Visualization of molecular machinery using agent-based animation

Daniel Gehrer, Ivan Viola

Institute of Computer Graphics and Algorithms, TU Wien, Vienna, Austria

Abstract

This paper 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 a behavior description 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 and extended. The resulted animations demonstrate that agent-based animation is a viable option for molecular machines.

Keywords:
molecular visualization, molecular machines, agent-based animation, visualization, Unity

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 [1].

Biologists examine molecular machines using X-ray crystallography, NMR spectroscopy and cryo-electron microscopy [2]. Based on the spectroscopy data, biologists develop models on how such machines may work. The gained knowledge then has 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 [3]. 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 [4].

Existing 3D software allows biologists to create animated models of molecular machines, but have also drawbacks for this specific task. This kind of software often requires weeks to months of training and regular use to create a basic molecular animation [5]. Sometimes biologists have to hire programmers and animators to create models and animations [6]. One of the major challenges is to model large molecular assemblies which can involve thousands of molecules. Animating such a large number of objects can be very tedious [5].

The goal of this work is to introduce a fast and effective way of creating animations of molecular machines. The produced animations aim to convey the knowledge on how the animated molecular machines work on a high level. It is not intended to be fully biologically correct, but to illustrate the basic working principles of the animated machine. Since the authors never produced animations of molecular machines before, three molecular machines were visualized and animated from scratch to learn about the visualization and animation process in this particular field. A framework was created that supports the visualization and animation process. The lessons learned by each animated machine were then used to iteratively improve the framework. Although some tools already exist that simplify the visualization and animation process [7, 6, 8], this very basic bottom up approach was used by the authors to learn about the core problems of the process and how to approach them.

The result is a prototypic framework that animates molecular environments using an agent-based model. The model represents an abstract molecular environment. Instead of animating a scene using key-frames, the framework requires the animator to describe the behavior of molecular particles. These molecular particles act as agents that interact with the environment according to their behavior description. To allow the animator to appropriately define a behavior, a suitable level of abstraction of biological phenomena had to be found for the model. The agents can respond to environmental conditions, e.g. the concentration of specific ligands, and can also actively participate by initiating or releasing molecular bonds. A molecular machine is built of molecules [2]. By describing the behavior of each molecule, the functionality of the whole machine can be animated.

The proposed approach provides some advantages compared to existing solutions [7, 6]. The behavior of molecular particles can be described using a biologically inspired model. Thus, the animator can model the animation using biological concepts, instead of using plain key-frames. Furthermore, the behavior has to be defined only once for each type of molecular particle. Since usually the number of distinct of molecular particles is small compared to the total number of particles, this approach requires less effort than a key-frame animation.

2. Related Work

There is a range of tools that support the visualization and animation process. Some of them cover only a small part of the whole visualization process, like providing structural information of molecules [9], or animating molecular structures [10, 11]. But there are also tools that support the biologist through the whole process of importing, animating and rendering molecular environments [7, 6].

The visualization of large molecules is based on scientific data. The Protein Data Bank (PDB) is a publicly available repository for such data [9]. It contains 3D structures of biological molecules. The structures range from tiny proteins and parts of DNA to complex molecular machines [9].

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 [10, 11, 12].

The Embedded Python Molecular Viewer (ePMV) is an open-source plugin for the 3D software Blender, Cinema4D and Maya [10]. 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 [10].

A similar tool is Molecular Maya (mMaya) [11]. It is a free plugin for the 3D software Maya and offers functions to import, build and animate molecular structures. The biologist can open PDB files or download them directly from the Protein Data Bank. Molecular Maya supports various representation forms [11].

BioBlender is a software package for visualizing biomolecules in the open source 3D software Blender [12]. The package extends Blender's capabilities to allow biologists to import molecular structures from PDB and provides notable visualization options for special surface properties [12].

cellVIEW is a tool that solely focuses on efficiently rendering large molecular assemblies [13]. The system 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. cellVIEW integrates various acceleration techniques like a level-of-detail scheme and dynamic generation of DNA strands directly on the GPU. cellVIEW is freely available to be used and extended [13].

Another category of tools supports biologists in creating their own molecular animations [7, 6, 8]. One such tool is the Molecular Flipbook. The intended audience are molecular biologists who want to visualize molecular models to communicate their ideas to their peers, their students or the public [7]. It is only meant for simple molecular models and not for complex or cinematic quality animations. Molecular Flipbook utilizes a simple user interface which allows the user to create plain key-frame animations [7].

SketchBio is another tool that enables biologists to rapidly construct molecular animations [6]. 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 [6].

UCSF Chimera is not only a tool for making simple videos, but a program to interactively visualize and analyze molecular structures [8]. It also makes use of related data like sequence alignments, docking results, trajectories, conformational ensembles, etc. Beside a long list of analysis features, UCSF Chimera can also be used to generate high-quality images and animations [8].

In already existing tools, the animator animates the molecular environment via key-frame animation [7, 6]. There are also works that utilize agent-based approaches, e.g. the work of Le Muzic et. al. [14] and Kolesar et. al. [15]. Agents are autonomous units with a modeled behavior. They are capable of autonomous action and are capable of interacting with other agents [15]. Le Muzic et. al. uses passive agents that are controlled by an omniscient intelligence to ensure quantitative correctness [14]. In contrast, