly. Self-assembly is very common in biology and not only used by molecular machines, but also by other structures, e.g. the cell membrane [Goo04].

A lot of structures in cells are symmetric, allowing the cell to build them using identical subunits. In that way, large assemblies can be described with a little amount of information. This modularity also provides for error control. Molecules that are constructed incorrectly are not capable of binding to other building blocks, because their binding sites do not match when they are corrupted [Goo04].

In most structures, the binding of one module to another increases the affinity for additional modules, due to allosteric activation. This leads to an all-or-nothing construction. Once started, the self-assemble process is very fast [Goo04].

2.4.3 Energy

Molecular machines need energy for driving difficult chemical reactions and to power directed motion. At nanoscale there are a lot of energetic processes that can be harnessed to power molecular machines [Goo04].

Cells pump ions across the membrane to create an electrochemical gradient. Molecular machines located in the membrane can harness this gradient by letting ions flow back through the membrane. The energy of the flow can be used to perform chemical or mechanical work [Goo04].

Energy is also stored in chemical bonds. The most common biological fuel molecule is adenosine triphosphate (ATP). Each molecule of ATP has three phosphates bound together. As each phosphate is negatively charged, which is very unfavorable, since they repel each other. By cleaving off one phosphate from the ATP, the chemical energy that

was trapped in the bond can be used to power various processes. The result is adenosine diphosphate (ADP), a phosphate molecule and the released energy [Goo04].

There are a lot of other energy sources that can be harnessed by molecular machines, for example, light energy or electrical energy. However, they are not further discussed, as they are not relevant for this work.

2.5 Examples of Molecular Machines

The following sections describe three examples molecular machines. Illustrations are shown and the biological processes that drive the machines are explained.

2.5.1 Hemoglobin

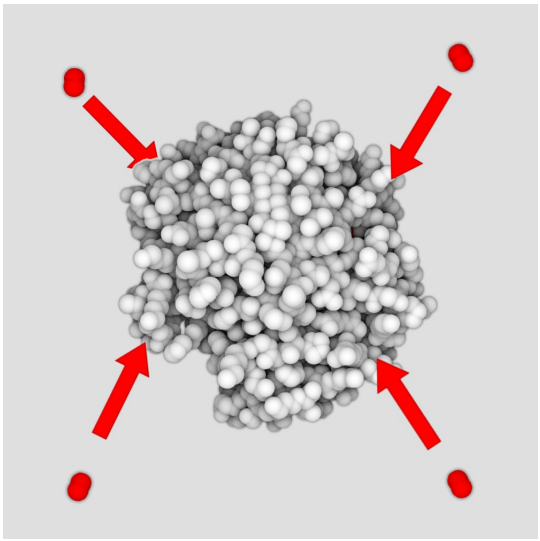
Hemoglobin is a molecule for oxygen-transport in red blood cells. It is a small molecular machine that picks up oxygen in the lung, and delivers it where it is needed. Hemoglobin has four binding sites where oxygen can bind to. Oxygen does not bind to all binding sites simultaneously, but one after another. Once the first oxygen is bound, a small change in the hemoglobin structure is induced. This structural change makes it easier for the next oxygen to bind. Thus, binding the first oxygen is the hardest, but from then it gets easier and easier [AJL$^+$14].

In the lungs, where oxygen is plentiful, the first oxygen binds easily and then quickly fills up the remaining binding sites. As the blood circulates through the body, it passes areas where the oxygen concentration is low. Here the hemoglobin releases its bound oxygen. As soon as the first oxygen drops off, the remaining binding sites release the other oxygen molecules more easily, as the shape changes back [AJL$^+$14]. When no oxygen is bound, the hemoglobin is called deoxygenated, when all binding sites are bound to oxygen, it is called oxygenated. An illustration of these two states is shown in Figure 2.8.

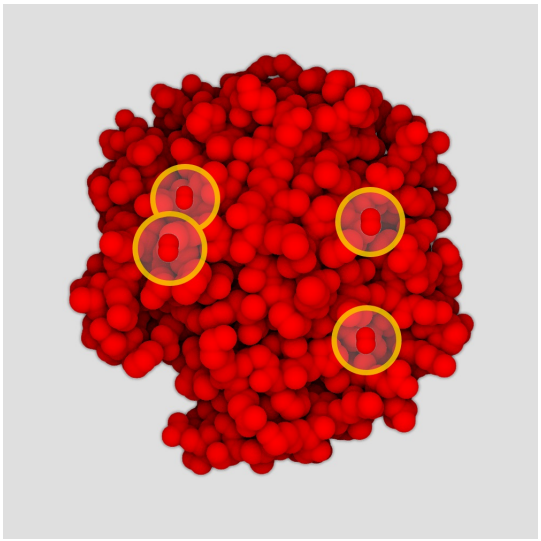
Carbon-monoxide is a dangerous gas for living organisms, because its affinity to hemoglobin is much higher than the affinity of oxygen. When carbon-monoxide binds to hemoglobin instead of oxygen, the oxygen-carrying capacity of the blood decreases drastically. Without sufficient oxygen, organs can fail and the organism dies. Due to the high affinity to hemoglobin, carbon-monoxide is much harder to unbind from hemoglobin compared to oxygen [GDHS03].

2.5.2 ATP-Synthase

Most cellular processes are powered by ATP. When energy is needed, ATP is cleaved into ADP and phosphate. This liberates energy that was previously stored in the bond. The liberated energy is then used to power a cellular process. ATP-synthase is a molecular machine that binds ADP and phosphate back together to form ATP, so that it can be used again. The energy needed for this process is obtained from a proton (H+) gradient over a membrane [Goo04].



(a) Deoxygenated hemoglobin molecule with four oxygen molecules that are attracted by the four binding sites.



(b) Oxygenated hemoglobin molecule. The oxygen molecules are bound at binding sites inside the hemoglobin and made visible inside the yellow circles.

Figure 2.8: Deoxygenated and oxygenated hemoglobin

ATP-synthase is located in the membrane and provides a channel for protons from outside the membrane to get inside. Since the proton concentration outside is higher than inside, the protons want to move in. The energy from this proton flow drives a rotor. The rotor, again, drives a generator that is capable of binding ADP and phosphate back together [Goo04].

The structure of ATP-synthase is shown in Figure 2.9. The so-called F_0 complex is located in the membrane and consists of a few equal subunits called F_0c. The number of F_0c can vary between organisms, but is typically around 10 to 12. The F_0 complex as a whole acts as the rotor of the machine. Protons from outside the membrane can enter the machine via the stator. From there they bind to the F_0c subunit that is next to stator. As soon a proton is bound, the subunit can move away from the stator which causes the rotor to rotate by one subunit. Then, the next F_0c subunit is bound to a proton and moved away, and so on. This causes the F_0 complex to rotate. After a proton made a whole revolution, it is unbound from the F_0c subunit by the stator and released to the interior of the membrane enclosed area. As a result, protons flow from outside the membrane to the inside causing the F_0 complex to rotate, whereas the stator keeps still [Goo04].

The F_1 complex consists of six subunits, three $F_1\alpha$ and three $F_1\beta$ proteins. The $F_1\alpha$ and $F_1\beta$ in the F_1 complex arranged in circular shape alternately. Together they form three sites where an ADP and a phosphate can bind respectively. ADP and phosphate

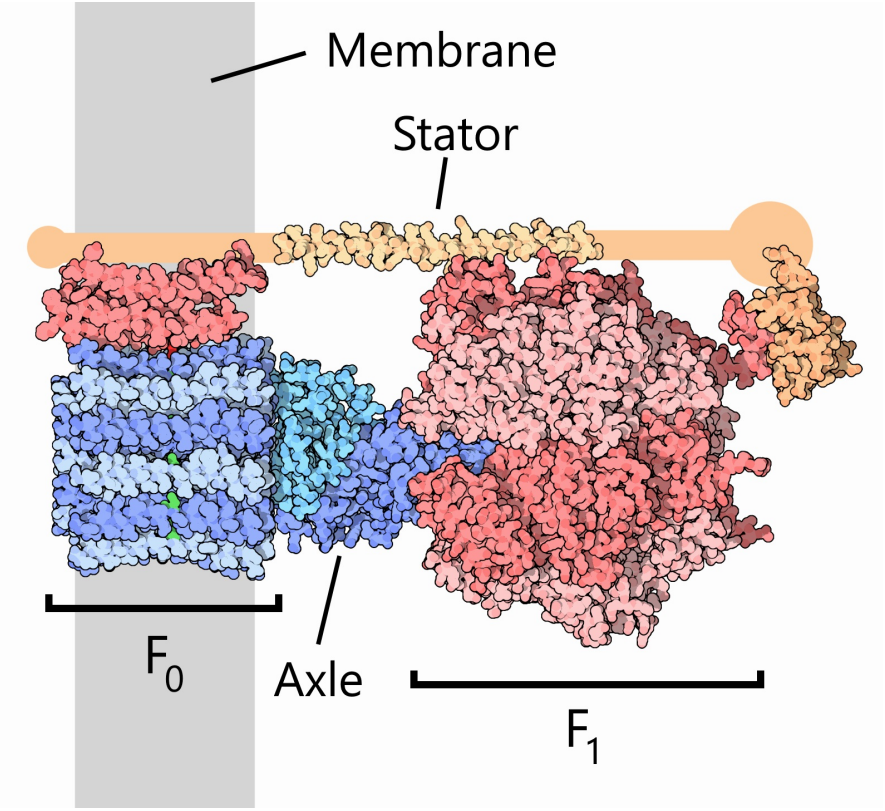


Figure 2.9: Structure of ATP-synthase [GDZ+05].

are attracted by those sites and bind to them [Goo04].

The F_1 complex is connected to the stator and does not rotate. The F_0 complex is connected to an axle that rotates with the F_0 complex. The axle extends into the F_1 complex. The rotation of the axle causes conformation changes in the F_1 complex, which again, causes the ADP and phosphate molecules at the binding sites of the F_1 complex to bind together and form ATP. The ATP is then released from this site and another ADP and phosphate can bind. In one revolution of the rotor, produces three ATP molecules [Goo04].

2.5.3 Kinesin

Kinesin is a molecular machine that moves along long tubular structures in the cell. The tubular structures are found throughout the cytoplasm and are called microtubules. They are composed of subunits called tubulin. Tubulin is a protein that consists of two parts: tubulin-α and tubulin-β. The kinesin can bind to the tubulin-β parts of the microtubule and then move along the microtubule with a movement similar to walking. Via a long neck, typically cargo is connected which is transported through the cell using the kinesin. The energy needed for the movement is obtained by cleaving ATP [Goo04].

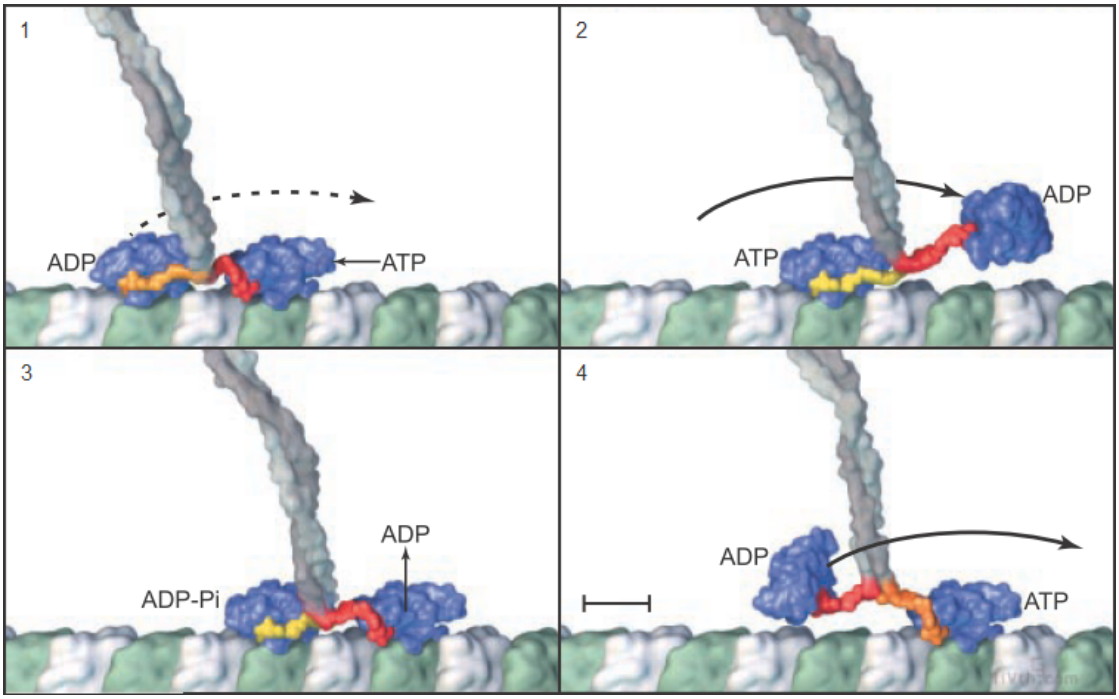


Figure 2.10: Steps of kinesin. Results in a forwards movement similar to walking [VM00].

The kinesin consists of two heads. The heads are the parts that bind to the microtubule. They are also the parts where the ATP is cleaved. At the heads, so-called neck-linkers are attached, which connect the heads with the long neck where the cargo is connected. The point where the neck-linkers of both heads meet is referred to as hip. Kinesin moves step by step, by pushing the heads forward alternately. The walking process is shown in Figure 2.10 [VMJ00].

At the beginning, when kinesin is not yet bound to a microtubule, both heads contain ADP. Due to Brownian motion it moves through the environment randomly. As soon as one kinesin head makes initial contact with a tubulin of a microtubule, it binds weakly to it. Then the conformation changes and the head binds tightly. Due to structural restrictions, only one kinesin head can bind tightly at a time. The tight binding releases the ADP. ATP then rapidly enters the empty binding site (Figure 2.10 Frame 1), which causes the neck-linker to be pulled towards the head. This causes the detached trailing head to be thrown forward and allows it to reach the next tubulin-β binding site (Figure 2.10 Frame 2). The now trailing head cleaves the ATP and reverts to a weak tubulin binding. The weak binding allows the phosphate to leave the trailing head, and also relaxes the neck-linker (Figure 2.10 Frame 3). Then the whole process repeats with the other head (Figure 2.10 Frame 4) [VMJ00].

CHAPTER 3

Related Work

There is a range of tools that support the visualization and animation process of molecular assemblies. Some of them cover only a small part of the whole visualization process, like providing structural information of molecules [BWF+00], or animating molecular structures [JAG+11, Too17]. But there are also tools that support the biologist through the whole process of importing, animating and rendering molecular environments [IMS+17, WTHT14]. This chapter discusses existing tools and animation methods that can be used for animating molecular environments.

3.1 Animation Methods

An animation is composed of multiple images that are intended to create the perception of motion. Those images, so-called frames, are generated from a 3D model with a set of variables, like the object's positions and orientations. For each frame, the values for those variables are slightly changed. Whereas often an object's position and orientation are changed during an animation, basically any variable can be animated. There are different methods on how to calculate appropriate values for each frame in an animation [Par12b]. A few methods that are relevant for this work are presented in this section.

Interpolation-based Animation

Interpolation-based methods allow the precise specification of the animation. There is little uncertainty about the positions, orientations, and shapes to be produced. The animator has to provide precise conditions, such as beginning and ending values, and having the values filled in for intermediate frames by some interpolation procedure. Interpolation-based methods allow a very low level of control [Par12c].

A wide spread method is key-framing. The term key-frame describes a frame in which the value of a variable is set to a specific value by the user. For frames between two

key-frames, the value of the variable is interpolated using a prescribed procedure. The simplest way to calculate intermediate values is by using linear interpolation, but often more sophisticated procedures are used. It is not only possible to interpolate numeric values, but also images and shapes. Tools that incorporate key-framing, often provide an interactive graphical user interface (GUI) with which the user can set key values and can control the interpolation curve [Par12c].

Another interpolation-based method are animation languages. An animation language defines a set of commands that control certain animations. By combining various commands, a wide variety of motions can be expressed. Most animation languages are composed of text instructions, but graphical languages are also common. The advantage of using an animation language is that all instructions that make up an animation are collected in a script. That script can regenerate the animation at any time and can be easily copied and transmitted. Also changes are easy and reproducible. The disadvantage is that the animator must be proficient with programming [Par12c].

Behavioral Animation

Instead of animating a variable like in interpolation-based approaches, in behavioral animations the cognitive process of an agent is modeled. An agent is a figure or a synthetic character in the scene with built-in rules of behavior. The term cognitive process refers to any decision-making process that results in the behavior of an agent, as reacting to the environment and voluntary or involuntary mental activity [Par12a].

Modeling behavior is used for various reasons: To release the animator from dealing with details in unimportant background animations, to generate animations on the fly or to populate virtual environments. This can be useful for animating large crowds in films, creating reactive characters in computer games, or to run simulations with individual decision-making agents involved. Depending on the requirements, the cognitive processes can be very simple, like moving towards a light, or very complex like balancing self-preservation, mood swings, etc. [Par12a].

Usual agents sense the environment and then process the sensory information together with their internal state and traits, which results in actions to interact with the environment and possibly changes in the internal state. Behavioral animation is often used when dealing with a large number of agents that interact with each other. The advantage is that the animator does not have to deal with each agent separately, instead they are animating themselves using the specified behavior [Par12a].

3.2 Data Sources

Before any animation method can be applied, the 3D structures have to be imported. In order to create visualizations that are close to reality, the structures must be based on scientific data. There are a few databases that can serve as source of scientific data in the context of molecular machines.

The largest source of molecular structures is the Protein Data Bank (PDB). The PDB is a publicly available repository for 3D atomic structures of biological molecules. The structures range from tiny proteins and parts of DNA to complex molecular machines [BWF+00]. The data is typically obtained by X-ray crystallography, NMR spectroscopy, or cryo-electron microscopy. Since a great number of scientists contribute with their investigated molecular structures, the PDB is a key resource for structural biology [BWF+00]. There are more than 118000 proteins, 3000 nucleic acids, 6000 protein and nucleic acid complexes contained in the database at the time of writing [PDB17].

The data provided in the PDB is a valuable source of static 3D structures of molecules. Some protein structures are even available in different conformations. However, the PDB does not contain information on how to animate these structures.

Molecular machines largely depend on binding capabilities of various distinct molecules. Thus, another resource that is usable for molecular visualizations is KBDOCK. KBDOCK is a database system containing binding site information for protein docking. Molecules need some very specific structural and physical properties for docking, as explained in Chapter 2. KBDOCK uses protein family classifications with coordinate data from PDB to analyze the spatial arrangements of protein interactions. The goal is to improve the understanding of biomolecular processes by comparing, classifying and modeling structural protein interactions. KBDOCK can also be used to search interactions involving different, but structurally similar proteins even when there is no similarity in the DNA sequence [GDSTR13].

For animations, the information from KBDOCK can be used to precisely model, where two molecules dock or bind. Together, KBDOCK and PDB provide a good scientific basis for structural data.

3.3 Structure Import

The scientific molecular data is the basis for the visualization and animation of molecular machinery. The next step is to import and visualize this data. A few tools exist that are specialized for this step. All tools introduced in this section are implemented as plug-ins for a 3D application and allow to import molecular structures [JAG+11, Too17, ACZ+12].

The Embedded Python Molecular Viewer (ePMV) is an open-source plug-in for the 3D software Blender [Ble17], Cinema4D [MAX17] and Maya [Aut17, JAG+11]. It allows scientists to import molecular structures from PDB into the 3D software. As soon as the structures are imported, the visualization capabilities of the 3D software can be used to animate and render the scene [JAG+11].

ePMV was created to simplify the process of preparing visual data for publications, education and for the public. The plugin extends the 3D software by molecular modeling features and also provides a common Python platform that allows users to create sophisticated algorithms and calculations. By uniting 3D software and molecular modeling software, ePMV allows to put data from various disciplines into the same context. This

allows users from various backgrounds to create professional quality interdisciplinary visualizations. Using the common Python platform, algorithms can be easily shared, which also encourages interdisciplinary research [JAG+11].

The tool is very flexible and optimized for illustrative visualizations. For animations, the key-frame animation capabilities of the 3D application can be used. Alternatively the 3D objects can be animated using Python scripts [JAG+11].

A similar tool is Molecular Maya (mMaya). It is a free plug-in for the 3D software Maya [Aut17] and offers functions to import, build and animate molecular structures. The biologist can open PDB files or download them automatically from the Protein Data Bank. Molecular Maya supports various representation forms, and is also capable of generating surface meshes out of PDB structures [Too17]. Like ePMV, mMaya relies on Maya's animation capabilities. Animating molecular environments still has to be done using key-framing [Too17].

BioBlender is plug-in for visualizing molecules in the open-source 3D software Blender [Ble17, ACZ+12]. It extends Blender's capabilities to allow biologists to import molecular structures from PDB and can calculate special surface properties. BioBlender also implements some notable visualization options for molecular surface properties. Furthermore, it is also capable of calculating protein movements on the basis of known conformations. The goal of BioBlender is to make the nanoscale world more understandable by visualizing invisible phenomena [ACZ+12].

BioBlender is primarily designed to visualize single molecules and special physical properties. There is no focus on creating animations of large molecular complexes [ACZ+12].

3.4 Rendering

Rendering is an essential part of visualizing molecular machines. There are already existing tools that primarily focus on rendering molecular structures. These tools are discussed in this section.

A tool that solely focuses on efficiently rendering large molecular assemblies is cellVIEW [MAPV15]. The system renders the molecules on the atomic level and is capable of interactively visualizing large datasets representing viruses and bacterial organisms consisting of several million atoms in real-time at 60 Hz display rate. It uses a level-of-detail scheme that allows to accelerate the rendering and also reduces visual clutter. Another acceleration technique is the dynamic generation of DNA strands directly on the graphics processing unit (GPU) out of sets of control points. cellVIEW is directly implemented inside a game engine to make it more accessible to the end-users. The system can freely be used and extended [MAPV15]. An image rendered by cellVIEW of HIV in blood plasma is shown in Figure 3.1.

Another rendering tool is JSmol [HPR+13]. It is a JavaScript tool that allows to visualize 3D chemical structures in web-browsers. It provides features for chemicals, crystals and

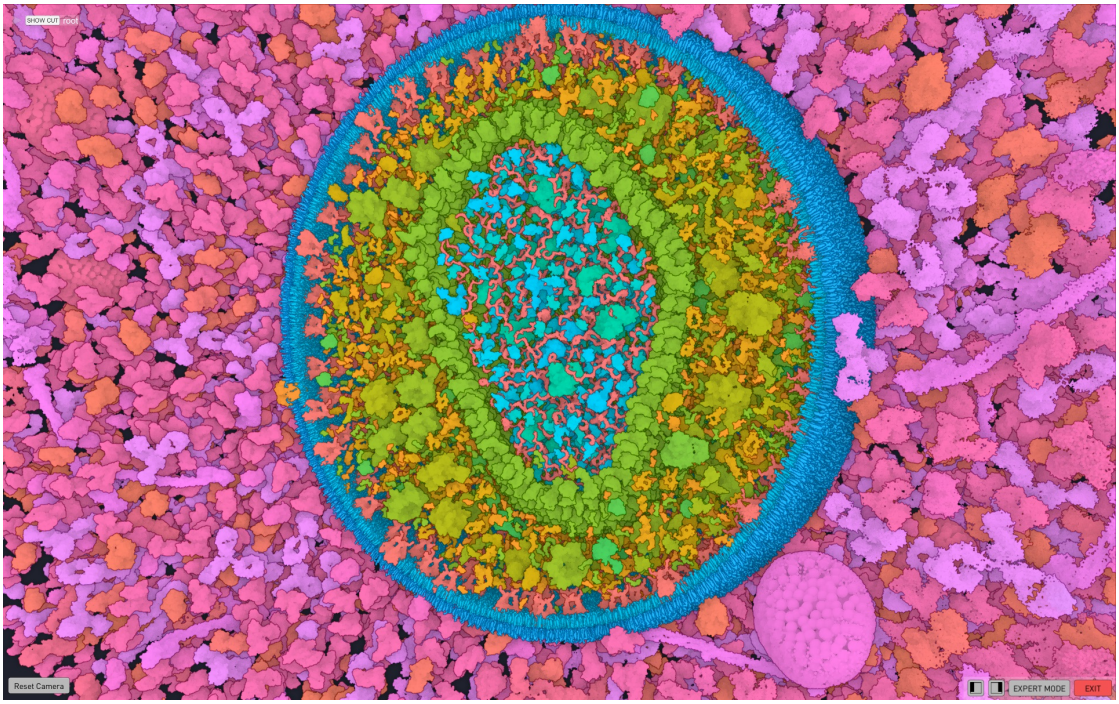


Figure 3.1: HIV in blood plasma. Rendered using cellVIEW [MAPV15]

biomolecules. JSmol displays the molecular structure stored in a file. An extensive list of file formats are supported. The user can view the structure and interactively rotate, zoom and pan. Eight different display styles are supported, including "Cartoon", "Ball and Stick", etc. JSmol renders the structures using HTML5 canvas. WebGL is avoided to gain a greater platform-independence. The tool also includes features for measuring distances and angles. Furthermore, the view can be exported to various image file formats. JSmol is one of the viewers that is integrated in the PDB website to allow users to display molecular structures directly in the browser. Figure 3.2 shows a screenshot of JSmol embedded in the PDB website. There is also a lite version of JSmol that only provides minimal functionality for small-bandwidth applications [HPR+13].

A framework that is usable for a broad domain of applications is MegaMol [GKM+15]. It is capable of processing data from thermodynamics, solid material physics and also macromolecules in biochemistry. Instead of utilizing the classical mesh-based graphics approach, MegaMol focuses on particle-based data and rendering. It is designed to handle very large datasets in an efficient manner by utilizing the GPU. MegaMol is designed as a development framework that can be adapted to custom needs [GKM+15].

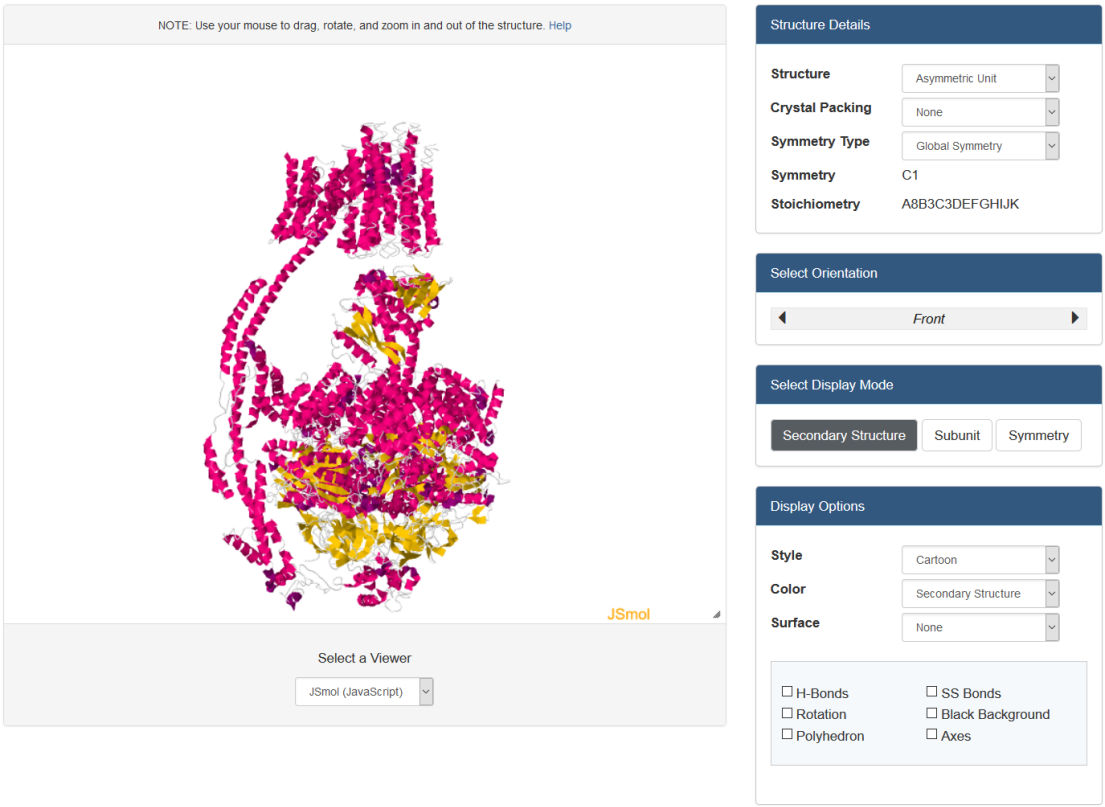


Figure 3.2: Screenshot of an ATP-Synthase complex [ZRS+15] visualized with JSmol [HPR+13] on the PDB website [BWF+00].

3.5 Molecular Animation

This category of tools supports biologists in creating their own molecular animations [IMS+17, WTHT14, PGH+04]. Everything from importing molecular structures, to animating and rendering is supported.

One such tool is the Molecular Flipbook [IMS+17]. The intended users are molecular biologists who want to visualize molecular models to communicate their ideas to their peers, their students or the public. It is only meant for simple molecular models and not for complex or cinematic quality animations [IMS+17].

The tool is not only useful for creating animations but also provides important insights to researchers [Rot16]. During the modeling and animation process, the user has to figure out how certain molecules fit together, how subunits assemble or disassemble, and missing parts become obvious. This helps to reflect on the knowledge and to fill in a lot of blanks [Rot16].

Molecular Flipbook utilizes a simple GUI which allows the user to create slides, where position and representation of the molecules can differ in each slide. The animation

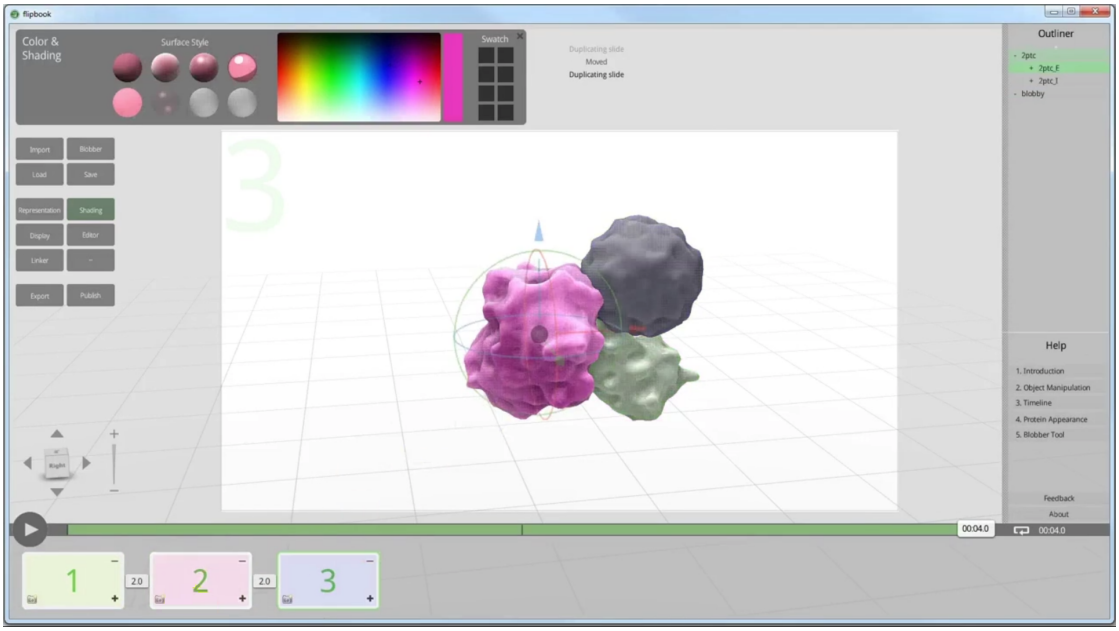


Figure 3.3: Screenshot of Molecular Flipbook. The user can create and select slides at the bottom and manipulate the scene at the center [IMS+17]

is created by interpolating between those slides, thus utilizing a simple key-frame approach [IMS+17]. A screenshot of the GUI is shown in Figure 3.3.

SketchBio is another tool that enables biologists to rapidly construct molecular animations [WTHT14]. It incorporates a two-handed manipulation technique using two six-degree-of-freedom magnetic trackers to edit a scene. That enables novel interaction patterns that accelerate common tasks when animating molecular models. The tool aims for an easier usage than standard 3D software [WTHT14].

The main motivation for the tool is to provide an easy to use interface, so that biologists can create their own animations. It is difficult to master standard 3D modeling and rendering software. Therefore, biologists often have to hire programmers or animators to create models and animations. However, this slows down work and provides opportunities for miscommunication. Hence, SketchBio aims enable the biologists to create the animations themselves [WTHT14].

Due to the use of two six-degree-of-freedom magnetic trackers, novel manipulation techniques are possible. For example, SketchBio implements a feature called "pose-mode physics" that enables rapid docking of proteins, aided by collision detection. Another feature is the "crystal-by-example" mode that allows to rapidly create polymer chains [WTHT14]. Figure 3.4 shows the six-degree-of-freedom magnetic trackers being used to control SketchBio. SketchBio makes extensive use of existing tools and libraries like UCSF Chimera [PGH+04] for PDB support and Blender [Ble17] for its render capabilities [WTHT14].

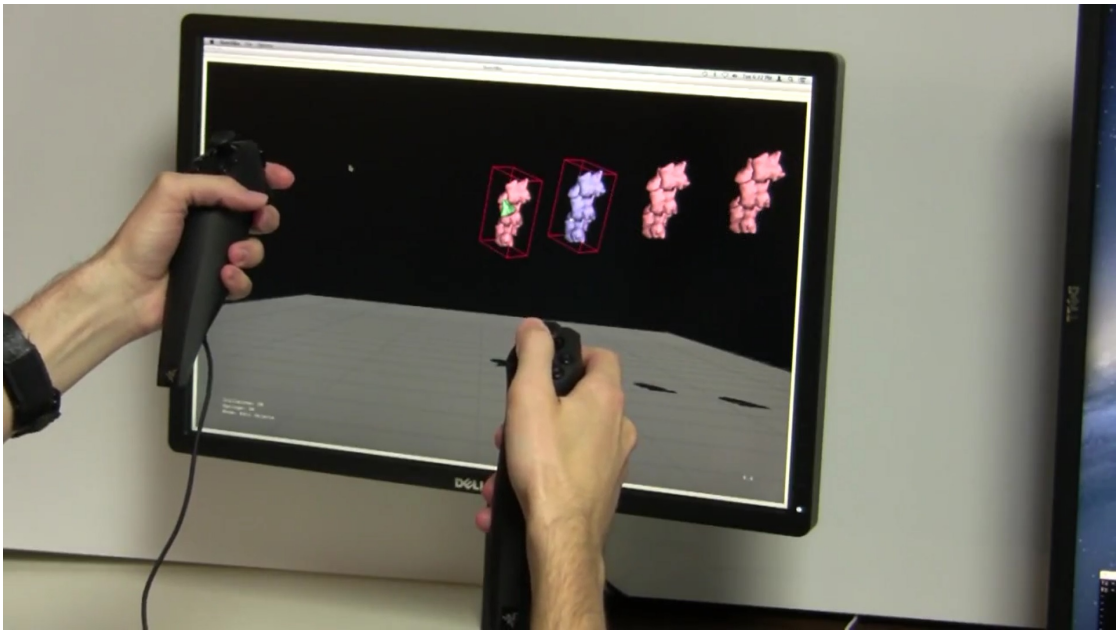


Figure 3.4: Editing of a scene in SketchBio using two six-degree-of-freedom magnetic trackers [The13].

To animate a scene, the user can move along a timeline and create key-frames for designated molecules. The value between key-frames are interpolated using splines to produce smooth motion [WTHT14].

UCSF Chimera is a tool for interactive visualization and analysis of molecular structures and related data [PGH+04]. It also allows the user to generate high-quality images and animations. The tool is free of charge for academic, government, nonprofit, and personal use [PGH+04].

UCSF Chimera is designed to meet the needs of structural biologists. Thus, the tool supports a long list of analysis and visualization features and is highly extensible [PGH+04]. Animations can be generated by a script consisting of animation commands [UCS12]. The script can then be executed and the resulted animation can be saved as video in various file formats [PGH+04]. A screenshot of UCSF Chimera is shown in Figure 3.5.

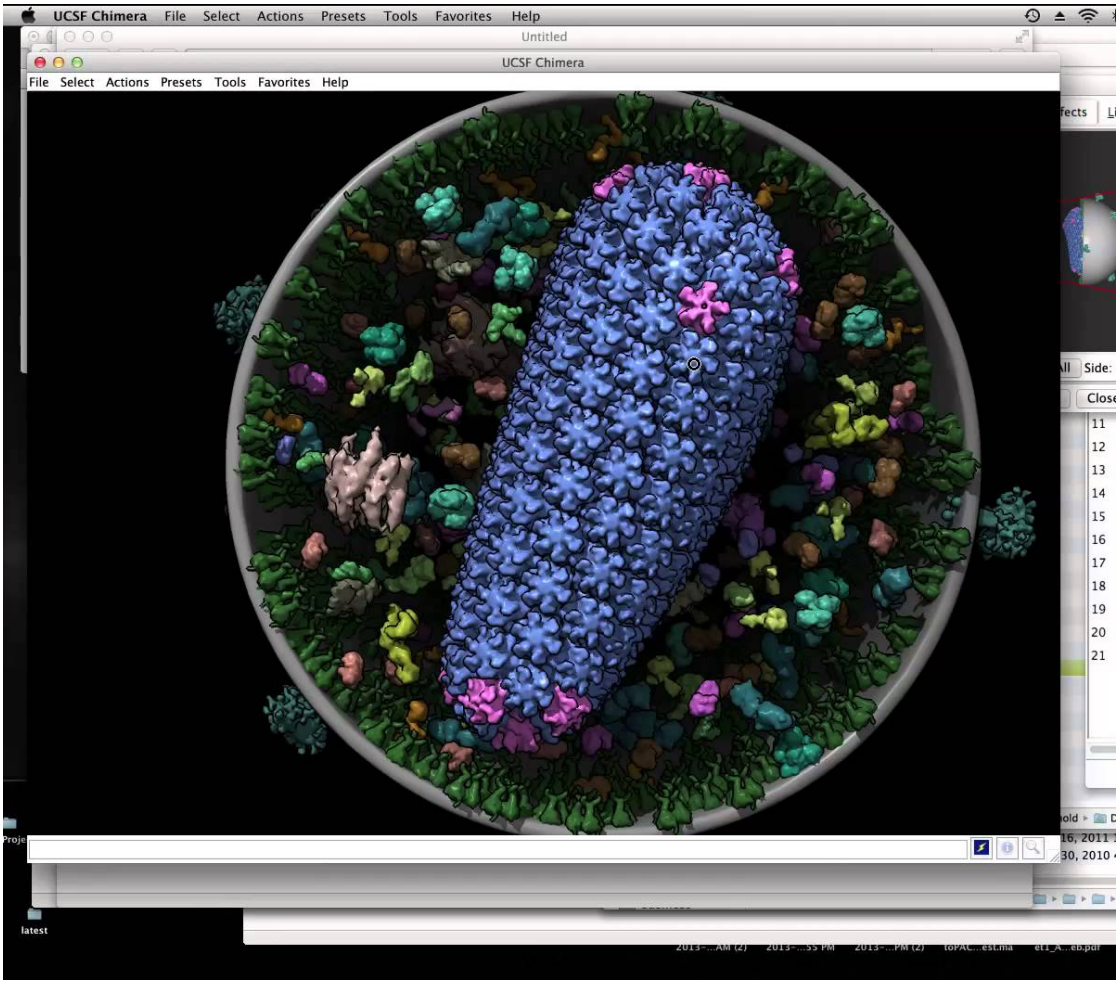


Figure 3.5: A HIV dataset displayed using UCSF Chimera [Joh13].

CHAPTER 4

Machinery Model

Animations are used by biologists to communicate ideas to fellow scientists and students. Biologists often use standard 3D software to model molecular environments according to their ideas [Iwa10]. A few specialized tools also exist. Similar to standard 3D software, these tools also focus on animating 3D objects in a scene using key-frame animation [IMS+17, WTHT14] or animation languages [PGH+04]. This paper proposes another way of producing animations. Instead of animating 3D objects directly, the animator describes the behavior of molecular particles on a high level. The molecular particles with the behavior description attached, are then placed in an environment. Each molecular particle interacts with other particles according to their behavior description, which causes the animation. In that way a behavioral animation method is utilized [Par12a]. If the behavior description is sufficiently detailed, the molecular particles also react correctly, when environmental conditions change. This agent-based approach should speed up the animation process, because not every molecular particle has to be animated by hand anymore. The animation also scales very well, since the behavior has to be defined only once for each type of molecular particle, independent from the total number of molecular particles in the scene. In contrast, interpolation-based animation requires the biologist to animate each particle independently which can be a very tedious task [Iwa10].

On a very high-level, the behavior could be defined by modeling physical and chemical properties of the molecular particles. An animation could then be accomplished by running a biological simulation. The animation would be defined very abstractly, by a set of physical and chemical properties. Such an abstract behavior definition is very simple and reusable, since it can be used for any number of agents and in all possible environments. On the other side it gives the animator only very little control over the resulting animation, because all the animator can do to influence the animation is to fine-tune physical and chemical properties until the desired animation is achieved. Since the goal of this work is to allow the animator to communicate ideas, the animator

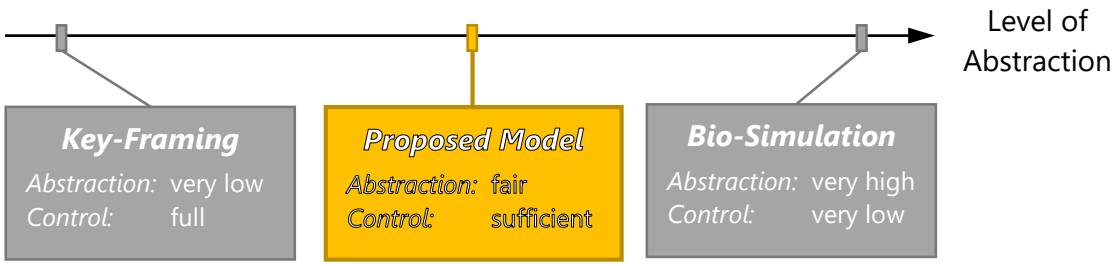


Figure 4.1: Different approaches arranged on a level of abstraction axis.

should have more direct control over the resulting animation. A key-frame animation, for example, gives the animator full control over the resulting animation. However, it requires the animator to precisely specify each detail of an animation [Owe05], which is often a repetitive and tedious task, especially for molecular animations [Iwa15]. To combine the advantages of both extremes, an abstract model is proposed, that allows to describe the behavior on a high-level. The proposed model is abstract enough to relieve the animator from a lot of repetitive tasks, but not abstract enough to deprive the animator of needed control over the resulting animation. As shown in Figure 4.1, the proposed model is placed somewhere between those both extremes.

The proposed model is based on common molecular details and features that are discussed in this chapter. The following section introduces the model by showing its relation to biological concepts. The further sections discuss all model components in more detail. The final section is dedicated to the methodology and the model development process.

4.1 Relation to Biology

The components of the model are inspired by biology. The following text introduces the model components and explains how they are related to biological concepts. An overview of the model and the relations of the components is illustrated in Figure 4.2.

Entities: In biology, the environment is inhabited by a large number of molecular particles. In the model, molecular particles are represented as so-called entities. Entities are defined as the smallest, indivisible molecular particles in the environment. In most cases a molecule equals an entity, but that is not always true. The special cases are discussed later. Entities are the central components in this model. They act as autonomous agents, and are the targets of the behavioral animation.

Behavior: In biology, molecular particles move and interact with each other according to physical forces and chemical reactions depending on their physical and chemical properties [AJL+14]. In existing solutions, these effects have to be animated manually by the user using key-framing [IMS+17, WTHT14]. Instead of animating each molecular particle by hand, the proposed model requires the user to provide a behavior description

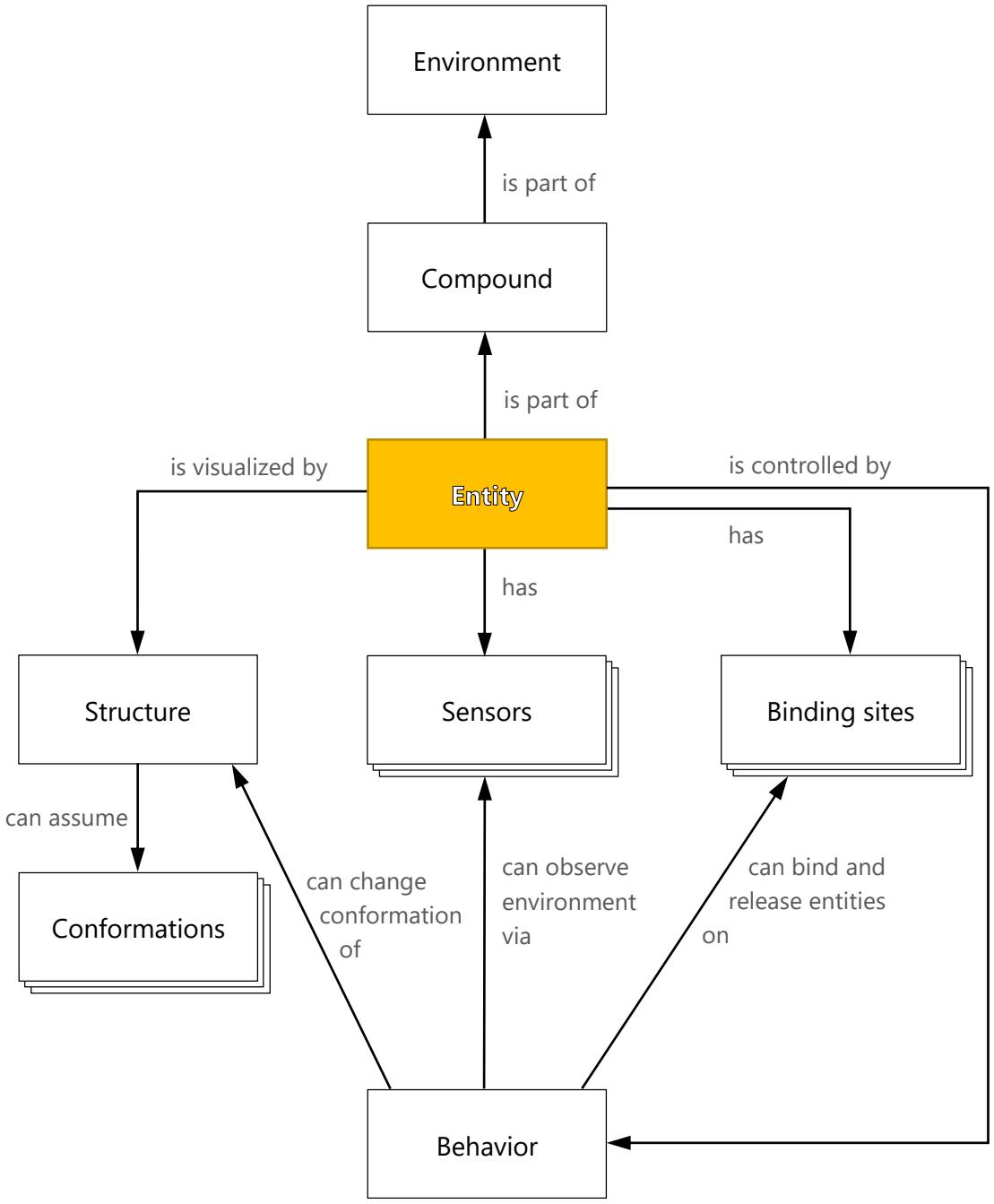


Figure 4.2: Overview of model components. Relations are depicted as arrows.

for each type of entity. This behavior description could, for example, involve scanning the neighborhood for possible reaction partners and, if found, attracting them. The animation is then generated according to this high level behavior description.

Structure and Conformation: The atomic structure of a molecule is very important in biology. Depending on its size and shape, it can interact with certain other molecules [AJL+14]. This atomic structure is reflected in the entity structure in the model. As described in Chapter 2, the shape of certain molecules can shift slightly when it interacts with other molecules, which is important for its function [AJL+14]. Such shapes a molecule can assume, are represented in the model as structure conformations. An entity can define a set of predefined conformations that its structure can assume. In biology, the structure changes to different conformations, because of the interaction with other molecules. In the model, the structure can be changed by the entity behavior. The behavior may decide to change the structure to another conformation when specific conditions are met.

Sensor: Other than in biology, in the model the entities are controlled by behavior descriptions. The behavior has to be aware of other entities nearby, in order to be able to interact with them. Therefore, sensors are introduced to the model. A sensor allows the entity behavior to detect other nearby entities in a specific range. It can then decide if it wants to interact with one of them or not. In biology, the interaction of molecules is vital, but molecules do not have sensors. Interactions are a result of physical attraction and repulsion forces. This model does not incorporate this automatic recognition phenomena, because it would require some form of biological simulation which takes away control from the animator. Instead the concept of a sensor is utilized, which allows the animator to mimic this physical behavior explicitly.

Binding Site: Another very important phenomena in biology is, that molecules dock or bind together to perform a task. Molecules have areas with a specific shape and charge profile where other molecules with complementary shape and charge profile can bind to [AJL+14], as described in Chapter 2. In the model, such areas are represented as binding sites. Each entity can include multiple binding sites. The behavior can control which entities should bind together on which binding sites. It can also decide to release a bond again.

Compound: Entities that are bound together on their binding sites form a compound. A compound can be a complex network consisting of multiple entities that are directly or indirectly interconnected. Just like molecular complexes in biology, compounds act as cohesive units.

Environment: To have a defined space, where all animations happen, the model defines an environment. The environment includes everything that should be visualized. It includes all entities, all behavior descriptions, all 3D models and everything else needed

for the visualization. The environment can be seen as the equivalent to a scene in standard 3D software.

4.2 Entity Overview

Entities are the central objects in an environment. Each entity acts as an agent and can operate individually, according to a defined behavior. Furthermore, entities can have binding sites and sensors defined, that allow the entity to interact with the environment. Its visual representation is defined by its structure.

An entity is the smallest, indivisible molecular particle in the environment. That can be either a molecule, an atom or an ion. Molecules that can split into smaller parts, are not atomic and therefore not represented as single entity. Instead they are represented as compound of smaller entities, in order to being able to split apart. What the smallest, indivisible molecular particles are, is dependent on the environment. For example, an environment that visualizes a biological process that reduces ATP to ADP for power [AJL+14], could represent these molecules using two entities: ADP and phosphate. ATP would be represented as an ADP entity bound to a phosphate entity.

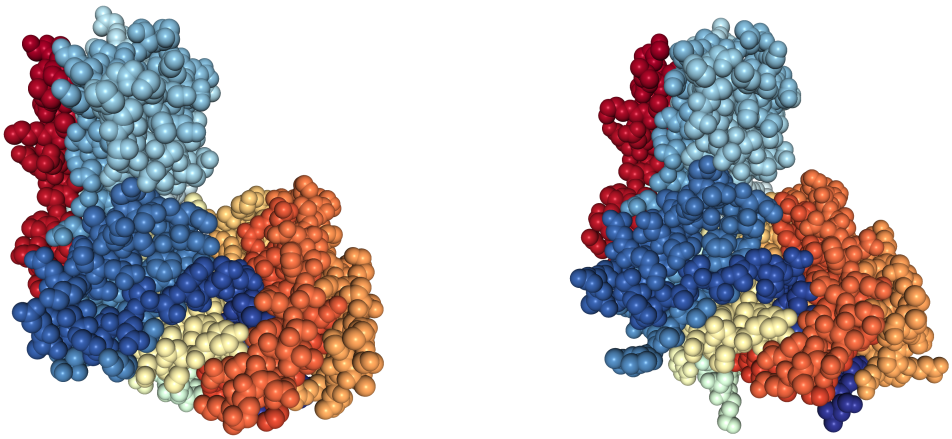
4.3 Entity Structure

The structure manages the visual representation of an entity. Entities represent molecular particles like molecules, therefore the entity structure is made out of atoms. The structure contains the element and the position of each atom of the entity.

As discussed in Chapter 2, certain molecules can change their conformation, which is a small shift in the molecular structure. Conformation changes occur, when the molecule is interacting with another molecule [AJL+14]. For example, the molecule hexokinase changes its molecular structure slightly when it is bound to a glucose molecule [FZC+15]. The ligand-free and the glucose-bound conformation are shown in Figure 4.3.

The structure of different entities may assume different conformations. To make conformation changes easy, the entity structure contains a list with all conformations the entity structure can assume. For each conformation, the exact position of every atom is stored. Additionally, a color and the locations and directions of binding sites and sensors are stored in each conformation. As a result, conformation changes do not only adjust the atoms, but also adjusts binding sites and sensors and can change the color of the entity for visual guidance.

When a conformation change is initiated, the structure is adjusted with the information stored in the target conformation. However, to avoid jumpy changes, the transition between two conformations is animated for a user defined amount of time. Atom positions, binding site and sensor positions as well as binding site and sensor directions are interpolated linearly. The red, green and blue components of the entity color are also interpolated linearly.



(a) Hexokinase structure in ligand-free conformation

(b) Hexokinase structure in glucose-bound conformation

Figure 4.3: Two slightly different conformations of hexokinase. One can see the gap in the middle closes as hexokinase binds to glucose [FZC+15]

The behavior of an entity is responsible for the conformation changes of its structure. The behavior may induce conformation changes when certain conditions are met, for example, when another entity is bound to a specific binding site.

4.4 Bonds between Entities

Entities have the capability to bind together and form a larger complex. The entities can establish temporary or lasting bonds at their binding sites. A binding site can only be bound to a binding site of another entity. Therefore, entities with the ability to bind to other entities must have one or more binding sites.

In biology the binding sites are areas with a specific shape and charge profile [AJL+14], as described in Chapter 2. In proposed model, the complex biological representation is reduced to a single position and a direction. This information is stored in the entity structure and is specified relative to the entity origin, so the entity can move and rotate along with its binding sites. The position and direction is defined for each binding site and allows two entities to bind together in a well-defined manner. When two entities bind together, they are aligned in such a way that those directions of the binding sites point at each other and the positions of the binding sites match. They do not rip apart as long as they are bound. The alignment is illustrated in 2D space in Figure 4.4. In 3D space, there is still one rotational degree of freedom left, which is simply set arbitrarily, to have a well-defined binding behavior. The exact mathematical procedure is described in the next section.

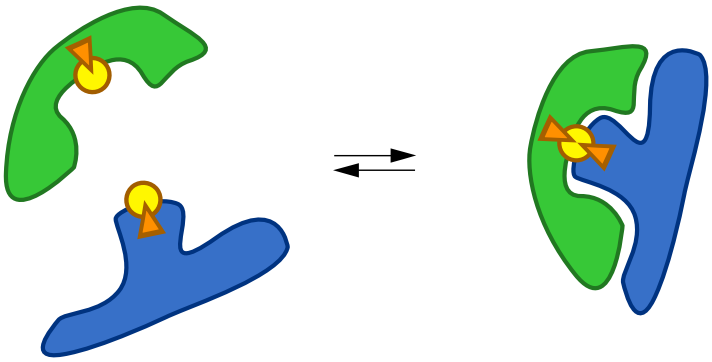


Figure 4.4: The binding site positions are depicted as yellow circles, the directions as orange arrows. When bound (right side), the positions match and the directions are pointing at each other.

4.4.1 Entity Alignment

Entities bound together on their binding sites form a well-defined complex. To achieve this, the bound entities are aligned relative to each other starting from an arbitrary root entity. The basic procedure for two entities that are bound together is explained in this section. The next section focuses on larger complexes.

The spatial location of an entity is represented by a 3D position vector and a quaternion indicating its rotation. The binding sites are defined by a 3D position vector and a quaternion as well, but the position and rotation is relative to the entity origin and stored in the entity structure. As these values are relative to the entity origin, they are referred to as local position and local rotation.

First of all, some basic functions must be defined that allow quaternions to be used as rotation representation in 3D space. To rotate a 3D position represented as vector using a quaternion, the vector itself has to be transformed to a quaternion first [DKL98]:

$$(x, y, z) \rightarrow x\mathbf{i} + y\mathbf{j} + z\mathbf{k} \tag{4.1}$$

The rotation quaternion can be applied to a 3D vector in quaternion form by the following formula, where p is the position, q is the rotation quaternion and p' is the resulting rotated position [DKL98]:

$$p' = rotate(p, q) = qpq^{-1} \tag{4.2}$$

It is also worth to mention that the quaternion q^{-1} represents the inverted rotation to q, which means that qq^{-1} equals the identity quaternion. Rotations of two or more rotation quaternions can be simply combined by multiplicating them [DKL98]. Therefore, the

global position $B.p$ and global rotation $B.r$ of a binding site B, where $B.lp$ is the local binding site position and $B.lr$ is the local binding site rotation and $E.p$ is the global entity position and $E.r$ the global entity rotation, is defined as:

$$B.p = E.p + rotate(B.lp, E.r) \tag{4.3}$$

$$B.r = E.r \cdot B.lr \tag{4.4}$$

Assuming there are two entities E_1 and E_2, which each has one binding site, which are referred to as $E_1.B_1$ and $E_2.B_2$ and they are bound together on those binding sites, the position and rotation of one entity can be derived from the other. At first, it is necessary to choose one entity to start with and derive the global position and rotation of the other entity. Here it is assumed that the position $E_1.p$ and rotation $E_1.r$ of entity E_1 are known, and the position $E_2.p$ and rotation $E_2.r$ of entity E_2 should be derived. Furthermore, the local position $B.lp$ and local rotation $B.lr$ of all binding sites are also known, as they are defined by the entity structure. As defined before, when two binding sites are bound, their global position matches:

$$E_2.B_2.p = E_1.B_1.p \tag{4.5}$$

Also, when two binding sites are bound together, their direction vectors are pointing at each other, as illustrated in Figure 4.4. The direction vector of a binding site with identity rotation is arbitrarily defined as the Y-axis vector $(0, 1, 0)$, or in quaternion representation: $\mathbf{j}$. The actual direction vector for any rotation can be determined by simply rotating the Y-axis vector using the rotation quaternion. In order that the two binding site direction vectors pointing at each other, the rotation $E_2.B_2.r$ must be the reversed version of $E_1.B_1.r$. The reversed version can be determined by rotating the direction vector around any orthogonal vector by 180°. All vectors on the orthogonal plane could be chosen as rotation axis, which results in one degree of freedom. The remaining degree of freedom has to be avoided to gain well-defined positions and rotations for bound entities. This is achieved by arbitrarily defining the Z-axis as rotation axis, which is orthogonal to the identity binding site direction vector $(0, 1, 0)$. Therefore, the rotation of $E_2.B_2.r$ is the combined rotation of $E_1.B_1.r$ and a rotation of 180° around the Z-axis, which is represented by the quaternion $\mathbf{k}$:

$$E_2.B_2.r = E_1.B_1.r \cdot \mathbf{k} \tag{4.6}$$

As the local position $E_2.B_2.lp$ and the local rotation $E_2.B_2.lr$ of binding site $E_2.B_2$ are known, the global position $E_2.p$ and global rotation $E_2.r$ of entity E_2 can be derived, by solving Equation 4.4 for $E.r$ and Equation 4.3 for $E.p$:

$$E_2.r = E_2.B_2.r \cdot E_2.B_2.lr^{-1} \tag{4.7}$$

$$E_2.p = E_2.B_2.p - rotate(E_2.B_2.lp, E_2.r) \tag{4.8}$$

The result provides the global position and global rotation of the entity E_2.

4.4.2 Compounds

All entities that are bound together are organized in compounds. Compounds act like one big entity, though with the ability to separate. Every entity is part of a compound, even when it is not bound to any other entity. In that case, it forms its own compound and is the only entity in it.

The entities in a compound cannot move freely. Their position and rotation relative to each other is well-defined by the position and direction of the binding sites on which the entities are bound together, as explained in Section 4.4.1. To enforce these constraints, the position and rotation of each entity is recalculated in every frame by an algorithm. The algorithm considers the whole environment as graph, where the entities are nodes and bindings between entities are edges, as shown in Figure 4.5. In this graph, each connected component represents a compound. When two entities bind together, the two compounds merge, as well as the connected components in the graph. When an entity unbinds, it leaves the compound and forms a new one, like the connected component in the graph divides [Wik16].

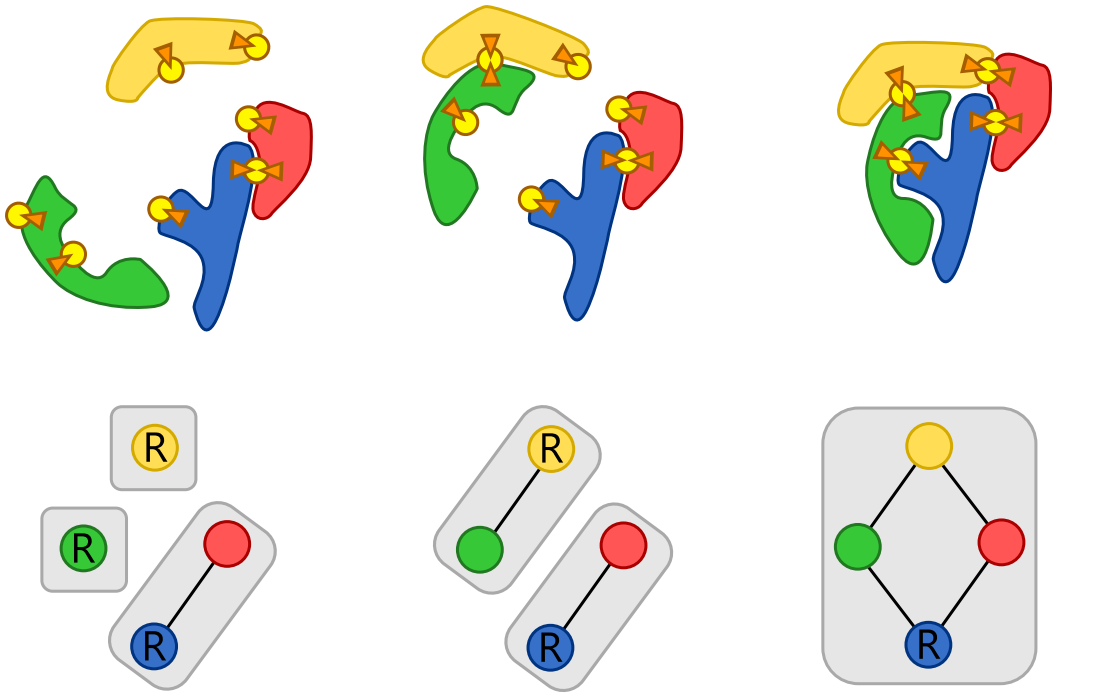
Each compound has an arbitrary root entity. When the entities are realigned, the algorithm starts from the root and traverses along all bindings to recalculate the position and rotation for each entity, with the formulas explained in Section 4.4.1. A connected component in the graph may contain loops, like shown in Figure 4.5c. To ensure the algorithm terminates, each entity is only handled once. The first position and rotation calculated are used for the entity, even if another path through the graph would result in slightly other values. The procedure is explained in Algorithm 4.1.

All compounds of an environment are stored in a list. This list is not static, but can change when bindings or unbindings occur. Therefore, the compounds are reevaluated when a binding site binds or releases. Only one bind or release event is handled at a time. In the event of binding, two distinct connected components in the graph merge to a single connected component. For example, the transition from Figure 4.5a to Figure 4.5b, where the yellow entity binds to the green entity. In the model, when two compounds are merged, all the entities from one compound are moved to the other. Then, the empty compound is deleted. If both entities that are binding together are already part of the same compound, no merging is required. This can happen, when entities bind together circularly, like shown in Figure 4.5c. The merging algorithm is summarized in Algorithm 4.2.

In the event of unbinding, one connected component in the graph may split. For example the transition from Figure 4.5b to Figure 4.5a, where the yellow entity releases the bond to the green entity. In the model, after a bond is released, all the entities of the former

Algorithm 4.1: Align all entities

```
1  Algorithm AlignEntities()
2      foreach compound in environment do
3          alreadyAligned = {}
4          AlignEntity(compound.rootEntity, compound.globalPosition,
               compound.globalRotation)
5      end
6      return
7  Procedure AlignEntity(entity, globalPosition, globalRotation)
8      entity.globalPosition = globalPosition
9      entity.globalRotation = globalRotation
10     foreach bindingSite in entity.BindingSites do
11         if bindingSite.isBound then
12             boundEntity = bindingSite.boundEntity
13             boundSite = bindingSite.boundBindingSite
14             if not alreadyAligned.contains(boundEntity) then
15                 newPosition, newRotation =
                       CalculateBoundEntityPositionAndRotation(
                       bindingSite.globalPosition, bindingSite.globalRotation,
                       boundSite.localPosition boundSite.localRotation)
16                 alreadyAligned.add(boundEntity)
17                 AlignEntity (boundEntity, newPosition, newRotation)
18             end
19         end
20     end
21 Procedure CalculateBoundEntityPositionAndRotation(
       rootBindingSiteGlobalPosition, rootBindingSiteGlobalRotation,
       boundBindingSiteLocalPosition, boundBindingSiteLocalRotation)
22     newRotation = rootBindingSiteGlobalRotation·k
23     newPosition = rootBindingSiteGlobalPosition -
           rotate(boundBindingSiteLocalPosition, boundBindingSiteLocalRotation)
           // see Equation 4.4 and Equation 4.3
24     return newPosition, newRotation
```



(a) Red and blue are bound together. Three individual compounds.

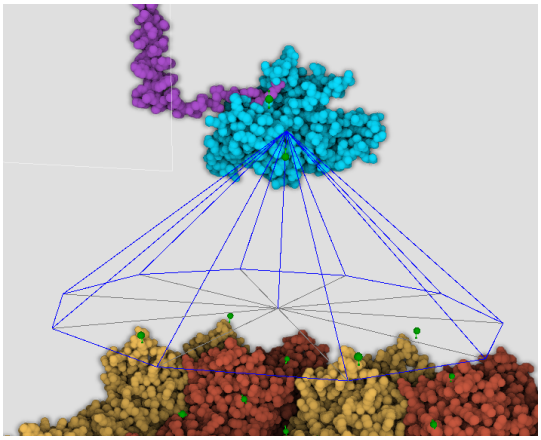
(b) Red and blue are bound together, and yellow and green are bound together. Two individual compounds.

(c) All entities are bound together to one compound.

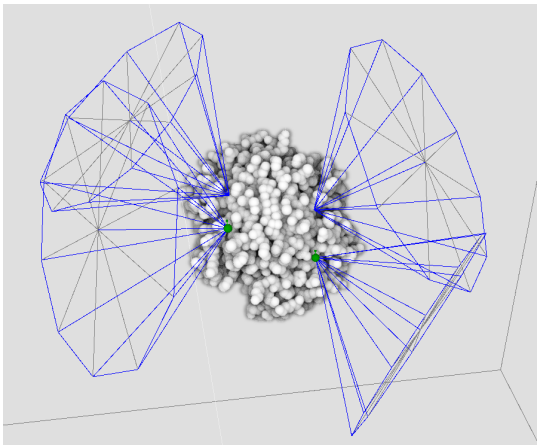
Figure 4.5: Four entities that are bound together in various ways. Illustrated in 2D and as graph representation (node color corresponds the entity color). In the graph, each connected component represents a compound and is shaded in gray. The root entity of a compound is marked with "R".

Algorithm 4.2: Merging of two compounds

```
1  Algorithm Bind(bindingSite1, bindingSite2)
2      if bindingSite1.ownerEntity.compound != bindingSite2.ownerEntity.compound
        then
3          compound = bindingSite1.ownerEntity.compound
4          oldCompound = bindingSite2.ownerEntity.compound
5          foreach entity in oldCompound do
6              compound.addEntity(entity)
7              entity.compound = compound
8          end
9          oldCompound.dispose()
10     end
```



(a) Sensor in kinesin head sensing for tubulin where it can dock to [KSM+97].



(b) Hemoglobin looking for oxygen with four sensors [FPSF84].

Figure 4.6: Examples of molecular machines models that utilize sensors

.

compound are reevaluated. Starting from the root entity of the compound, all bindings are traversed to determine which entities are still part of the compound. All entities, that are not accessible from the root entity, are then removed from the former compound. Then, a random remaining entity is chosen as root entity for a new compound. Again, from the new root all bindings are traversed to determine which entities are part of the new compound. All accessible entities are added to the new compound. This step is repeated until there are no unconnected entities left. The algorithm is summarized in Algorithm 4.3.

4.5 Sensor

Sensors are used by the entity behavior to find other entities that are nearby. The entity then may choose a nearby entity and interacts with it. Like binding sites, sensors are defined by a position and a direction. Furthermore, a sensor has an aperture and a range. The position, direction, aperture and range define a cone. The sensor can detect all entities that are inside that cone. Examples of sensor usage are shown in Figure 4.6.

The position and direction of a sensor are stored relative to the entity origin in the entity structure, just like binding sites. Thus, the sensor position and direction can change as a result of a conformational change.

As mentioned before, a sensor defines a cone to detect other entities. The defined cone is shown in Figure 4.7. In the entity structure, the local position $S.lp$ and the local rotation in quaternion form $S.lr$ is stored. This position defines the apex of the cone (point A in Figure 4.7). Therefore, the global position of point A, when $E.p$ is the entity position

Algorithm 4.3: Splitting of a compound

```
1  Algorithm ReleaseBond(bindingSite)
2      compound = bindingSite.ownerEntity.compound
3      otherSite = bindingSite.boundBindingSite
4      otherSite.boundBindingSite = null
5      bindingSite.boundBindingSite =null
6      unassignedEntities = copy(compound.entities)
7      entitiesConnectedToRoot = FindConnectedEntities
        (compound.rootEntity)
8      compound.removeEntities(unassignedEntities)
       // Find connected entities in the unassignedEntities list
          and create a new Compound for each connected
          component:
9      while unassignedEntities.count > 0 do
10         newRoot = unassignedEntities.first()
11         entitiesConnected = FindConnectedEntities (newRoot)
12         newCompound = new Compound(newRoot)
13         foreach connectedEntity in entitiesConnected do
14             newCompound.addEntity(connectedEntity)
15             connectedEntity.compound = newCompound
16         end
17     end
18 Procedure FindConnectedEntities(entity)
19     unassignedEntities.remove(entity)
20     connectedEntities = { entity }
21     foreach bindingSite in entity.bindingSites do
22         if bindingSite.isBound then
23             boundEntity = bindingSite.boundEntity
               // this "if" avoids infinite loop in circular
                  graphs:
24             if unassignedEntities.contains(boundEntity) then
25                 connectedEntities.addAll(FindConnectedEntities(boundEntity))
26             end
27         end
28     end
29     return connectedEntities
```

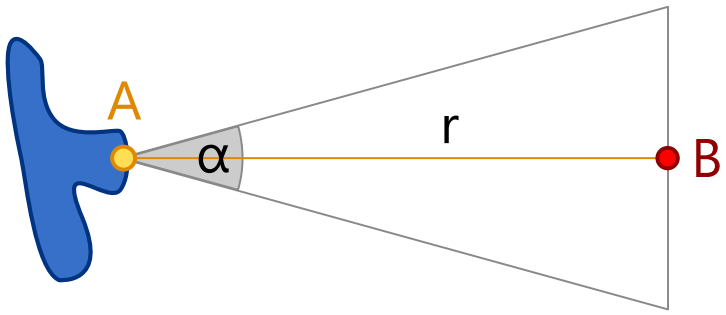


Figure 4.7: Entity with a sensor attached. The cone represents the sensing area. A is the apex, α the aperture, r the range and B the center of the base circle.

and $E.r$ the entity rotation, is defined by:

$$A = E.p + rotate(S.lp, E.r) \tag{4.9}$$

Point B is located in the middle of the base circle of the cone. B is dependent on the sensor direction and the range. The sensing direction for identity rotation is arbitrarily defined as the direction of the y-axis. The unit vector in direction of the y-axis is $(0, 1, 0)$, this equals in quaternion representation: $\mathbf{j}$. Therefore, when the sensor range is r, the global position for B is determined by:

$$B = A + rotate(\mathbf{j} \cdot r, E.r \cdot S.lr) \tag{4.10}$$

Every entity that is located inside the defined cone, is detectable by the sensor. To find relevant entities, it is tested for each entity if its origin is inside the cone. Thus, the testing algorithm consumes an arbitrary point P and returns *true* if the P is inside the cone and $false$ otherwise. When P is inside the cone, the angle between the vector from the apex to the base circle center ($\overrightarrow{AB}$) and the vector from the apex to P ($\overrightarrow{AP}$) can be $\frac{\alpha}{2}$ at maximum. If this condition is not fulfilled, then P cannot be inside the cone. If the condition is fulfilled, P is either inside the cone or is somewhere too far away, where the cone already has ended. To find out, $\overrightarrow{AP}$ is projected on $\overrightarrow{AB}$. If the projected length is r at maximum, then P is definitely inside the cone, if it exceeds r, P is definitely not inside the defined cone [Wei15]. The exact algorithm is shown in Algorithm 4.4. Note that "$\cdot$" indicates the dot product and $|\overrightarrow{AB}| = r$.

A sensor only detects entities on demand. The entity behavior controls the sensors and can use them when required.

Algorithm 4.4: Test if a point is inside a cone

Algorithm `IsPointInsideCone(`P`)`
  $\overrightarrow{AB} = B - A$
  $\overrightarrow{AP} = P - A$
  $\beta = \arccos\left(\frac{\overrightarrow{AP} \cdot \overrightarrow{AB}}{|\overrightarrow{AP}||\overrightarrow{AB}|}\right)$
  if $\beta > \frac{\alpha}{2}$ **then**
    return false
  end
  $projectedLength = \frac{\overrightarrow{AP} \cdot \overrightarrow{AB}}{|\overrightarrow{AB}|}$
  if $projectedLength > r$ **then**
    return false
  else
    return true
  end

4.6 Environment

As mentioned at the beginning of this chapter, the environment is the space where all animations happen. It is the model component that manages every entity, every structure and every compound. It also ensures that compounds are held together, and initiates all animation steps. The spatial boundaries of the animated molecular world are also defined by the environment.

When an environment is created, all different classes of entities must be registered to the environment. The environment is then able to instantiate individual entities. When an entity is created, its structure and behavior is created simultaneously by the environment. The environment holds a list of all structures, all entities and all compounds for fast access. The entity structures are updated in every frame, if they are currently changing conformation. Also the behaviors of all entities are invoked every frame, so they can react to changes since the previous frame, and take corresponding action. After that, the entity positions in all compounds are adjusted using the algorithm discussed in Section 4.4.2.

4.6.1 Compartments

Some animations require the environment to be subdivided into multiple areas, where entities cannot randomly escape. This areas are called compartments and are needed, for example, to model an environment that is divided to outside membrane enclosed area and inside membrane enclosed area. A screenshot of an environment consisting of two compartments is shown in Figure 4.8. Entities cannot randomly move outside their

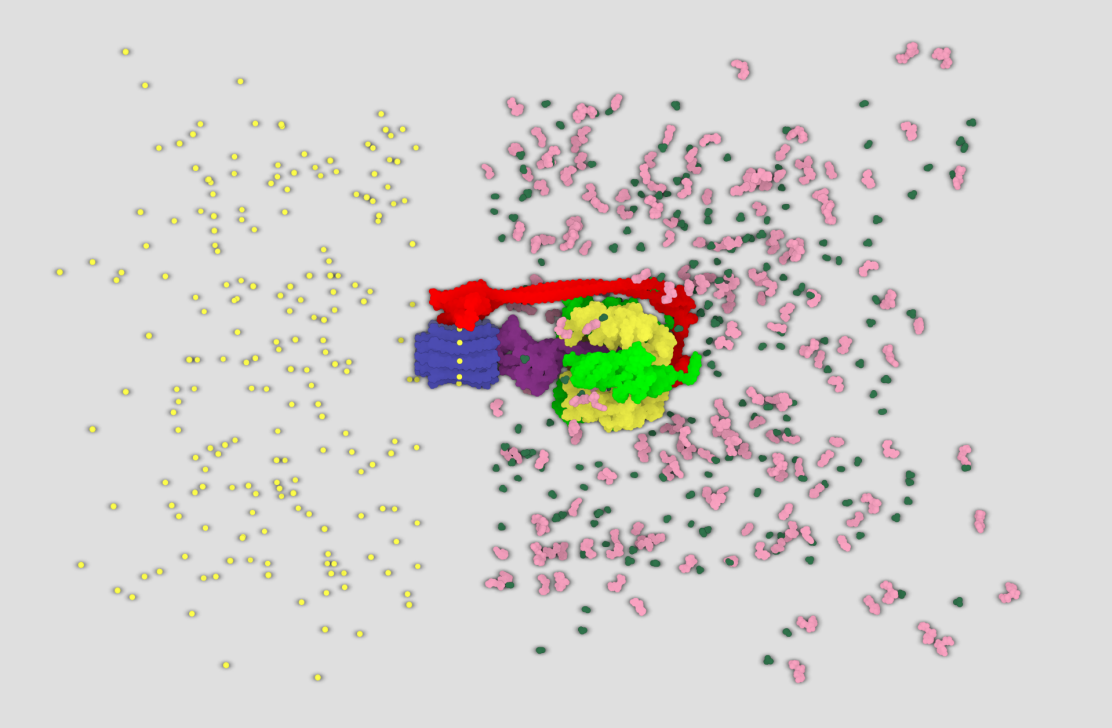


Figure 4.8: Screenshot of an environment divided into two compartments. Each compartment contains a few hundred entities. The two compartments are connected via a molecular machine that moves entities across compartments [SSW+16].

compartments by Brownian motion, but need to be explicitly moved across compartments by an entity behavior. The model only supports cuboid-shaped compartments.

4.6.2 Concentration Controllers

Concentration controllers allow the user to interactively change the concentration of a specific entity in the environment. The concentration controllers are represented as sliders on the GUI. By moving the sliders, the number of entities of a specific class is adjusted during the animation. An example is shown in Figure 4.9. It is possible to define multiple concentration controllers, with each dedicated to one specific entity class.

Each concentration controller has a customizable name and is bounded between a minimum and maximum value. When the slider on the GUI is adjusted by the user, the difference between desired and existing number of entities is determined. If the difference is positive, new entities have to be created. The animator can define how a new entity is created and where it is placed by a custom factory algorithm. This factory algorithm is executed as many times as number of entities are missing. Conversely, if the difference is negative, there are too many entities of the specified class in the environment. In such case, random entities of the specified class are deleted, until the desired and actual

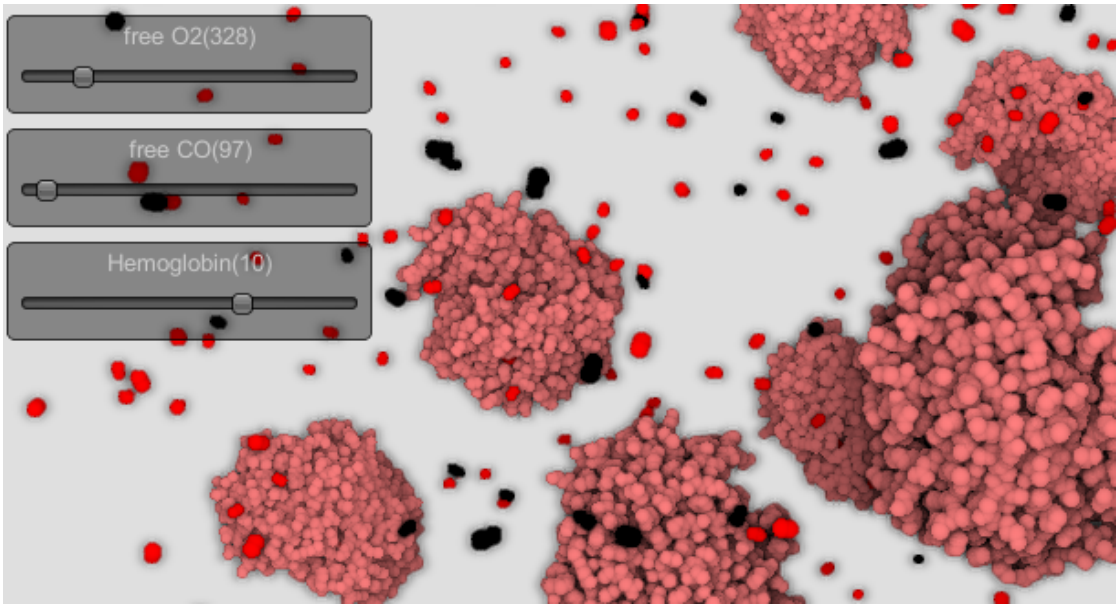


Figure 4.9: Screenshot of an environment showing three concentration controllers on the top left. By moving the slider the number of entities of a specific class interactively changes [Sha83].

number of entities equal. If the entity, that is about to get deleted, is bound to any other entity, the bond is automatically released.

4.7 Spatial Location

All entities are part of a compound, even if they are not bound to any other entity. When multiple entities are bound together, they do not move independently, but together as a compound. Therefore, the spatial location in the environment is handled per compound and not per entity. As previous sections indicate, entities of course have a position assigned, however, the individual entity positions are not independently mutable, because they are derived from the compound location. Changing the entity position property does not actually move the entity, because the value is recalculated and overwritten every frame. Instead, entities have to move with their compound. The entity position is only cached after calculation, so it does not have to be recalculated for further uses like, for example, rendering.

The position and rotation of a compound is determined by its spatial link. Different compounds can have a different types of spatial links to the environment, depending on their task. The concept of different spatial link types was chosen to give the animator more control over the compound movement. The model defines three types of spatial links:

Floating: A compound that is floating, moves randomly to mimicking Brownian motion. Floating compounds must always be located inside a compartment. They cannot leave the compartment automatically, because they bounce off the compartment walls. When floating, the collision detection is active, which means that the compounds bounce off other compounds. They are also pushed away by moving compounds that are following a trajectory.

Trajectory: The compound is moved along an arbitrary trajectory. A trajectory can be composed of multiple sections. There are two types of sections: linear movement and attraction force.

A linear movement section moves the compound to a defined destination position and rotation in a specified amount of time. After the time elapsed and the compound has reached its destination, the trajectory continues with the next section.

An attraction force section applies a physical force at an arbitrary position on the compound that pulls into the direction of a specified destination. The section is finished when either the distance to the destination is below a specified threshold, or a custom amount of time has elapsed.

Those two section types can be combined multiple times and in any order to define a trajectory. At the end of a trajectory, the spatial link is changed to either fixed or floating. Alternatively, the trajectory ends by binding to a binding site.

The primary use case for trajectories is to attract other compounds to eventually bind to them. Another use case is to move a formerly floating compound across compartment boundaries, because trajectories are not restricted by those. This is, for example, useful for membrane proteins. The animator can decide if the collision detection of the compound should be active or inactive during the movement. When the compound is moving through a compartment, it is usually desired to have an active collision detection, so floating compounds in the compartment collide and get out the way. Otherwise, with deactivated collision detection, the moving compound would just float through them. In contrast, when attracting a compound to bind it to an inner binding site, the collision detection has to be deactivated so the attracted compound can penetrate.

Fixed: A compound with a fixed spatial link, is locked at a certain position with a certain rotation and cannot be moved. This type of spatial link may be used to locate big or heavy compounds that do not move. It can also be used to place a compound between two compartments, so it can interact with entities from both sides. This is suitable, for example, for entities representing membrane proteins. Membrane proteins in biology can float along the 2D membrane surface, however, such a representation is currently not supported by the model [AJL+14].

4.8 Behavior Description

The behavior is the agent logic of an entity. Each entity is linked to a behavior which controls its interaction with the environment. By interacting with the environment the animation is created, because the interactions result in movements and conformation changes. The animator describes an entity behavior by program code. Behavior descriptions can include observing the environment, manipulating inner state, making decisions and interaction with the environment.

The behavior can observe the environment using the sensors attached to the entity. Sensors can be used determine the concentration of a specific ligand or to find nearby entities. The information obtained from the sensors must then be processed and evaluated. The behavior may contain an inner state that is updated, depending on this information. The inner state and the sensor information are then used to interact with the environment. Often, entities look for nearby entities of a specific class, with which they can interact. The behavior can attract or repel those entities to mimic physical forces, by using trajectories. Bindings on the entity's binding sites can be established or released. And the conformation of the entity structure can be changed.

The behavior description defined by the animator is executed by the environment for every frame. This enables the behavior to evaluate the environment and take action in a periodic manner. Since each entity has a behavior description that is executed every frame of the animation, the program code describing the behavior should be efficient and as short as possible. The longer it takes to execute all behavior descriptions, the lower is the frame rate of the resulting animation.

Often, the molecule behavior depends on its current conformation. For example, enzymes that are allosterically activated behave different than the same enzyme that is not activated. Therefore, the behavior can often be described as state machine. Depending on the current state, the behavior interacts differently with the environment.

The behavior has full control over other entities and a toolset of interaction possibilities like trajectories and conformations. As a result, it is possible to create animations that are not necessarily compliant with laws of nature. For example, the principle of conservation of energy is easily violated. Another example is when two binding sites bind, it is not checked if they are compatible in a complementary shape and charge way, like it is in biology. Therefore, the behavior description is responsible to keep animations as realistic as required.

4.9 Methodology

To learn about the challenges of the animation process, three molecular machines were visualized. For this learning process, none of the import or animation tools described in the related work section were used. The reason for this design choice is to get an in-depth understanding of the challenges involved and avoid being biased by possible solutions.

The goal was to create a framework that can be used to animate a variety of molecular machines. To achieve this, the framework was created and evolved iteratively by following these steps:

- Choose a simple molecular machine to start with. Get familiar with it and its functionality.
- Create a framework that can be used to visualize and animate the machine. Make sure that the framework supports all tasks necessary, from importing molecular data, to generating a model for the chosen molecular machine, to render the machine. Also take care the framework does not require the user to do repetitive tasks. Repetitive tasks often expose as molecular features that can be abstracted and reused.
- When the chosen molecular machine is sufficiently represented, choose another more complex one. Then repeat these steps with the new machine, but instead of creating a new framework, advance the existing one, if necessary.

This approach is also called "Harvested Framework" [Fow03]. When repeated for a few iterations, the resulting framework should support quite a variety of molecular machines. Ideally, the framework should not require new features after a few iterations, because all features needed are already implemented and can be reused for the new machine. When this happens, the framework has reached a certain matureness and should be usable for a wide range of molecular machines.

To find a suitable first molecular machine to start with, various sources were considered, including the books "Molecular Biology of the Cell" by Bruce Alberts et al. [AJL+14] and "Bionanotechnology: Lessons from Nature" by David Goodsell [Goo04]. Another great source was an extensive list of molecular machines, published by Casey Luskin [Lus10]. After some research, Hemoglobin was chosen as very first molecular machine. It is relatively simple, well known and easy to understand.

Initially, a mechanical animation approach was considered. As it turns out, this approach is not suitable for many molecular machines. Thus, finally the agent-based approach that was discussed in the previous sections was implemented. The following sections describe the two approaches in more detail.

4.9.1 Mechanical Approach

Molecular machines have some analogies to machines in the macroscopic world [BBC+01]. Thus, the first idea was that some molecules may act like macroscopic machine elements. Assuming there is a strong relation between certain molecules and macroscopic machine elements, a mapping between those worlds could be established as illustrated in Figure 4.10. For macroscopic machine elements, it is possible to infer the interactions of the machine elements by their structure, as shown by Mitra at al. [MYY+10] If a similar approach

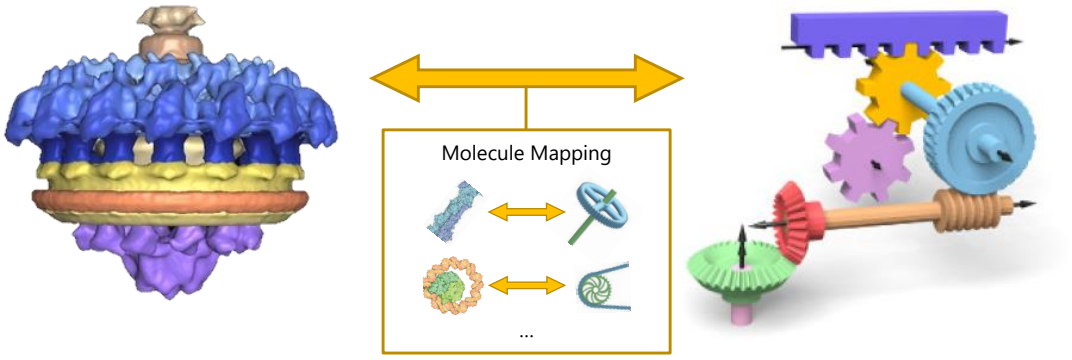


Figure 4.10: Mapping between molecular representation and machine representation by mapping molecules to corresponding machine elements [MYY+10, ZNL14, GVL14].

could be used to for molecules, then even an automatic mapping would be imaginable. Using the mapping knowledge, a mechanical model could be generated out of a molecular machine. A mechanical model would be very illustrative to demonstrate how a molecular machine works. Furthermore, the mechanical model can be animated, as demonstrated by Mitra at al. [MYY+10]. The movements of the mechanical animation could then be mapped back to the molecular machine. Thus, the mechanical model would not only provide a different representation form, but would also help to animate the original molecular machine. Another advantage would be that the mapping could be used the other way around: By creating a mechanical model, a new molecular machine could be generated by using the mapping. Out of a library of known macromolecular machine elements and the corresponding molecules, new molecular machines could be designed.

Some molecular machines actually work in a very mechanistic way. For example, ATP-synthase or the Bacterial Flagellar Motor both contain rotatory parts [SSW+16, ZNL14]. However, the majority of molecules do not map to known mechanical parts very well. Furthermore, a lot of aspects of molecular machines work fundamentally different to macroscopic machines. The reason for this is the small scale and the very different environmental conditions [Goo04]. Gravity and inertia are very important for macroscopic machines, but are negligible for molecular machines. Thermal motion in not relevant for macroscopic machines, but vital for the functionality of molecular machines. The very crowded water environment in which molecular machines work are also quite different to the environment of most mechanical machines. More peculiarities of molecular environments are discussed in Section 2.4.1. Moreover, molecular machines do not work purely mechanical, but also utilize other approaches, like molecule interactions, structural changes, electric forces and chemical energy generation, just to name a few [Goo04].

Another fundamental difference is that macroscopic machines use very general elements that can be used for a lot of machines. For example gears, shafts, bearings, pins, etc. [MoME05]. In contrast, each part of every molecular machine is highly specialized

for its specific task. Thus, a specific molecule cannot be seen as a general machine element that can be reused for other molecular machines, but as a unique part that is only usable for one machine. However, there are molecules that are used in various molecular machines, like ATP, but the molecular machine elements usually only exist in one machine [Goo04].

As it turns out, a mechanical description is not very appropriate for most molecular machines. Already the first chosen molecular machine (hemoglobin) could not be represented by a mechanical model. Therefore, another more low-level approach was chosen, which is discussed in the next section.

4.9.2 Agent-based Approach

As molecular machines could not be mapped to macroscopic machine parts in a satisfying manner, molecular machines were examined on a lower level. Instead of trying to transform molecular machines to another concept, it was tried to isolate their traits and features. One obvious trait is that molecular machines are built out of molecules. These molecules interact with each other to fulfill their tasks. Since molecules in nature are not controlled by an omniscient intelligence, but behave dependent on their surroundings, the idea of autonomous molecules arose. Those autonomous molecules were used as agents and were equipped with agent logic that controls their behavior.

The molecular environment is represented using a model. Every action that takes place in the environment is initiated by an action on the model. Also the behavior descriptions use the model to interact with the environment. Like explained earlier, the framework was improved iteratively, by visualizing more and more complex molecular machines. A total of three molecular machines were visualized using the model presented in this chapter. During the visualizations the model needed to improve. At the very beginning, the relatively simple molecular machine hemoglobin was visualized. As second machine, a visualization of the more sophisticated ATP-synthase was created. The last molecular machine was kinesin. The biological functionality of these machines was already explained in Chapter 2. How the model evolved from machine to machine is presented in the next section.

Model for Hemoglobin

For the first molecular machine, the model had to be created from scratch. The model should be able represent hemoglobin, oxygen and carbon-monoxide molecules. Moreover, the hemoglobin should automatically bind to oxygen or carbon-monoxide depending on the concentration of these molecules.

At first, an underlying model component was introduced that can represent the molecules. Since hemoglobin is a single molecule, the general molecule component can be used to represent it as well. Each molecule is represented by a set of atoms.

The essence of hemoglobin is that it can bind and release oxygen molecules. Since hemoglobin has four sites where the oxygen molecules can bind to, and all of these sites do exactly the same, a general abstraction for binding sites was introduced. At the beginning, those binding sites were much simpler than the more matured concept explained in Section 4.4. The early binding sites were only suitable to attract and hold smaller molecules. They were only defined by a position, without direction. It was not possible to build molecular complexes and compounds using them. But for the hemoglobin, this was not a requirement.

Hemoglobin binds to oxygen or carbon-monoxide depending on the concentration of these molecules in the surrounding. Likewise, these molecules are likely to be released again, when the concentration in the surrounding drops below a certain threshold. To enable the hemoglobin to decide when to bind and when to release a molecule on a binding site, it has to know the concentration. Sensors were introduced to the model to address this requirement. The sensors were designed to only analyze small, cone-shaped areas, so each binding site can have its own sensor. Each binding site having its own sensor has a few advantages compared to just search entities that are inside a defined radius: First, the binding sites can be handled individually, so the exact same logic can be used for all of them. And second, by using individual cone shaped sensors, the molecules in the sensor area are in the "field of view" of the binding site. When a molecule from the sensor area is attracted to the binding site, it seems more natural, than choosing a random molecule nearby the hemoglobin, that is possibly out of "sight" for the binding site.

Hemoglobin has a dynamic affinity for oxygen. The more oxygen molecules are already bound, the easier it is for the next oxygen molecule to bind. This increasing affinity is caused by allosteric activation, which is noticeable by small shifts in the atomic structure. Therefore, it was necessary to consider changes in the atomic structure in the model as well. To accomplish this, the concept of different conformations was introduced, as explained in Section 4.3.

For the hemoglobin visualization, also a simple environment component was introduced to the model. The environment was responsible for keeping a list of all molecules and to limit the space where the molecules can float. Concentration controllers also were available from the first molecular machine, to control the oxygen and carbon-monoxide concentration in the environment. The method of describing behavior was very similar to the method described in Section 4.8. The only changes in the behavior description was the model that improved, and therefore provided more possibilities to interact with the environment, and the more comfortable access to certain components due to dependency injection, which is described in Section 5.5. The model components that were introduced for the hemoglobin visualization are shown in Figure 4.11.

Model for ATP-Synthase

Like in hemoglobin, some parts rely on conformation changes to perform their tasks, for example, the F_1 complex that joins ADP and phosphate back together. Also, binding

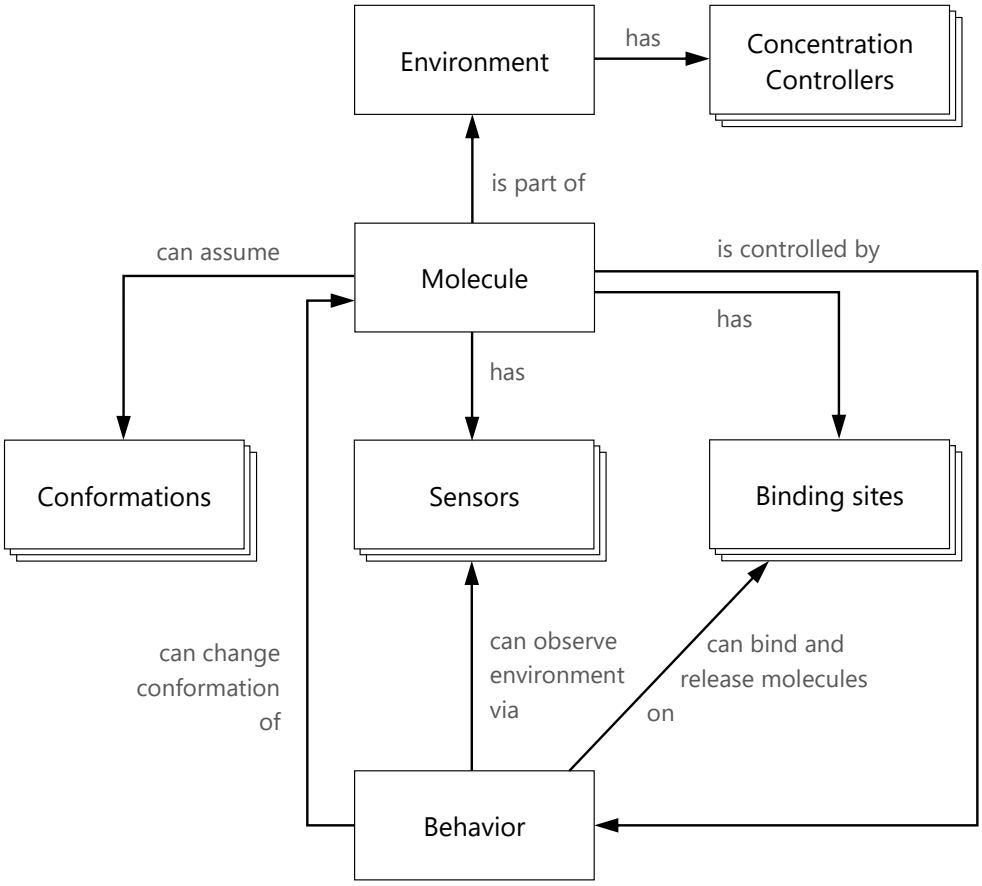


Figure 4.11: First model that was used to visualize hemoglobin. All components were created from scratch.

sites for protons, ADP and phosphate were necessary for ATP-synthase. To find free ligands, the concept of a sensor was reused, similar to the usage in hemoglobin. Behaviors to control the machine parts, and concentration controllers to influence the proton concentration were used for the ATP-synthase as well. Thus, those components of the previous model could be reused. But yet a few adjustments and extensions had to be made, to being able to visualize the ATP-synthase.

ATP-synthase is a molecular machine that consists of multiple molecules. Some of those molecules change their conformation independent from each other and some molecules even rotate as part of the machine activity. Still, the whole molecular complex holds together. The model created for the hemoglobin is not quite capable of representing such a molecular complex, since it does not allow large molecules to bind together in a well-defined manner.

ATP-synthase is driven by protons trying to get from outside a membrane to inside a membrane enclosed space. To avoid that the protons get to the inside themselves, those

spaces must be strictly separated, which is not supported by the old model. The division of the environment, again, presents new challenges, because then a method is needed to move molecules across those divided areas.

To allow the construction of molecular complexes, the binding sites were extended by a direction, so bound together molecules can align in a well-defined way, as explained in Section 4.4.1. The bound together molecules are spatially not independent anymore. Therefore, whereas those molecular complexes consist of multiple molecules, a complex has to be treated like one big unit. To achieve this, the concept of a compound was introduced to the model, as discussed in Section 4.4.2.

For the visualization of ATP-synthase it was necessary, to combine ADP and phosphate to ATP. For this task, the same approach was used as combining molecules to molecular complexes. In biology, ATP is a single molecule, in this ATP-synthase visualization, the ATP is represented as a compound of two smaller molecules. Furthermore, it was necessary to visualize protons, which are, strictly speaking, no molecules. Due to this reasons, the molecule component was renamed to entity, to clarify that it is a more general component than a molecule.

As mentioned before, the environment needed to be divided into two parts. To achieve this, compartments were introduced to the model, as explained in Section 4.6.1. To be able to move compounds across compartments, the concept of compartment independent trajectories was used. Furthermore, it was necessary to fix the ATP-synthase between two compartments and let the other molecules float freely. Thus, also the concepts of fixed and floating compounds were added. The location controlling parts of the compounds were then abstracted to a spatial location component in the model, as explained in Section 4.7.

Another change in the model was that the visual representation of an entity was summarized to an entity structure. The extensions and changes made to the previous model that originated from the hemoglobin, are summarized in Figure 4.12.

Model for Kinesin

In the visualization of kinesin, not only the kinesin itself is visualized, but also a microtubule. The microtubule consists of multiple subunits that are bound together. To enable such a large complex of molecular subunits, the compound feature was reused, that was introduced for the ATP-synthase. Also the kinesin itself is composed of multiple parts that are bound together by binding sites and therefore forming a compound.

The microtubule is fixed in the environment. The kinesin is floating as long as it is not bound to the microtubule. Thus, the spatial location feature that was introduced with the ATP-synthase, could also be reused for kinesin.

The ability to change the conformation that was introduced for the hemoglobin, could also be utilized for the kinesin visualization. The kinesin heads are changing their

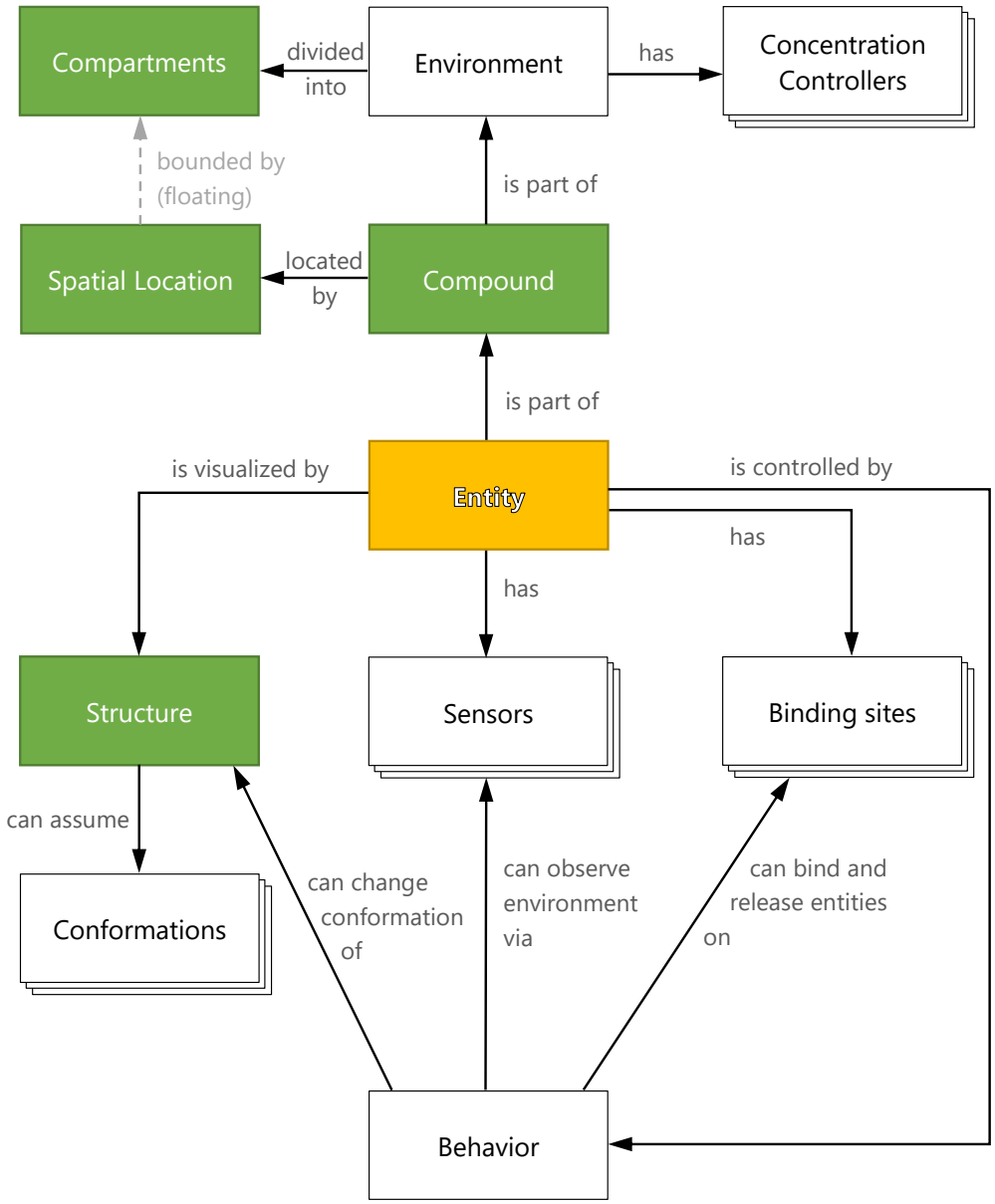


Figure 4.12: Model used for ATP-synthase. The components that differ from the previous hemoglobin model are colored. The green colored components were added, the yellow shaded component was renamed.

conformation when binding to the microtubule, when ATP is consumed and when ATP is cleaved.

The model of kinesin also makes use of a sensor to find a microtubule unit to which it can bind to. Of course also the entity behavior was needed, to describe the molecular machine's behavior.

The kinesin did utilize most of the features that were added to the model for the previous machines. However, one exception are the compartments. This feature is not needed by kinesin, because everything takes place in the same, undivided space. Also, kinesin did not require to extend or modify the model. That shows that the model already gained a certain maturity, and that the introduced concepts for one molecular machine are likely to be reused for other machines.

CHAPTER 5

Implementation Details

A prototypic framework was implemented according to the model proposed in Chapter 4. The framework provides visualization and animation capabilities for virtual molecular environments. This chapter summarizes the implementation details of this prototype.

5.1 Platform and Architecture

The framework is implemented using the programming language C#. It is based on the platforms: Unity 5.4.3 [Tec17] and .NET Framework 3.5 [Mic17]. The framework is divided into two parts:

The first part is the library. It is a .NET library [Mic17] which contains the model discussed in Chapter 4 and implements its basic logic. The library provides an application programming interface (API) that allows the user to create a custom molecular environment. The created environments can be saved and loaded in a JSON file format [Bra14] that is further discussed in Section 5.3. The library is based on .NET Framework 3.5, even though newer .NET Framework versions are available. The reason for this is that it must be possible for the library to be used in Unity and Unity only supports .NET Framework libraries up to version 3.5 [Tec17].

The second part is the player. The player is implemented as Unity project [Tec17] which extends the library to add Unity's physics engine and rendering capabilities of the framework. Unity renders and animates the created molecular environments in real-time. Using mouse and keyboard, the user can move around in 3D space and see the molecular machines working. The user can interact with the molecular environment by changing the ligand concentration using concentration controllers and can experience the effects of changes in real-time.

Although the whole framework could have been implemented in Unity without a separate library, the project is divided in its two parts to provide an API independent from Unity that can be used by console application for import scripts, as described in Section 5.2.

5.2 Data Import

Before a molecular environment can be animated, it has to be modeled using the framework. To create a molecular environment, data from different sources has to be combined. The data has to be imported and transformed to fit the model. The framework exposes an API for that purpose.

The suggested way to create a molecular animation using the framework is by creating an import script. An import script is composed of import and transformation instructions to create a molecular environment. At the end of the script, the environment can be saved to JSON files that can be read by the player. By using import scripts, the user can iteratively adjust details in the script and obtain reproducible results by rerunning the script. Import scripts can be easily made by creating a .NET console application and reference the library of this framework. The library provides all the functionality needed to import and transform data from the Protein Data Bank [BWF+00], create compartments, entities, bindings, etc.

The framework includes a parser for PDBx/mmCIF files, which is the standard archive format for the PDB [wwP16]. Using the parser, structures from the PDB can be imported and used in the environment. The framework also provides inherent support for spatial manipulation via 3D vectors and quaternions.

The framework provides some helper classes like `ImportBase` and `EntityBuilder`. They make it easy to import data and create a molecular environment. Listing 5.1 shows a simple example how to create an entity class and import the molecular structure from a PDBx/mmCIF file. Then 100 instances of this entity are placed in a compartment.

5.3 Data Format

A molecular environment created with the framework can be saved and loaded using JSON files [Bra14]. The format can be used by import scripts to save the created environment. To animate the environment, the player can load the JSON files and play the animation. By using this data format, the data import and the animation player are strictly separated. This allows the user to save different molecular environments without having to reimport from the original source files before each usage. That makes it easy to quickly switch between different molecular environments and improves the startup time of the player.

When an environment is saved, each class of entity that is present in the environment is serialized into a separate file. A specialized environment file saves the individual entities, compartments and concentration controllers. The data format relies heavily on IDs to

```
1 public class ImportExample : ImportBase
2 {
3   public override void Import()
4   {
5     var compartment = new Compartment(
6       "main",
7       new Vector(-250, -250, -250),
8       new Vector(250, 250, 250)
9     );
10    Environment.AddCompartment(compartment);
11
12    var hemoglobinClass = new EntityClassBuilder("hemoglobin");
13    hemoglobinClass.AddAtoms(LoadCif("2hhb.cif"));
14    hemoglobinClass.ReCenter();
15
16    for (int i = 0; i < 100; i++)
17    {
18      CreateEntityFloating(hemoglobinClass, compartment);
19    }
20  }
21 }
```

Listing 5.1: Import script example

wire different components together. When the player is started, it simply loads all JSON files in a specified directory and animates the loaded environment.

Entity Class File

An entity class file includes the definition of a specific class of entity. It basically provides a template for creating entities. In the environment, multiple instances of the entity class can be created. Entity class files are not bound to a specific environment. Thus, it is quite possible to reuse an entity class in other environments without any modification of the entity class file.

The entity class contains an ID, so it can be uniquely referenced by other files. A behavior-ID wires together the entity class with a specific behavior. The behavior description itself is not part of the entity class file. How behaviors are created and how they are associated with a behavior-ID is explained in Section 5.5. Next, the entity class contains the specification for its markers. "Marker" is an umbrella term for binding site and sensor. The largest part of the file, however, is used for the structure definition.

The structure definition contains a list with all atoms that make up the entity. Then the predefined conformations of the entity class are encoded. Each conformation has an ID, so it can be referenced from the behavior description. The conformation contains the 3D position for each atom. Furthermore, the position and rotation for each marker are defined by the conformation. Moreover, the conformation has a color assigned, and a set of collider geometries that are used to define the approximate entity boundaries, so the

physics engine can perform a collision detection. The collision detection is explained in Section 5.4 in more detail. An example entity class file is shown in Listing 5.2.

```
{
    "id": "kinesinHead",
    "behavior": "kinesin.head",
    "markers": [
        { "id": "tubulin", "type": "bindingSite" },
        { "id": "tubulinSensor",  "type": "sensor", "range": "70",
          "aperture": "1.8" },
        ...
    ],
    "structure": {
        "id": "kinesinHead1",
        "atoms": [ {"symbol": "N"}, {"symbol": "C"}, ... ],
        "conformations": [
            {
                "id": "bound tight",
                "color": "0.2737045;0.2919063;0.4673147;1",
                "atomPositions": [
                    "-1.010026;22.67282;18.10739",
                    "-1.04894;23.41682;16.81354",
                    ...
                ],
                "markerLocRots": [
                    {
                        "loc": "-15.93514;6.988817;-3.163164",
                        "rot": "0;-1;0;0"
                    },
                    ...
                ],
                "collider": {
                    "geometries": [
                        {
                            "type": "sphere",
                            "center": "0;0;0",
                            "radius": "36.22589"
                        }
                    ]
                }
            },
            ...
        ]
    }
}
```

Listing 5.2: Entity class file example

Environment File

The environment file defines a molecular environment and makes use of entity class files. It includes all compartments and their bounding boxes. Each compartment has an own unique ID. Next, all entities that are present in the environment are defined. Again, each entity gets an ID. Entities which are the root entity of their compound, additionally have a spatial location defined.

When the environment is loaded from the file, the entities are sequentially created. In order that compounds are preserved in the environment file, the bonds between entities are also saved. By sequentially binding them together as defined in the file, the original compounds are reconstructed. The bonds in the environment file are saved by referencing both entities and their binding sites.

The concentration controllers that are visible to the user, are also defined in the environment file. They are defined by a display name, a minimum and maximum value and the ID of the entity factory used by the concentration controller. More information about entity factories and concentration controllers can be found in Section 5.6.

An example environment file is shown in Listing 5.3.

5.4 Unity Integration

The player is implemented in Unity [Tec17] in order to harness its physics engine and render capabilities. The Unity project references the library to gain the functionality of the framework. The framework provides various hooks where it can be customized. Most framework classes can be extended to add new functionality. It is also possible to be notified on specific model events like bindings.

Unity's physics engine is used for collision detection. The physics engine is based on the GameObject model of Unity, therefore the framework model must be represented as GameObject model as well. To achieve this, the entity and compound components of the framework are extended to create corresponding GameObjects. The model components and GameObjects are tightly bound. The compound-entity hierarchy is preserved in the GameObject model. The parent of each entity GameObject is the corresponding compound GameObject, just like it is in the model. Additionally, the model uses the location information of the GameObjects. Thus, when the model component is moved, the GameObject moves as well and vice versa. In that way the physics engine can interfere with the model by using the GameObjects.

For collision detection, the physics engine should not consider all entities independently, but must handle the compounds as a whole units. This is vital, because forces applied by collisions should affect all entities in a compound and not only to the entity at the impact site. Besides, it is not desired that physical forces can rip an entity out of a compound. Even though this is a natural phenomenon in biology, it is not desired in

```
{
    "name": "Kinesin Environment",
    "compartments": [
        {
            "id": "main",
            "corner1": "-1000;-1000;-1000",
            "corner2": "1000;1000;1000"
        }
    ],
    "entities": [
        {
            "id": "274",
            "class": "kinesin",
            "spatialLocation": {
                "type": "floating",
                "locRot": {
                    "loc": "0;50;0",
                    "rot": "0;0;0;1"
                }
            }
        },
        { "id": "271", "class": "kinesinNeckLinker" },
        ...
    ],
    "bonds": [
        {
            "active": { "entity": "274", "bindingSite": "neckLinker1" },
            "passive": { "entity": "271", "bindingSite": "hip" }
        },
        ...
    ],
    "concentrationControllers": [
        {
            "name": "ATP",
            "entityFactoryId": "kinesin.atpEntityFactory",
            "min": "0",
            "max": "500"
        },
    ]
}
```

Listing 5.3: Environment file example

this framework. The aim of this framework is not to act as a simulator, but requires the biologist to explicitly define such behavior if wanted.

The physics engine only considers GameObjects with a Rigidbody component [Tec16]. Thus, only GameObjects representing compounds have a Rigidbody component assigned, but the GameObjects representing entities do not. To detect collisions, the physics engine combines all Colliders that are assigned to all sub elements of the Rigidbody GameObject [Tec16]. Thus, the GameObjects that represent entities get the Colliders assigned. Because the entity GameObjects are sub elements of the compound GameObjects, the physics engine combines all Colliders of the entities in a compound, and therefore treats a compound as a single physical object, which is exactly what is desired.

5.5 Behavior Description

The entities in a molecular environment interact with each other according to a specific behavior. The behavior is defined using a behavior description. In the framework, the behavior is described using C# program code. For each class of entity a separate behavior description can be provided. The behavior is described in a class that extends `EntityBehavior`. To make the behavior classes available to the player, all behavior classes must be compiled to a .NET library, which is added to the Unity assets of the player. When the player is started, all referenced libraries are searched for classes that extend `EntityBehavior` and uses them for the animation. As described in Section 5.3, the entity classes are linked to a behavior via a behavior-ID. The behavior-ID is added to a behavior class by the `EntityBehaviorId` attribute. An example is shown in Listing 5.4. At the beginning of an animation, for each entity a corresponding behavior object is instantiated, that controls the entity.

```
1 [EntityBehaviorId("atpSynthase.f0c")]
2 public class F0cBehavior : EntityBehavior
3 {
4   ...
5 }
```

Listing 5.4: Entity behavior definition with `EntityBehaviorId` attribute.

The behavior can be described by using the model components described in Chapter 4. To access the model components in code, the framework includes a dependency injection for those components. The needed components can be declared as properties with an appropriate attribute that tells the dependency injector what to inject.

The following attributes can be used:

Sensor(id): Injects the sensor with the specified ID into the property. The sensor object can be used to scan the surrounding, as described later. An example usage is shown in Listing 5.5.

```
1 [Sensor("tubulinSensor")]
2 public Sensor TubulinSensor { get; set; }
```

Listing 5.5: Sensor injection into a property.

BindingSite(id): Injects the binding site with the specified ID into the property. The binding site can be used to check if an entity is bound, or to initiate and release bonds. An example definition is shown in Listing 5.6.

```
1 [BindingSite("protonSite")]
2 public BindingSite ProtonSite { get; set; }
```

Listing 5.6: Binding site injection into a property.

BoundEntity(bindingSiteId): Injects the entity bound to the binding site with the specified ID. Depending on the type of the property, either the bound entity itself or its behavior is injected. This provides fast and easy access to bound entities. The property is automatically updated, when an entity is bound or released from the specified binding site. If nothing is bound to the binding site, the property value is `null`. An example definition is shown in Listing 5.7.

```
1 // inject entity
2 [BoundEntity("rotorSite")]
3 public Entity RotorEntity { get; set; }
4
5 // inject behavior
6 [BoundEntity("rotorSite")]
7 public RotorBehavior RotorBehavior { get; set; }
```

Listing 5.7: Bound entity and bound entity behavior injection into a property.

The examples above show how to gain access to the model components in a behavior description. In the following sections it is explained, how this model components can be used to observe the environment, how the inner state can be used and modified, and how the behavior can interact with the environment.

5.5.1 Managing Inner State

Entity behaviors are implemented as state machines. Each state of the state machine has a dedicated code block. The code in the code block has the option to observe the environment, change the state, or interact with the environment. In the behavior C# class, the code blocks are defined using ordinary, parameterless methods with the `State` attribute. The method representing the code block for the current state is called for

```
1 [EntityBehaviorId("atpSynthase.f0c")]
2 public class F0cBehavior : EntityBehavior
3 {
4   [BindingSite("protonSite")]
5   public BindingSite ProtonSite { get; set; }
6
7   [State(InitialState = true, Conformation = "tensed")]
8   public void Tense()
9   {
10    // Change to Relax-State as soon a proton is bound
11    if (ProtonSite.IsBound)
12    { SetState(Relax); }
13  }
14
15  [State(Conformation = "relaxed")]
16  public void Relax()
17  {
18    // Change to Tense-State as soon the proton leaves
19    if (!ProtonSite.IsBound)
20    { SetState(Tense); }
21  }
22 }
```

Listing 5.8: Simple example of a behavior description.

every frame. The current state of the state machine can be changed with the `SetState` method. A simple example is shown in Listing 5.8.

One state must be marked as initial state, by setting the `InitialState` variable of the `State` attribute to `true`. Optionally, the `Conformation` variable of the `State` attribute can be set to a conformation ID that is defined in the entity class file. If set, the conformation changes to the specified conformation when the state is entered. If the `Conformation` variable is not set, the entity structure stays in the current conformation, without being changed. Alternatively, the conformation can be changed by code using the `SetConformation` method.

As the behavior is described using a C# class, it is simple to add custom fields to save custom information. This information can be accessed inside the state methods, and can be considered in more sophisticated decision making processes.

5.5.2 Observing the Environment

The surrounding of the entity can be observed by using the entity's sensors. In order to be used, the sensors must be defined in the entity class file, as shown in Section 5.3. To make a sensor accessible in code, the sensor object must be injected into a property using the `Sensor` attribute, as shown in Listing 5.5. The sensor-ID in the attribute must reference the corresponding sensor-ID in the entity class file. The injected sensor object

provides various methods to find nearby entities. The available methods are shown in Listing 5.9.

```
IEnumerable<Entity> FindEntities();

IEnumerable<Entity> FindEntitiesOfClass
  (string entityClassId, Func<Entity, bool> filter);

IEnumerable<TBehavior> FindEntitiesWithBehavior<TBehavior>
  (Func<TBehavior, bool> filter);

bool FindNearestWithBehavior<TBehavior>
  (Func<TBehavior, bool> filter, out TBehavior nearest);

bool FindNearestEntityOfClass
  (string entityClassId, Func<Entity, bool> filter, out Entity nearest);

bool FindNearestEntityOfClassWithFreeSite
  (string entityClassId, string freeBindingSiteId, out Entity nearest);
```

Listing 5.9: Methods that are implemented by a sensor.

`FindEntities` simply returns all entities that are currently located inside the sensor cone. `FindEntitiesOfClass` is very similar, but only returns the entities with a specific entity class ID. Additionally, a filter function can be provided that can be expressed as lambda expression. `FindEntitiesWithBehavior` is a generic function that filters the entities in the cone by their behavior type. Instead of returning the entities, it returns the behavior objects of the entities. Again, a filter method can be provided. `FindNearestWithBehavior` is a generic function that finds the entity that is the nearest to the sensor and whose behavior has a specific type. The method returns true, if an entity is found with the specified behavior, otherwise it returns false. The behavior of the nearest entity is returned via the output parameter. A very similar method is `FindNearestEntityOfClass`. But instead of using the behavior type to find the nearest entity, it uses the entity class ID. The method `FindNearestEntityOfClassWithFreeSite` is very specialized, however, it is a quite commonly needed in molecular environments. It looks for the nearest entity inside the sensor cone, with a specific entity class ID that has no other entity bound at a specific binding site.

5.5.3 Environment Interaction

The behavior can interact with the environment by moving around compounds or by initiating and releasing bonds to other entities. Entities to interact with can be found using sensors, as described in Section 5.5.2. To move around compounds, the behavior can define a trajectory consisting of various sections, as discussed in Section 4.7. In the behavior description, a trajectory can be created by using the `TrajectoryBuilder`

class. `TrajectoryBuilder` is a helper class that provides a fluent interface to define a trajectory. An example usage of the `TrajectoryBuilder` is shown in Listing 5.10.

```
// create trajectory for proton
var trajectory = new TrajectoryBuilder()
  // move proton to the local ProtonSite, in 1 sec,
  // with colliders disabled
  .Movement(ProtonSite, TimeSpan.FromSeconds(1.0), false)
  // after that, bind the binding site of the proton
  // to the local ProtonSite
  .Binding(ProtonSite, proton.BindingSiteById("BindingSite"))
  // Create the trajectory
  .Create();

// initiate binding, so the binding sites are not in the
// status "free" anymore, but in "binding". When the binding
// is complete, it changes automatically to "bound".
ProtonSite.InitiateBinding(proton.BindingSiteById("BindingSite"));

// Apply the trajectory to the compound of the proton
proton.Compound.Trajectory(trajectory);
```

Listing 5.10: Moving a proton entity to a local binding site and then bind.

To create more complex trajectories, multiple `Movement` or `Attraction` sections can be combined. The destination can be specified by a spatial object like an entity, binding site, sensor, etc. or it can be specified explicitly using a vector and a quaternion. Instead of binding to a binding site at the end of a trajectory, the spatial location of the compound can also be set to floating or fixed.

The binding sites can be manipulated even without trajectories. There are three methods provided by the binding site that allow to manipulate the binding state. They are shown in Listing 5.11.

```
void InitiateBinding(BindingSite otherSite);
void InstantBind(BindingSite otherSite);
void ReleaseBond(Trajectory ejectTrajectory)
```

Listing 5.11: Binding site methods.

The method `InitiateBinding` solely marks both binding sites as "binding". It is an intermediate state between "free" and "bound". `InitiateBinding` should be used when two binding sites are currently moving towards each other, but they are not bound yet. This intermediate state avoids that other entities, that may desire to interact with this binding site as well, mistakenly assume that the binding site is still free and initiate a binding too. `InstantBind` is straight forward, it binds the two binding sites instantly together. `ReleaseBond` releases the bond between two binding sites again. Optionally, an eject trajectory can be provided, to move the released entity away.

5.6 Concentration Controllers

Using concentration controllers, the concentration of a specific entity in the environment can be changed interactively. However, depending on the environment and the entity class, creating and deleting entities on demand may be no trivial task. For example, it may be undesirable to delete entities that are currently bound to other entities. Therefore, the framework uses customizable entity factories that are responsible for creating, finding and deleting certain entities in the environment.

Similar to behaviors, entity factories are defined in specialized classes. Those classes must be compiled to a .NET library, which is added to the Unity assets of the player, just like behavior classes. Behavior classes and entity factories can also be compiled into the same .NET library. Just like behaviors, entity factories have an ID. The ID is set by using the `EntityFactoryId` attribute. The concentration controllers defined in the environment file, are wired together with the entity factory by using this ID.

Entity factories defined in code, must extend the `EntityFactory` class. The class declares three abstract methods that must be implemented: `Create`, `Filter` and `Delete`. The implementation of the `Create` method, must create one new desired entity. This method is called every time, when the concentration should increase. The number of calls depends on how many entities should be created. The `Filter` method, gets an entity as parameter. The implementation has to determine if this is an entity that is handled by this entity factory. It can also implement some conditions under which the entity is handled. The method is used to count the number of entities and to find suitable entities to delete when there are too many of them in the environment. The `Delete` method is used to safely remove an entity.

An example implementation of an entity factory is shown in Listing 5.12. In this example, the entity factory creates entities of the entity class "o2", and locates them at a random position in the compartment, from where it starts to float freely. The entity factory only handles entities of the class "o2" and only if the first (and only) binding site is not bound. An entity is deleted by simply disposing it.

5.7 Rendering

The player uses cellVIEW for rendering the environment [MAPV15]. cellVIEW is specifically designed for rendering large molecular environments [MAPV15]. In cellVIEW a set of molecular structures are defined and then individual instances of those structures are placed in the environment. This can save a lot of memory when the same structures are used over and over again [MAPV15]. Unfortunately, this framework cannot use this feature extensively. Entity structures of the same entity class can change their conformation independently. That means that entities can have distinct structures at the same time, even if they are of the same class. This is the reason why they cannot share the same structure in cellVIEW. However, entity types whose structures have only one possible conformation and therefore cannot change, such a structure sharing is utilized.

```
1 [EntityFactoryId("hemoglobin.ef.freeO2")]
2 public class O2EntityFactory : EntityFactory
3 {
4   public override void Create()
5   {
6     var entity = environment.AddEntity("o2", false);
7     entity.Compound.Float(
8       new LocRotStatic(
9         GeometryUtils.RandomVectorInsideCompartment(environment.
              Compartments.First()),
10        GeometryUtils.RandomQuaternion()
11      )
12    );
13  }
14
15  public override bool Filter(Entity entity)
16  {
17    return (entity.Class.Id == "o2" && entity.BindingSites[0].IsFree);
18  }
19
20  public override void Delete(Entity entity)
21  {
22    entity.Dispose();
23  }
24 }
```

Listing 5.12: Example implementation of an entity factory

cellVIEW uses several compute buffers to transfer the scene information from the central processing unit (CPU) to the GPU. The buffer used for the positioning of molecules is updated every frame. The buffer containing structure information is only updated on demand, e.g. during conformation changes.

CHAPTER 6

Results

The model proposed in Chapter 4 was implemented as framework using Unity [Tec17], cellVIEW [MAPV15] and the Microsoft .NET Framework [Mic17]. In order to being able to animate more and more complex molecular machines, new features were added to the framework during its emergence. In that way, the framework was iteratively improved. Thus, the created animations of the three molecular machines were used to test the framework and simultaneously function as a proof of concept.

At first, the relatively simple molecular machine hemoglobin was animated. Next the more sophisticated ATP-synthase. Finally, an animation of kinesin was created using the framework. The following sections describe how the framework was used to create the animations of the three molecular machines.

6.1 Hemoglobin

Hemoglobin is a quite simple molecular machine that carries oxygen to from the lungs, where oxygen is plentiful, and delivers it to all areas of the body where the oxygen concentration is low [AJL+14]. The biological background of hemoglobin is explained in Section 2.5.1.

The animation includes multiple oxygen, carbon-monoxide and hemoglobin molecules. Via concentration controllers, the viewer can change the concentration of the three molecules separately. If the concentration of oxygen or carbon-monoxide exceeds a certain threshold, it is bound to a binding site of the hemoglobin. When the concentration drops below a certain threshold, the bound molecule is released again. Carbon-monoxide binds to hemoglobin much easier than oxygen. Therefore, a small carbon-monoxide concentration is sufficient to occupy the binding sites of a hemoglobin molecule. By varying the concentrations of oxygen and carbon-monoxide, the viewer can see how the molecules are absorbed by hemoglobin and released again. A snapshot of the resulting animation is

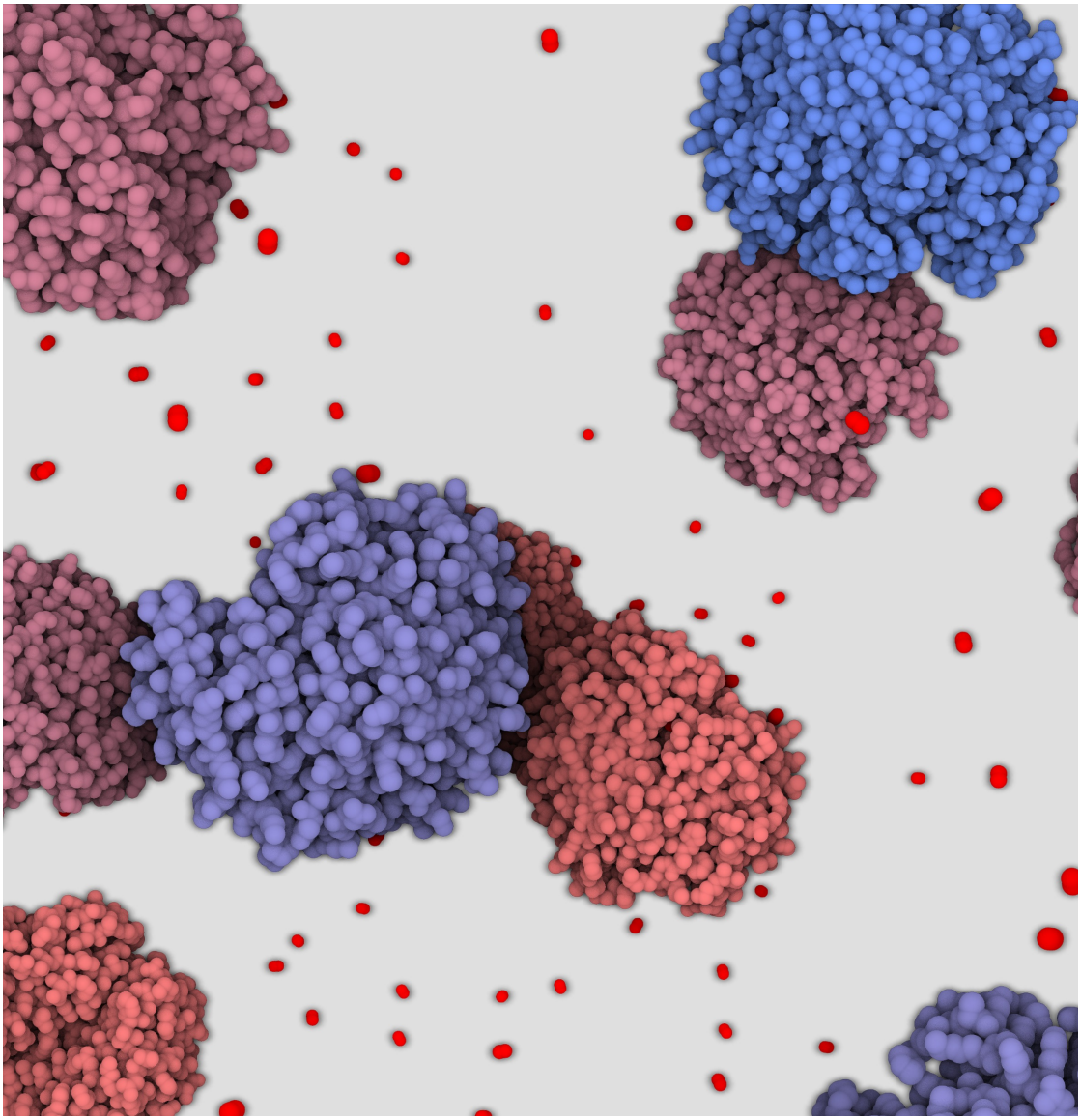


Figure 6.1: Snapshot of the hemoglobin animation showing oxygen and hemoglobin molecules. The redder the hemoglobin, the more oxygen is bound. The blue hemoglobin molecules are deoxygenated.

shown in Figure 6.1. In reality, a hemoglobin molecule is approximately seven nano-meters in size [FPSF84].

The environment contains three distinct entity classes: hemoglobin, oxygen and carbon-monoxide. Oxygen and carbon-monoxide are rather simple. Both have no explicit behavior, because they are attracted and repelled by the hemoglobin behavior. They both have only one conformation and consist of two atoms. The oxygen structure consists

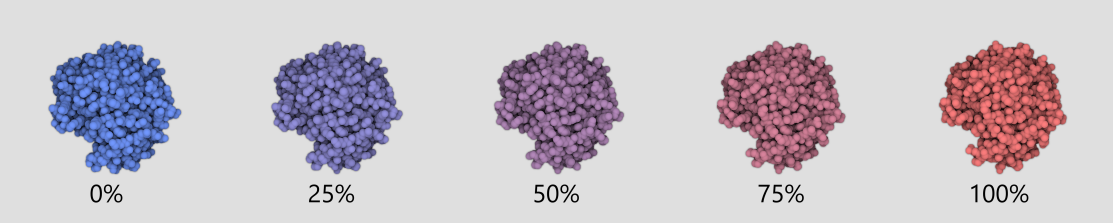


Figure 6.2: Different conformations of hemoglobin. Not only the color, but also the atom structure changes slightly.

of two oxygen atoms, and the carbon-monoxide structure consists of one oxygen atom and one carbon atom.

The hemoglobin is the central entity in this environment. It has a total of five conformations: deoxygenated, 25 % oxygenated, 50 % oxygenated, 75 % oxygenated and 100 % oxygenated. Depending on how many molecules are bound to the hemoglobin, the corresponding conformation is assumed. The entity structure is imported from the PDB. For the deoxygenated conformation, the PDB structure 2HHB [FPSF84] is used. The 100 % oxygenated conformation uses the PDB structure 1HHO [Sha83]. The intermediate conformations are interpolated linearly. To highlight the oxygen saturation of the hemoglobin, its color changes with each conformation. The blue hemoglobin have no molecules bound. The redder the color of the hemoglobin, the more molecules are bound. The different conformations are shown in Figure 6.2.

The hemoglobin has four binding sites. The positions of the binding sites are extracted from the oxygenated PDB structure 1HHO [Sha83]. This structure contains oxygen molecules that are bound to the hemoglobin. The position of these molecules is used as binding site position in the framework model. For each binding site, additionally a sensor is attached at the same position. The sensors monitor the molecule concentrations near the binding sites. A hemoglobin entity with visible sensor cones is shown in Figure 4.6b.

The hemoglobin changes its conformation depending on the number of bound molecules at the binding sites. The binding and unbinding is works equally for each binding site. Thus, they are all controlled by the same behavior code. An excerpt from the method that handles the binding sites is shown in Listing 6.1. First, the corresponding sensor is used to find all nearby free oxygen (O_2) and carbon-monoxide (CO) entities. Then the number of found entities is averaged over time. This is necessary for a smoother animation. Because the entities are floating randomly, they often enter and leave the sensor cone, which results in quickly changing concentration values. If not averaged, the hemoglobin binds and unbinds molecules too fast to see. When the binding site is free, it is checked if the carbon-monoxide or oxygen concentration exceeds the bind threshold. The bind threshold is different for oxygen and carbon-monoxide, because carbon-monoxide is much more affine to bind to hemoglobin [GDHS03]. Furthermore, the bind thresholds are dependent on the number of molecules already bound ($boundCount$), because the more molecules are already bound, the easier it is for the remaining binding

sites to bind [AJL+14]. If the binding site is already bound, it is checked if the averaged concentration dropped below an unbind threshold. Again, this threshold is dependent on the entity class and the total molecules bound to the hemoglobin. If the concentration dropped below the unbind threshold, the bound entity is released.

6.2 ATP-Synthase

ATP-synthase is a molecular machine that uses a proton gradient across a membrane to rotate a rotor that drives a generator that binds ADP and phosphate together. The biological background of ATP-synthase is explained in Section 2.5.2.

The animation shows two compartments, one filled with protons and one filled with ADP and phosphate. The compartment with the protons represents the outside of the membrane enclosed area and the other the inside. The two compartments are connected via an ATP-synthase. A snapshot of the animation in shown in Figure 6.3. The membrane between the two compartments is not included in the animation for simplicity reasons and to provide a better view on the ATP-synthase. The ATP-synthase attracts protons from the outside compartment, and routes it through the stator to the F_0c subunit of the F_0 complex that is next to the stator. Then the F_0 complex is rotated by one subunit. After a whole revolution, the proton is unbound from the F_0c subunit and released to a random position in the inside compartment. As the axle rotates, the conformation changes of the F_1 complex are animated and ADP and phosphate molecules are attracted, bound together and released as ATP again.

Each rotation step takes about two seconds in the animation, in order that the process is easily comprehensible. The shown ATP-synthase has ten F_0c subunits, thus 20 seconds are needed for one full revolution, assuming enough protons, ADP and phosphate molecules are present. In biology, ATP-synthase can rotate with a speed of about 130 revolutions per second [NSAS08]. In every revolution, three ATP molecules are generated. Thus, the about 20 nano-meter molecular machine produces approximately 390 ATP molecules per second [SSW+16, NSAS08].

The environment is composed of two compartments that are located next to each other. The structure of the ATP-synthase is imported from the PDB structures 5T4O, 5T4P, FT4Q [SSW+16]. These three structures include the whole ATP-synthase in three different conformations. The different proteins of the machine are annotated inside the PDB file and can be easily separated. This ATP-synthase contains ten F_0c subunits in the F_0 complex.

A proton is basically a hydrogen atom without an electron. Therefore, it is represented as an entity with a structure containing only one hydrogen atom. Protons are shown in yellow on the left side in Figure 6.3. ADP and phosphate are two separate entity classes. They are extracted from the ATP-synthase PDB structure [SSW+16]. ADP has a binding site which can bind to phosphate. When bound together, they form ATP. ATP has no dedicated entity, and is only represented by the compound of ADP and phosphate.

```
1 public void HandleBidningSite(BindingSite bindingSite, Sensor sensor,
      AverageOverTime o2Avg, AverageOverTime coAvg, int boundCount)
2 {
3   // Find free O2 and CO with sensor
4   var o2 = sensor.FindEntitiesOfClass("o2", p => p.BindingSites[0].IsFree)
        .ToArray();
5   var co = sensor.FindEntitiesOfClass("co", p => p.BindingSites[0].IsFree)
        .ToArray();
6
7   // Calculate average concentration
8   o2Avg.Push(o2.Length);
9   coAvg.Push(co.Length);
10
11  if (bindingSite.IsFree)
12  {
13    if (coAvg.GetAvg() >= bindThresholdCoByBoundCount[boundCount])
14    {
15      InitiateBinding(co, bindingSite);
16    }
17    else if (o2Avg.GetAvg() >= bindThresholdO2ByBoundCount[boundCount])
18    {
19      InitiateBinding(o2, bindingSite);
20    }
21  }
22  else
23  {
24    var boundClassId = bindingSite.OtherEntity.Class.Id;
25    if (boundClassId == "co")
26    {
27      // avg+1 because one is bound that is not in the scope of the sensor
28      if (coAvg.GetAvg() + 1 <= unbindThresholdCoByBoundCount[boundCount])
29      {
30        bindingSite.ReleaseBond();
31      }
32    }
33    else if (boundClassId == "o2")
34    {
35      if (o2Avg.GetAvg() + 1 <= bindThresholdO2ByBoundCount[boundCount])
36      {
37        bindingSite.ReleaseBond();
38      }
39    }
40  }
41 }
```

Listing 6.1: Excerpt from the hemoglobin behavior description for one binding site.

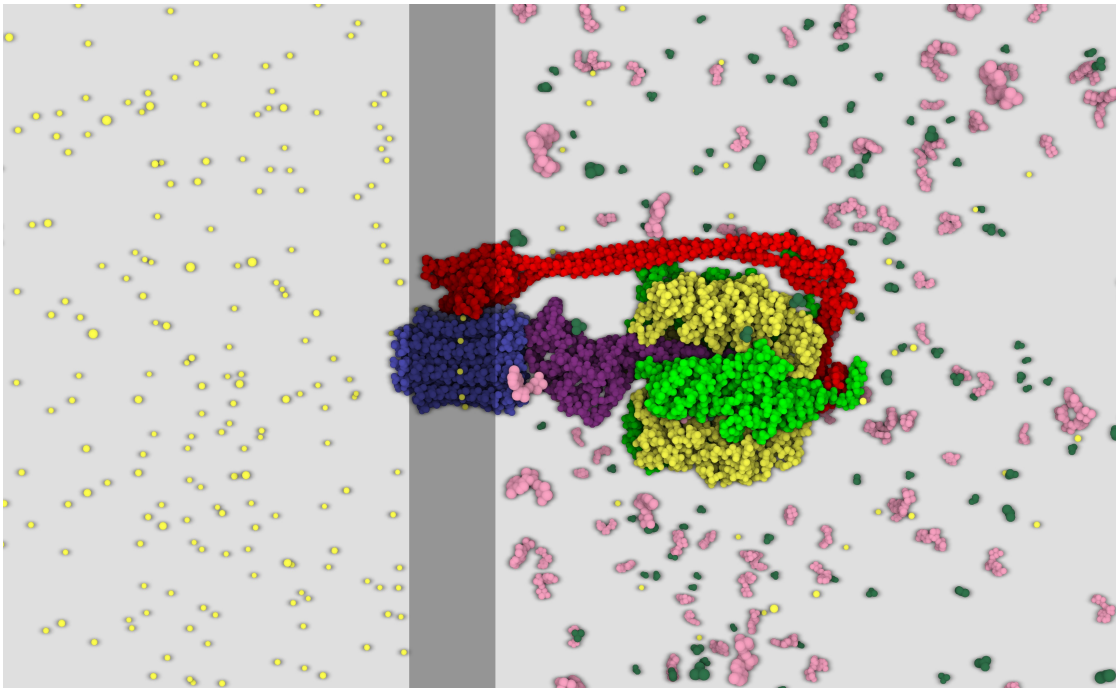


Figure 6.3: Snapshot of the ATP-synthase animation. The yellow entities on the left are protons, the small pink molecules on the right are ADP, the dark green molecules are phosphates, and the ATP molecules are green and pink. The compartments are divided by a membrane.

ADP has another binding site, which is able to bind to the ADP binding site in the $F1$ complex. Phosphate has one binding site that either binds to ADP or to the phosphate binding site in the $F1$ complex.

In biology, ATP-synthase is controlled by complex interactions of different proteins. To simplify the interaction in the animation model, a control hierarchy is introduced. This is realized by an ATP-synthase entity class that represents an abstract control entity without any atoms. Instead, it provides a special binding site for each machine part. The parts of the ATP-synthase are represented as separate entities, and are bound to the appropriate binding sites of the ATP-synthase entity. The hierarchy and the binding sites on which the different entities are bound together is shown in Figure 6.4. The ATP-synthase entity also has a sensor that is directed to the outside compartment to find protons.

The stator entity contains all atoms that are related to a stator component. As the rotor rotates, the stator slightly shifts its shape. Because there are ten F_0c subunits, one revolution of the rotor requires ten steps. Therefore, ten conformations are created for the stator, one for each step. The atom positions for three steps can be extracted from the three PDB files [SSW+16]. The conformations for the intermediate steps are

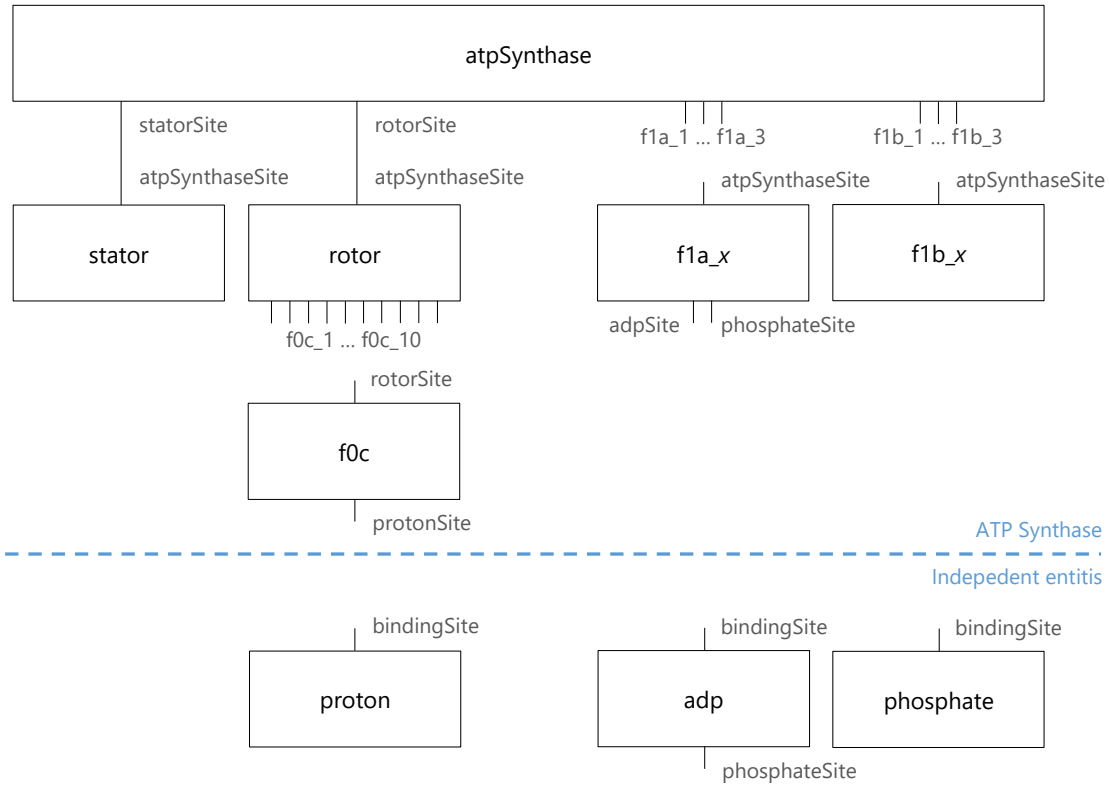


Figure 6.4: Hierarchy of the ATP-synthase entities. The rectangles represent entity classes with their entity ID. The lines are their binding sites.

interpolated linearly from the three known ones.

The atoms for the rotor entity are imported from the axle part in the PDB file 5T4O [SSW+16]. The F_0c proteins are represented as separate entities. They are attached to the rotor using ten binding sites that are located around the rotor entity. Each F_0c entity can assume two different conformations: tensed and relaxed. When a proton is bound, the F_0c subunit is relaxed, when the proton binding site is free, it is tensed. In the PDB file 5T4O, the F_0c subunit at the stator is tensed, all others are relaxed [SSW+16]. By extracting the atom positions of two subunits in different conformations, they can be combined to create an F_0c entity with two conformations.

The F_1 complex consists of three $F_1\alpha$ and three $F_1\beta$ proteins. It is easier to import each of the six F_1 proteins separately than it is to import one $F_1\alpha$ and one $F_1\beta$ protein and rebuild the F_1 complex using the subunits. Therefore, each of the six F_1 proteins is imported as separate entity class and bound directly to the ATP-synthase entity. As a consequence, there are three distinct $F_1\alpha$ entity classes and three distinct $F_1\beta$ entity classes. In biology, the binding sites for ADP and phosphate are located between the $F_1\alpha$ and $F_1\beta$ proteins. Since the binding sites have to be assigned to an entity, the binding

sites are arbitrarily assigned to the $F_1\alpha$ entities. Thus, the $F_1\alpha$ entities also have a sensor attached that is used to find nearby ADP and phosphate entities. $F_1\alpha$ and $F_1\beta$ have three different conformations, that are dependent on the rotor position. In the first, they attract ADP and phosphate. In the second, they bind ADP and phosphate together to form ATP. In the third, the newly formed ATP is ejected from the binding site.

Behavior Description

As the behaviors are described using C# classes, they can be used to aggregate specific animation actions in methods. For example, the behavior class for the $F_1\alpha$ provides methods for attracting and binding ADP and phosphate, bind them together to form ATP and release ATP. The implementation of these methods is shown in Listing 6.2. Similarly, the behavior description of the rotor exposes methods to rotate the rotor by one step, and to access the F_0c entity that is currently next to the stator.

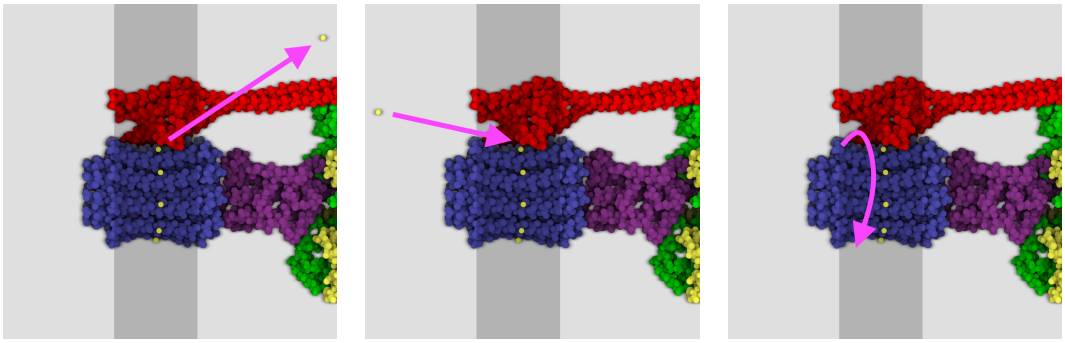
The behaviors of the $F_1\alpha$ entities and the rotor entity are injected into the ATP-synthase behavior by using the `BoundEntity` attribute. As a result, the ATP-synthase behavior has access to the exposed methods and can control the subunits using them.

The ATP-synthase behavior is implemented as state machine, that steps through all its four steps sequentially. The implementation of these steps is shown in Listing 6.3. The first step releases the proton bound at the F_0c next to the stator into the inside compartment (see Figure 6.5a). Next, it uses the sensor that is directed to the outside compartment to find a proton. If a proton is found, it is attracted to the sensor, which is at the stator, and then moved to the proton binding site of the F_0c entity next to the stator (see Figure 6.5b). During the movement of the proton, the ATP-synthase behavior is in the `BindingInProgress` state. As soon as the proton is bound, the state is changed to `WaitUntilF1IsReady`. The ATP-synthase behavior stays in this state as long as the method `F1Ready` returns `false`. The method `F1Ready` makes use of the methods exposed by the $F_1\alpha$ behavior to make sure ADP and phosphate is bound. It also makes and releases ATP depending on the current rotation progress. As soon as all $F_1\alpha$ entities are ready for the next rotation step, the method returns `true`, and the ATP-synthase rotates the rotor by one step (see Figure 6.5c). During the rotation is in progress, the state machine stays in the state `Rotating`. When the rotation step is done, the state machine starts again in the `Start` state.

The method `F1Ready` calls the exposed methods of the $F_1\alpha$ behavior depending on the current rotation progress. When the rotation progress is 0%, the first $F_1\alpha$ entity finds an ADP and a phosphate entity and attracts them to its binding sites using the method `BindAdpAndPhosphate` shown in Listing 6.2. The second $F_1\alpha$ entity binds the already bound ADP and phosphate entities together to form ATP using the `MakeAtp` method. The third $F_1\alpha$ entity already contains an ATP that is now released using the method `ReleaseAtp` (see Figure 6.5d). When the rotation progress exceeds 33.3%, the first $F_1\alpha$ entity binds the previously bound ADP and phosphate entities together to create a new ATP. The second $F_1\alpha$ entity releases the ATP that was created in the last step. The

```
1 [BindingSite("adpSite")]       public BindingSite AdpSite {get;set;}
2 [BindingSite("phosphateSite")] public BindingSite PhosphateSite {get;set;}
3 [BoundEntity("adpSite")]       public AdpBehavior BoundAdp {get;set;}
4 [BoundEntity("phosphateSite")] public PhosphateBehavior BoundPhosphate{...
5
6 public void BindAdpAndPhosphate()
7 {
8   if (BoundAdp == null)
9   {
10    AdpBehavior adp;
11    if (Sensor.FindNearestWithBehavior<AdpBehavior>(a => a.BindingSite.
          IsFree && a.PhosphateSite.IsFree, out adp))
12    {
13      var trajectory = new TrajectoryBuilder()
14        .Movement(this.AdpSite, TimeSpan.FromSeconds(1.0), false)
15        .Binding(this.AdpSite, adp.BindingSite)
16        .Create();
17      AdpSite.InitiateBinding(adp.BindingSite);
18      adp.Owner.Compound.Trajectory(trajectory);
19    }
20  }
21  if (BoundPhosphate == null)
22  {
23    PhosphateBehavior phosphate;
24    if (Sensor.FindNearestWithBehavior<PhosphateBehavior>(p => p.
          BindingSite.IsFree, out phosphate))
25    {
26      var trajectory = new TrajectoryBuilder()
27        .Movement(this.PhosphateSite, TimeSpan.FromSeconds(1.0), false)
28        .Binding(this.PhosphateSite, phosphate.BindingSite)
29        .Create();
30      PhosphateSite.InitiateBinding(phosphate.BindingSite);
31      phosphate.Owner.Compound.Trajectory(trajectory);
32    }
33 } }
34
35 public void MakeAtp()
36 { BoundAdp.PhosphateSite.InstantBind(BoundPhosphate.BindingSite); }
37
38 public void ReleaseAtp()
39 {
40   var trajectory = new TrajectoryBuilder()
41     .Movement(GeometryUtils.RandomVectorInsideCompartment("inside"),
          TimeSpan.FromSeconds(1.0), false)
42     .Float() // float after trajectory
43     .Create();
44   AdpSite.ReleaseBond(trajectory);
45 }
```

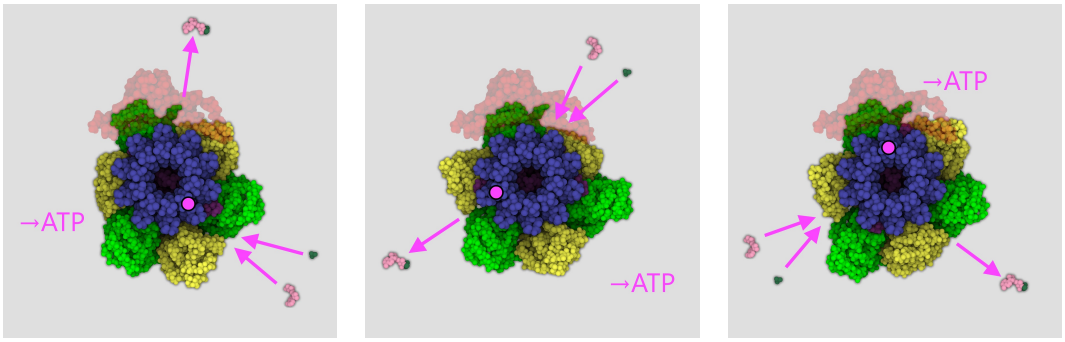
Listing 6.2: Methods exposed by the $F_1\alpha$ behavior description.



(a) Release the proton bound to the F_0c next to the stator into the inside compartment.

(b) Find proton in the outside compartment and move it to the F_0c entity next to the stator.

(c) Rotate F_0 complex.



(d) The right binding sites attract and bind ADP and phosphate, the left binding sites create ATP, and the top binding sites release the ATP.

(e) The right binding sites create ATP from the previously bound ADP and phosphate, the binding sites at the left release the created ATP, and the binding sites at the top attract new ADP and phosphate.

(f) The right binding sites release the created ATP, the right binding sites attract new ADP and phosphate, and the binding sites at the top create ATP.

Figure 6.5: Different steps in the ATP-synthase animation. Figures a-c show steps of the F_0 complex seen from the side, Figures d-f show steps of the F_1 complex, seen from the F_0 complex towards the F_1 complex. The stator is semi-transparent.

third $F_1\alpha$ entity attracts and binds new ADP and phosphate entities to its binding sites (see Figure 6.5e). When the rotation progress exceeds 66.7%, the first $F_1\alpha$ releases the newly formed ATP. The second $F_1\alpha$ entity looks for new ADP and phosphate entities, and the third $F_1\alpha$ entity forms a new ATP (see Figure 6.5f). Then the cycle repeats.

The stator behavior simply reads the rotation progress from the rotor entity and changes the stator entity conformation accordingly. The $F_1\beta$ behavior works identically.

```
[State(InitialState = true)]
public void Start()
{
  if (Rotor.F0cAtStator.ProtonSite.IsBound)
  {
    var ejectTrajectory = new TrajectoryBuilder()
      .Movement(GeometryUtils.RandomVectorInsideCompartment("inside"),
          TimeSpan.FromSeconds(1.0), false)
      .Float() // start floating at the end
      .Create();
    Rotor.F0cAtStator.ProtonSite.ReleaseBond(ejectTrajectory);
  }
  Entity proton;
  if (ProtonSensor.FindNearestEntityOfClass("proton", p => p.BindingSites
      [0].IsFree, out proton))
  {
    var trajectory = new TrajectoryBuilder()
      .Movement(ProtonSensor, TimeSpan.FromSeconds(0.7), true)
      .Movement(Rotor.F0cAtStator.ProtonSite, TimeSpan.FromSeconds(0.3),
          false)
      .Binding(Rotor.F0cAtStator.ProtonSite, proton.BindingSites[0])
      .Create();
    Rotor.F0cAtStator.ProtonSite.InitiateBinding(proton.BindingSites[0]);
    proton.Compound.Trajectory(trajectory);
    SetState(BindingInProgress);
  }
}
[State]
public void BindingInProgress()
{
  if (Rotor.F0cAtStator.ProtonSite.IsBound)
  { SetState(WaitUntilF1IsReady()); }
}
[State]
public void WaitUntilF1IsReady()
{
  // this method returns true when all F1 entities are ready for rotation
  if (F1Ready())
  {
    Rotor.RotateOneStep();
    SetState(Rotating);
  }
}
[State]
public void Rotating()
{
  if (!Rotor.IsRotating)
  { SetState(Start); } // repeat
}
```

Listing 6.3: Excerpt from the ATP-synthase behavior description, showing the different states.

6.3 Kinesin

Kinesin is a molecular machine that walks along a microtubule. The biological background of kinesin is explained in Section 2.5.3.

The kinesin animation shows a microtubule and a kinesin floating near the microtubule. In biology, the environment would be full of smaller molecules, including ATP. Those smaller molecules are omitted from the animation, due to simplicity reasons and to allow a better view on the kinesin. The long neck and the cargo are also not shown in the animation. A snapshot of the animation is shown in Figure 6.6. In the beginning, the kinesin and its heads are floating randomly to mimic Brownian motion (see Figure 6.7a). As soon as one head of the kinesin gets near enough to the microtubule, it binds weakly. After a second, the head binds strongly, which is pictured as moving nearer to the microtubule (see Figure 6.7b). Then, the bound head turns yellow, which symbolizes that an ATP molecule is bound. Simultaneously, the neck-linker is pulled towards the head, which causes the other head to be thrown forward (see Figure 6.7c and Figure 6.7d). After a short time, the bound head turns back to its previous color, which symbolizes that the ATP was cleaved. The unbound head now is near enough to the next tubulin-β to detect and bind to it. Then all steps repeat with the other head.

In the animation, one step of the kinesin takes four seconds. In reality, the kinesin can walk with up to 250 steps per second. With each step, the kinesin moves approximately by eight nano-meters, thus it can walk up to 2000 nano-meters in one second [How01].

The tubulin structure is imported from the PDB structure 5SYC [KHH+17]. For both subunits tubulin-α and tubulin-β a separate entity class is created. The microtubule is realized by a large, atom-less entity that provides binding sites for tubulin-α and tubulin-β entities at the right positions. At startup, the required number of tubulin-α and tubulin-β entities are created and instantly bound to the binding sites of the microtubule entity. The tubulin-β additionally provides a binding site where a kinesin head can attach.

The kinesin structure is imported from the PDB structure 3KIN [KSM+97]. The PDB file contains the whole kinesin as one unit. For the animation, the kinesin was divided into four entities: two heads and two neck-linkers, which are connected via binding sites. Each head is bound to a neck-linker and the two neck-linkers are bound together. The heads have a sensor attached that is used to find nearby tubulin-β entities. Furthermore, the heads have five conformations that are related to the animation steps: `free`, `bound weak`, `bound tight`, `neck-linker tight` and `neck-linker tight bound weak`. The conformations `bound weak` and `bound tight` are almost the same, only the binding site position for tubulin-β differs, so the head is nearer at the microtubule in `bound tight` conformation. In the conformation `neck-linker tight`, the binding site where the neck-linker is bound is rotated in such a way, that the neck-linker is pulled towards the head, which causes the neck-linker to point forward. Furthermore, the color is set to yellow, which symbolizes that an ATP molecule is bound.

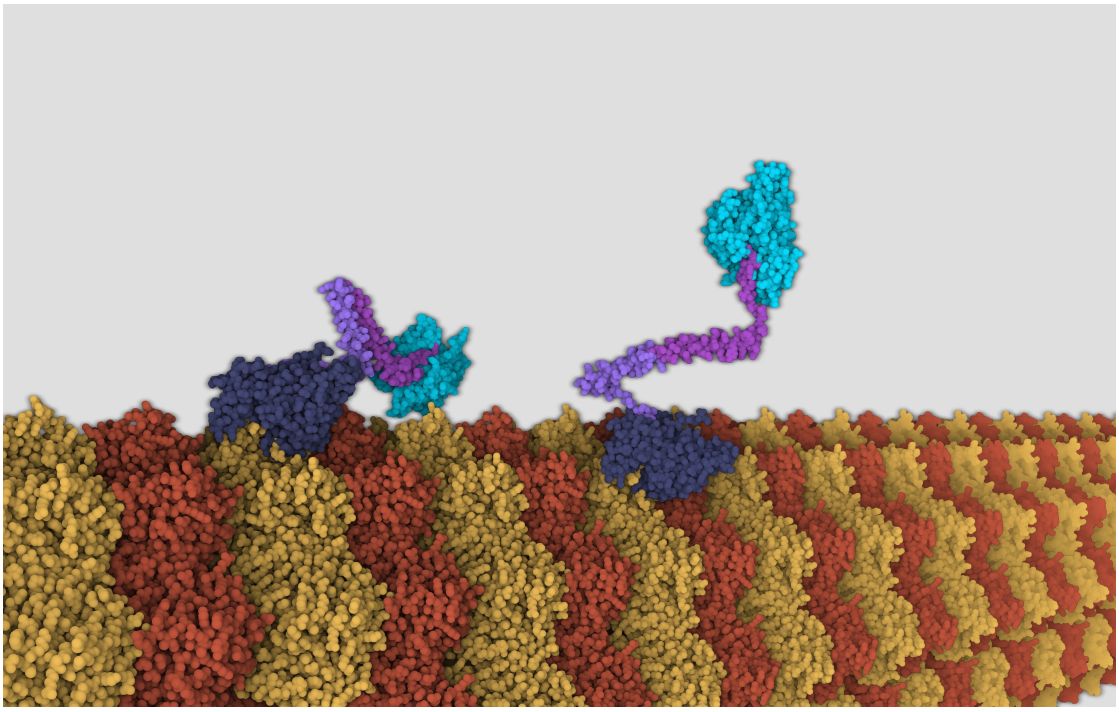
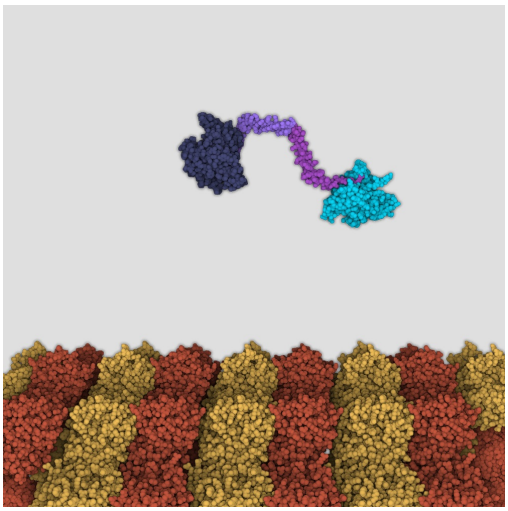


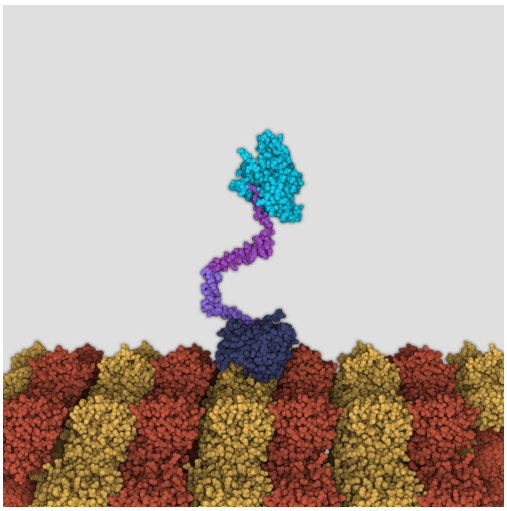
Figure 6.6: Snapshot of the kinesin animation. Two kinesins are walking along the microtubule.

The difference of the conformations `free`, `bound tight` and `neck-linker tight` are shown in Figure 6.7.

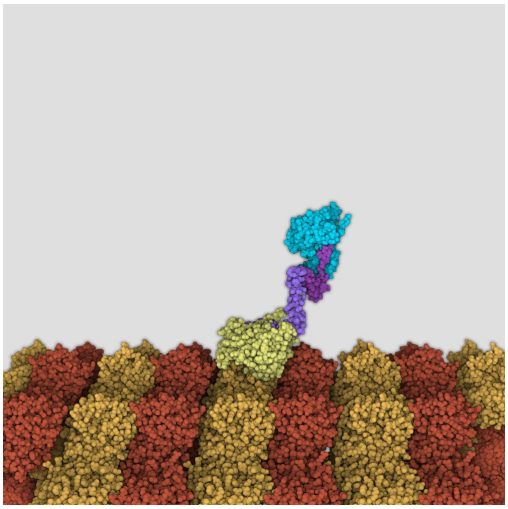
The kinesin behavior is controlled by the two heads. As soon as a head finds a tubulin-β entity, a series of states are run through, as shown in Listing 6.4. Each state induces a conformation change. After one second, the state machine moves forward to the next state. In the `HydrolizeAtp` state, the bond to the tubulin-β is released. After that, the head stays in pause for 2 seconds, so the other head has time to find the next tubulin-β entity on the microtubule and move this head away. Otherwise, if the head would directly go back to the `FindMicrotubule` state, the head would instantly bind to the same tubulin-β entity again.



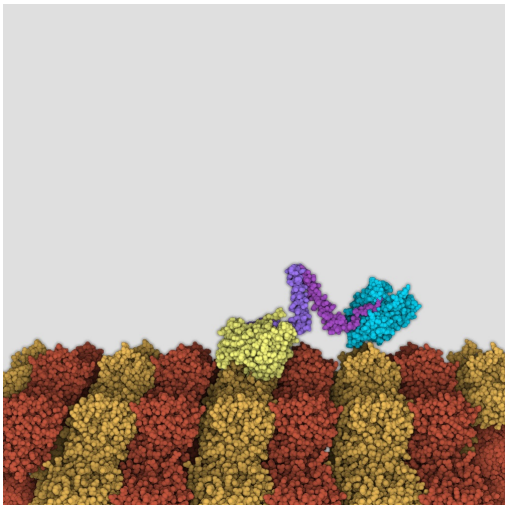
(a) Kinesin is floating, head conformation is `free`.



(b) Head on the microtubule is in `bound tight` conformation.



(c) Conformation is changing from `bound tight` to `neck-linker tight`.



(d) Head on the microtubule is in `neck-linker tight` conformation.

Figure 6.7: Kinesin with different head conformations of the bound head.

```
1  [State(InitialState = true, Conformation = "free")]
2  public void FindMicrotubule()
3  {
4    NeckLinkerBrownianMotion(); // moves neck linker binding site randomly
5    TubulinBetaBehavior tubulinBeta;
6    if (TubulinSensor.FindNearestWithBehavior<TubulinBetaBehavior>(tb => tb.
         DockSite.IsFree, out tubulinBeta))
7    {
8      d.Trajectory(trajectory);
9      tubulinBeta.DockSite.InstantBind(this.TubulinSite);
10     SetState(WeakBindToMicrotubule);
11   }
12 }
13 [State(Conformation = "bound weak")]
14 public void WeakBindToMicrotubule()
15 {
16   NeckLinkerBrownianMotion();
17   if (SecondsInState >= 1) // waits for 1sec
18   { SetState(TightBindToMicrotubule); }
19 }
20 [State(Conformation = "bound tight")]
21 public void TightBindToMicrotubule()
22 {
23   NeckLinkerBrownianMotion();
24   if (SecondsInState >= 1) // waits for 1sec
25   { SetState(ReleaseAdp); }
26 }
27 [State(Conformation = "neck-linker tight")]
28 public void TightNeckLinker()
29 {
30   if(SecondsInState >= 1) // waits for 1sec
31   { SetState(HydrolizeAtp); }
32 }
33 [State(Conformation = "neck-linker tight bound weak")]
34 public void HydrolizeAtp()
35 {
36   if (SecondsInState >= 1) // waits for 1sec
37   {
38     TubulinSite.ReleaseBond();
39     SetState(Pause);
40   }
41 }
42 [State]
43 public void Pause() // avoids that head instantly binds again to the same
       tubulinBeta
44 {
45   NeckLinkerBrownianMotion();
46   if (SecondsInState >= 2) // waits for 2sec
47   { SetState(FindMicrotubule); }
48 }
```

Listing 6.4: Excerpt from the kinesin head behavior description.

CHAPTER 7

Discussion and Conclusion

This chapter summarizes the outcome of this work. The result is critically reflected and limitations are discussed. At the end, some insights and final thoughts are provided.

7.1 Critical Reflection

The framework introduces a novel approach for animating molecular machines. In already existing tools, the animator authors the molecular environment via key-frame animation [IMS+17, WTHT14]. In the proposed framework, the animator provides a behavior description of molecular entities. These entities act as agents and behave according to the provided description in the molecular environment. Usually, the number of distinct entity classes is small compared to the total number of entities. Since the behavior description only has to be formulated once for each entity class, such an animation scales very easy, even for a large number of molecular particles in an environment. Moreover, the animated machine reacts automatically to changing environmental conditions, without having to be explicitly animated. This is because the animation is inferred from the behavior description, which defines the desired actions for different conditions.

Because the entities can react to environmental conditions, they also can be reused in other environments, where the conditions may be different. It is also possible to put multiple molecular machines in the same environment, without having to adjust the animation. They simply behave according to their description.

The framework is only implemented at prototypical level and requires programming skills to create environments. Therefore, it is not yet ready to be used productively by biologists. Hence, if and how much time is saved using this approach cannot be measured yet. However, the animations created with the framework demonstrate that agent-based animation is a viable option for visualizing molecular machines.

A simple animation that only includes a few molecules is may be easier accomplished by using a conventional key-frame approach, instead of the proposed framework. This is because, the framework requires some overhead work, like defining binding sites and sensors. But for a large number of objects, the key-frame approach is more tedious, because each object has to be animated separately. The overhead work required for this framework may be reduced by future tools that support the animator in creating appropriate entity classes and environments.

Because the framework acts primarily as proof of concept, it is not optimized for efficiency. For example, sensors iterate through all entities in the environment to find the ones that are in the sensor cone. This approach is not very efficient for environments with a large number of entities, especially when a lot of entities make use of sensors. This could be optimized using some type of spatial decomposition approach, e.g. grids, binary space partitioning, a hierarchical quadtree, etc. [Par12a].

7.2 Limitations

The framework does not include a GUI for designing environments. This makes it difficult to precisely model spatial components like binding sites and sensors. Assembling multiple PDB datasets together is also a challenging task without visual guidance. Since only an API is provided, the framework is only usable by animators that are already experienced in programming.

Animations of conformation changes in entities are produced by linearly interpolating the atom positions. This is not realistic, since the constraints imposed by the chemical bonds are not considered. Moreover, it can happen that two or more atoms are at the same spatial position during the animation. An interpolation mode that considers certain constraints would be more suitable, e.g. the normal mode analysis for proteins [SHR09].

7.3 Levels of Abstraction

One goal of this framework is to find problems in the animation process, then to generalize them and find a solution that suits a broader domain. The challenge is to find a suitable level of abstraction to maximize the benefits for the animator.

Independent from the animation approach, the animator has to provide some sort of description of what should happen in the animation. Finding an appropriate way of describing this is challenging. In current tools, the animator provides key-frames as description [IMS+17, WTHT14]. This approach gives the animator the most control over the animation. As a consequence the animation task often is tedious. On the other extreme, the animator could model the physical properties of a molecule. The animation would then be computed by a simulator. In that case, the behavior and ultimately the animation would results from physical properties. Such a description is very abstract and highly reusable. However, it gives only a very indirect control over the resulting

animation. Since the goal of this framework is to communicate ideas via animations, this is not a suitable approach. The animator should not have to fine-tune physical properties to achieve the expected animation. It is more straightforward to describe the behavior more directly.

This framework aims for an abstraction level that has the most advantages for the animator. It should be abstract enough to avoid repetitive tasks, yet not take away the ability to precisely influence the resulting animation. It is intended to find a certain level of abstraction for the framework that allows to describe most molecular machines easy and well, but does not limit the possibilities either. As a rule of thumb, the less abstract the description is, the more possibilities the animator has, but also the more effort is needed to take care of every detail. In too abstract approaches the animator cannot intuitively imagine the effect of the changes in the description to the animation. The chosen level of abstraction of the proposed model provides a good starting point, but may be adjusted in future work to better fulfill the requirements.

7.3.1 Remarks on the Mechanical Approach

In Section 4.9.1 the initial mechanical approach is discussed. As it turns out, the initial idea to map molecular machines to mechanical machines is not suitable for most molecular machines. One reason for this could be that both types of machinery utilize different underlying abstractions. Mechanical machines can be reduced to their machine elements, which are gears, shafts, bearings, pins, etc. These machine elements are basically arbitrary abstractions of objects that can be used for momentum and torque transmission. Not only mechanical machines share such a common ground, but also molecular machines. The model presented in Chapter 4 represents essentially the machine elements of molecular machines. Binding sites and conformations are nothing else than abstractions of reoccurring phenomena. The difference between the mechanical and the molecular machine elements is that the mechanical abstraction is arbitrary, and the molecular abstraction is made by nature. Unfortunately, these two abstractions are not compatible. Also, it is challenging to find the abstraction made by nature for molecular machines, since the abstraction and therefore the machine elements are not well-defined.

7.4 Future work

The framework implements only the most common biological features of three well-known molecular machines. Although, these features are sufficient for some molecular machines, future work can focus on finding and abstracting more biological features, to support an even broader domain of molecular machines. Such features could be, for example, elasticity in molecular structures or certain degrees of freedom on binding sites [AJL+14].

To make the framework more attractive for biologists, future work should provide a GUI for creating and editing molecular environments. An API as only way to interact with the framework is also a limiting factor for broader usage. Focusing on a more intuitive

way of describing entity behavior, for example via a domain specific language could be also beneficial.

Also other ways of generating behavior descriptions can be examined. For example, deriving a behavior description automatically from example key-frame models. By learning what the animator is doing, an algorithm could try to match the actions with the proposed model.

Future work could also focus on integrating a very large number of molecular machines into one huge environment, e.g. a whole cell. When the environment contains billions of molecular machines, an agent-based approach where each agent is computed individually is not feasible anymore. Thus, a computationally efficient representation of the machinery has to be found, that is applicable on various scaling levels. For example, machines that are hidden or too far away to see do not need to be fully animated. For those, a purely quantitative representation, similar to the approach proposed by Le Muzic et. al. [MPSV14], may be sufficient. Ideally, the representations for all scaling levels can be automatically and seamlessly transformed into each other.

Glossary

adenine A base, that when attached to a nucleotide, is used to encode information in DNA and RNA [AJL+14] . 8

adenosine diphosphate Molecule that is formed when one phosphate is split from ATP [AJL+14]. 14, 95

adenosine triphosphate Molecule composed of adenine, ribose, and three phosphates. The bonds of the phosphates contain chemical energy that can be harnessed by molecular machines. When one phosphate is cleaved, energy is released and ADP is formed [AJL+14]. 13, 95

allosteric activation When the function of a protein is activated by a conformation change [AJL+14]. 13, 47, 51

allosteric inhibition When the function of a protein is inhibited by a conformation change [AJL+14]. 11

amino acid Amino acids are a class of chemical compounds. Some amino acids are used as building blocks for proteins [AJL+14]. 8–10

Brownian motion Molecular particles are in constant motion and the environment is very crowded. Thus, they collide often with each other which results in a random motion, which is referred to as Brownian motion [Goo04]. 12, 13, 17, 44, 46, 82

carbon-monoxide A toxic gas, because it binds much easier to hemoglobin than oxygen [GDHS03]. 14, 50, 51, 71–73

connected component In graph theory, it is a subgraph in with any two vertices are connected to each other by paths, and which is connected to no additional vertices in the supergraph [Wik16]. 37, 39, 41

cryo-electron microscopy Spectroscopy method used to determine molecular structures. The molecules are frozen and fired at with an electron beam. The scattered electrons pass a lens an create a magnified image on a detector [Cal15] . 21

cytoplasm Content of the cell that is enclosed by the cell membrane [AJL+14]. 7, 16

cytosine A base, that when attached to a nucleotide, is used to encode information in DNA and RNA [AJL+14]. 8

deoxyribonucleic acid Double-stranded chain of linked together nucleotides. In the cell it is used to store genetic information [AJL+14]. 5, 95

enzyme A protein that enables or speeds up a specific chemical reaction [AJL+14]. 11, 12, 47

guanine A base, that when attached to a nucleotide, is used to encode information in DNA and RNA [AJL+14]. 8

hydrogen bond A weak noncovalent chemical bond [AJL+14]. 10

ion An atom carrying either a positive or negative charge [AJL+14]. 6, 7, 13, 33

lipid bilayer Structure that consists of two layers of lipid molecules. Cell membranes are made of lipid bilayers [AJL+14]. 6

macromolecule A large molecule consisting of a few thousand atoms, like most proteins and nucleic acids [AJL+14]. 5, 7, 8, 10, 11, 13, 23

macroscopic Scale for objects with a size from centimeters to meters [Goo04]. 1, 12, 13, 48–50

membrane A very thin barrier made of lipids that forms the boundary of a cell or and many parts inside the cell [AJL+14]. 5–7, 9, 13, 15, 43, 46, 52, 75, 92

microtubule Long, cylindrical structures with various functions in cells, like shape regulation and transport highway. Microtubules are built with tubulin molecules [AJL+14]. 16, 17, 53, 55, 82–84, 93

molecular particle A term for a nanoscale object. Can be either a molecule, an atom or an ion. 2, 3, 29, 30, 33, 87

nanoscale Scale for objects with a size of only a few nanometers (billionth of a meter) [Goo04]. 12, 13, 22, 92

NMR spectroscopy Spectroscopy method used to determine molecular structures. Certain atomic nuclei have an intrinsic magnetic moment that aligns in a strong magnetic field. By radio frequency pulses the alignment is disturbed. The realignment of the disturbed nuclei causes them to emit characteristic radio frequency radiation that reflects the local environment of the atom. The data obtained can be used to derive an ensemble of possible structures [Goo04] . 1, 21

nucleic acid Chains of linked together nucleotides. DNA and RNA are nucleic acids [AJL+14]. 7, 8, 21, 92, 93

nucleotide Chemical compound that is used as building block for nucleic acids. Each nucleotide has a base attached that is used to encode information [AJL+14]. 8, 9, 91–93

phosphate A phosphate is a negatively charged chemical. When bound together, by a covalent bond, the bond holds energy because due the negative charge they repel each other. Such bonds are found in ATP [AJL+14]. 13–17, 33, 51–53, 75–78, 80

protein A molecule composed of a chain of linked together amino acids in a specific sequence. The amino acid chain folds into sequence specific 3D shape that determine its function [AJL+14]. 5–13, 15, 16, 21, 22, 25, 46, 58, 76, 77, 91–93

ribonucleic acid Single-stranded chain of linked together nucleotides. In the cell it is used to copy parts of the genetic information from DNA [AJL+14]. 8, 95

ribose A sugar molecule that is frequently found in cells [AJL+14]. 91

ribosome A fundamental molecular machine in every cell. It is able to synthesize proteins as defined by blueprints encoded in RNA [AJL+14]. 9

self-assembly Spontaneous assembly of molecules into structured and stable aggregates [Goo04]. 13

substrate Term for the molecule on which an enzyme acts [AJL+14]. 12

thymine A base, that when attached to a nucleotide, is used to encode information in DNA [AJL+14]. 8

tubulin Protein that serves as subunit to build microtubules [AJL+14]. 16, 17, 40, 82, 92

uracil A base, that when attached to a nucleotide, is used to encode information in RNA [AJL+14]. 8

X-ray crystallography Technique method used to determine the atomic structure of molecules. The molecules must be provided an crystal form. X-rays diffract into many specific directions. The intensity and angles of the diffracted beams can be used to produce a electron density map, from which the atom positions can be derived [Goo04] . 1, 21

Acronyms

2D two-dimensional. 1, 34, 39, 46

3D three-dimensional. 1, 2, 19–22, 25, 29, 32–35, 57–59

ADP adenosine diphosphate. 14–17, 33, 51–53, 75–78, 80, 95, *Glossary:* adenosine diphosphate

API application programming interface. 57, 58, 88, 89

ATP adenosine triphosphate. 13, 14, 16, 17, 33, 50, 53, 55, 75–78, 80, 82, 95, *Glossary:* adenosine triphosphate

CPU central processing unit. 69

DNA deoxyribonucleic acid. 5, 7–9, 21, 22, 91–93, 95, *Glossary:* deoxyribonucleic acid

ePMV Embedded Python Molecular Viewer. 21, 22

GPU graphics processing unit. 22, 23, 69

GUI graphical user interface. 20, 24, 25, 44, 88, 89

JSON JavaScript Object Notation. 57–59

mMaya Molecular Maya. 22

PDB Protein Data Bank. 2, 21–25, 58, 73, 76, 77, 82, 88

RNA ribonucleic acid. 8, 9, 12, 91–93, 95, *Glossary:* ribonucleic acid

Bibliography

[ACZ+12] Raluca Mihaela Andrei, Marco Callieri, Maria Francesca Zini, Tiziana Loni, Giuseppe Maraziti, Mike Chen Pan, and Monica Zoppè. Intuitive representation of surface properties of biomolecules using BioBlender. *BMC Bioinformatics*, 13(4):S16, 2012.

[AJL+14] Bruce Alberts, Alexander Johnson, Julian Lewis, David Morgan, Martin Raff, Keith Roberts, and Peter Walter. *Molecular Biology of the Cell*, volume 6. Garland Science, 2014.

[Aut17] Autodesk. Maya - Computer Animation & Modeling Software - Autodesk. `http://www.autodesk.com/products/maya/`, 2017. Accessed: 2017-03-22.

[BBC+01] Roberto Ballardini, Vincenzo Balzani, Alberto Credi, Maria Teresa Gandolfi, and Margherita Venturi. Artificial Molecular-Level Machines: Which Energy To Make Them Work? *ACCOUNTS OF CHEMICAL RESEARCH*, 34(6):445–455, 2001.

[Ble17] Blender Foundation. The Blender project - Free and Open 3D Creation Software. `http://www.blender.org`, 2017. Accessed: 2017-03-22.

[Bra14] T Bray. The JavaScript Object Notation (JSON) Data Interchange Format Abstract. *Internet Engineering Task Force (IETF) RFC7159*, pages 1–16, 2014.

[BWF+00] H M Berman, J Westbrook, Z Feng, G Gilliland, T N Bhat, H Weissig, I N Shindyalov, and P E Bourne. The protein data bank. *Nucleic acids research*, 28(1):235–242, 2000.

[Cal15] Ewen Callaway. The revolution will not be crystallized: a new method sweeps through structural biology. *nature*, 2015. `http://www.nature.com/news/the-revolution-will-not-be-crystallized-a-new-method-sweeps-through-structural-biology-1.18335`, Accessed: 2017-03-31.

[Col13] OpenStax College, editor. *Biology*. OpenStax College, 2013. http://cnx.org/content/col11448/latest/.

[DKL98] Erik B Dam, Martin Koch, and Martin Lillholm. Quaternions, Interpolation and Animation. *Technical Report DIKU-TR-98/5*, page 103, 1998.

[FHFH15] Katrina T Forest, Christopher P Hill, Katrina T Forest, and Christopher P Hill. ScienceDirect Editorial overview : Macromolecular machines and assemblies : Rise and fall at the molecular level. *Current Opinion in Structural Biology*, 31:vii–viii, 2015.

[Fow03] Martin Fowler. HarvestedFramework. https://martinfowler.com/bliki/HarvestedFramework.html, 2003. Accessed: 2017-02-05.

[FPSF84] G. Fermi, M.F. Perutz, B. Shaanan, and R. Fourme. The crystal structure of human deoxyhaemoglobin at 1.74 Å resolution (PDB ID: 2HHB). *Journal of Molecular Biology*, 175(2):159 – 174, 1984.

[FZC+15] Juan Feng, Shun Zhao, Xuemin Chen, Wenda Wang, Wei Dong, Jinghua Chen, Jian-Ren Shen, Lin Liu, and Tingyun Kuang. Biochemical and structural study of Arabidopsis hexokinase 1 (PDB ID: 4QS7, 4QS8). *Acta Crystallographica Section D*, 71(2):367–375, Feb 2015.

[GDHS03] Des Gorman, Alison Drewry, Yi Lin Huang, and Chris Sames. The clinical toxicology of carbon monoxide. *Toxicology*, 187(1):25 – 38, 2003.

[GDSTR13] Anisah W. Ghoorah, Marie-Dominique Devignes, Malika Smaïl-Tabbone, and David W. Ritchie. Kbdock 2013: a spatial classification of 3d protein domain family interactions. *Nucleic Acids Research*, 42(D1):D389, 2013.

[GDZ+05] D.S. Goodsell, S. Dutta, C. Zardecki, M. Voigt, H.M. Berman, and S.K. Burley. The RCSB PDB "Molecule of the Month": Inspiring a Molecular View of Biology – ATP-Synthase. *PLoS Biol*, 2005. https://pdb101.rcsb.org/motm/72, Accessed: 2017-03-31.

[GKM+15] S. Grottel, M. Krone, C. Muller, G. Reina, and T. Ertl. MegaMol – A Prototyping Framework for Particle-based Visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 21(2):201–214, Feb 2015.

[Goo04] David S. Goodsell. *Bionanotechnology: Lessons from Nature*. John Wiley & Sons, Inc., 2004.

[GVL14] David S. Goodsell, Maria Voigt, and Rob Lowe. Molecular Machinery: A Tour of the Protein Data Bank. http://mm.rcsb.org/, 2014. Accessed: 2017-03-29.

[How01] Jonathon Howard. *Mechanics of Motor Proteins and the Cytoskeleton*. Sinauer Associates, Inc., 2001.

[HPR+13] Robert M. Hanson, Jaime Prilusky, Zhou Renjian, Takanori Nakane, and Joel L. Sussman. Jsmol and the next-generation web-based representation of 3d molecular structure as applied to proteopedia. *Israel Journal of Chemistry*, 53(3-4):207–216, 2013.

[IMS+17] Janet Iwasa, Gael McGill, Piotr Sliz, Ronald Mourant, Mike Pan, and Rise Riyo. Molecular flipbook. `https://www.molecularflipbook.org/`, 2017. Accessed: 2017-01-31.

[Iwa10] Janet H. Iwasa. Animating the model figure. *Trends in Cell Biology*, 20(12):699 – 704, 2010. Special issue - CellBio-X.

[Iwa15] Janet H Iwasa. ScienceDirect Bringing macromolecular machinery to life using 3D animation. *Current Opinion in Structural Biology*, 31:84–88, 2015.

[JAG+11] Graham T. Johnson, Ludovic Autin, David S. Goodsell, Michel F. Sanner, and Arthur J. Olson. ePMV Embeds Molecular Modeling into Professional Animation Software Environments. *Structure*, 19(3):293 – 303, 2011.

[Joh13] Graham Johnson. UCSF Chimera autoPACK result file (.apr) reader: How to load, beautify, and animate HIV. `https://youtu.be/-o_k5DX6Oxc`, 2013. Accessed: 2017-03-22.

[KHH+17] Elizabeth H. Kellogg, Nisreen M.A. Hejab, Stuart Howes, Peter Northcote, John H. Miller, J. Fernando Díaz, Kenneth H. Downing, and Eva Nogales. Insights into the Distinct Mechanisms of Action of Taxane and Non-Taxane Microtubule Stabilizers from Cryo-EM Structures (PDB ID: 5SYC). *Journal of Molecular Biology*, 429(5):633 – 646, 2017.

[KSM+97] F Kozielski, S Sack, A Marx, M Thormählen, E Schönbrunn, V Biou, A Thompson, E.-M Mandelkow, and E Mandelkow. The Crystal Structure of Dimeric Kinesin and Implications for Microtubule-Dependent Motility (PDB ID: 3KIN). *Cell*, 91(7):985 – 994, 1997.

[Lus10] Casey Luskin. Molecular Machines in the Cell. *Discovery Institute*, 2010. `http://www.discovery.org/a/14791`, Accessed: 2017-03-31.

[MAPV15] Mathieu Le Muzic, Ludovic Autin, Julius Parulek, and Ivan Viola. cellview: a tool for illustrative and multi-scale rendering of large biomolecular datasets. In Katja Bühler, Lars Linsen, and Nigel W. John, editors, *Eurographics Workshop on Visual Computing for Biology and Medicine*, pages 61–70. EG Digital Library, The Eurographics Association, September 2015.

[MAX17] MAXON. MAXON - 3D FOR THE REAL WORLD. `http://www.maxon.net/`, 2017. Accessed: 2017-03-22.

[Mic17] Microsoft. .net – powerful open source cross platform development. `https://www.microsoft.com/net`, 2017. Accessed: 2017-02-06.

[MoME05] C. Matthews and American Society of Mechanical Engineers. *ASME Engineer's Data Book*. Engineering Management. ASME Press, 2005.

[MPSV14] Mathieu Le Muzic, Julius Parulek, Anne-Kristin Stavrum, and Ivan Viola. Illustrative visualization of molecular reactions using omniscient intelligence and passive agents. *Computer Graphics Forum*, 33(3):141–150, June 2014. Article first published online: 12 JUL 2014.

[MYY+10] Niloy J. Mitra, Yong-Liang Yang, Dong-Ming Yan, Wilmot Li, and Maneesh Agrawala. Illustrating how mechanical assemblies work. *ACM Transactions on Graphics*, 29(3):58:1–58:12, 2010.

[NSAS08] Robert K. Nakamoto, Joanne A. Baylis Scanlon, and Marwan K. Al-Shawi. The Rotary Mechanism of the ATP Synthase. *Archives of biochemistry and biophysics*, 476(1):43–50, 2008.

[Owe05] G. Scott Owen. *HyperGraph - Computer Animation*. ACM SIGGRAPH Education Committee, 2005.

[Par12a] Rick Parent. Behavioral Animation. In *Computer Animation*, pages 339–363. Elsevier Science, 2012.

[Par12b] Rick Parent. Computer Animation Introduction. In *Computer Animation*, pages 1–32. Elsevier Science, 2012.

[Par12c] Rick Parent. Interpolation-Based Animation. In *Computer Animation*, pages 111–160. Elsevier Science, 2012.

[PDB17] RCSB PDB. PDB Current Holdings Breakdown. `http://www.rcsb.org/pdb/statistics/holdings.do`, 2017. Accessed: 2017-03-20.

[PGH+04] Eric F. Pettersen, Thomas D. Goddard, Conrad C. Huang, Gregory S. Couch, Daniel M. Greenblatt, Elaine C. Meng, and Thomas E. Ferrin. UCSF Chimera A visualization system for exploratory research and analysis. *Journal of Computational Chemistry*, 25(13):1605–1612, 2004.

[Rot16] Yasinee Rotratsirikun. A new way to visualize molecular models with Molecular Flipbook. `http://crastina.se/a-new-way-to-visualize-molecular-models-with-molecular-flipbook/`, 2016. Accessed: 2017-03-22.

[Sha83] Boaz Shaanan. Structure of human oxyhaemoglobin at 2.1 Å resolution (PDB ID: 1HHO). *Journal of molecular biology*, 171(1):31–59, 1983.

[SHR09] Lars Skjaerven, Siv M. Hollup, and Nathalie Reuter. Normal mode analysis for proteins. *Journal of Molecular Structure: THEOCHEM*, 898(1-3):42–48, 2009.

[Spo10] Sponk. Difference DNA RNA. https://commons.wikimedia.org/wiki/File:Difference_DNA_RNA-EN.svg, 2010. Accessed: 2017-03-17.

[SSW+16] Meghna Sobti, Callum Smits, Andrew SW Wong, Robert Ishmukhametov, Daniela Stock, Sara Sandin, and Alastair G Stewart. Cryo-EM structures of the autoinhibited E. coli ATP synthase in three rotational states (PDB ID: 5T4O, 5T4P, 5T4Q). *eLife*, 5:e21598, dec 2016.

[Tec16] Unity Technologies. Unity Manual. https://docs.unity3d.com/540/Documentation/Manual/UnityManual.html, 2016. Accessed: 2017-03-31.

[Tec17] Unity Technologies. Unity – game engine. https://unity3d.com, 2017. Accessed: 2017-02-06.

[The13] TheCISMM. Sketchbio Features, October 2013. https://youtu.be/pT5SFztmgEg, 2013. Accessed: 2017-03-22.

[Too17] Molecular Maya Toolkit. Molecular Maya Toolkit – mMaya. http://www.molecularmovies.com/toolkit/, 2017. Accessed: 2017-01-31.

[UCS12] UCSF Chimera. Commands for Making Animations. http://www.cgl.ucsf.edu/chimera/data/movie-howto-mar2012/movie_examples.html, 2012. Accessed: 2017-03-22.

[VM00] Ronald D. Vale and Ronald A. Milligan. The way things move: Looking under the hood of molecular motor proteins. *Science*, 288(5463):88–95, 2000.

[VMJ00] Ron Vale, Ron Milligan, and Graham Johnson. An Animated Model for Processive Motility by Conventional Kinesin. http://www.scripps.edu/milligan/research/movies/kinesin_text.html, 2000. Accessed: 2017-03-29.

[Wei15] Eric W. Weisstein. Cone. *From MathWorld–A Wolfram Web Resource*, 2015. http://mathworld.wolfram.com/Cone.html, Accessed: 2017-03-27.

[Wik16] Wikipedia. Connected component (graph theory) — wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Connected_component_(graph_theory), 2016. Accessed: 2017-03-31.

[WTHT14] Shawn M Waldon, Peter M Thompson, Patrick J Hahn, and Russell M Taylor. SketchBio: a scientist's 3D interface for molecular modeling and animation. *BMC Bioinformatics*, 15(1):1–17, 2014.

[wwP16] wwPDB. PDBx/mmCIF Dictionary Resources. http://mmcif.wwpdb.org/, 2016. Accessed: 2017-03-31.

[ZNL14] Xiaowei Zhao, Steven J Norris, and Jun Liu. Molecular Architecture of the Bacterial Flagellar Motor in Cells. *Biochemistry*, 2014.

[ZRS+15] Anna Zhou, Alexis Rohou, Daniel G Schep, John V Bason, Martin G Montgomery, John E Walker, Nikolaus Grigorieff, and John L Rubinstein. Structure and conformational states of the bovine mitochondrial ATP synthase by cryo-EM (PDB ID: 5ARA). *eLife*, 4:e10180, oct 2015.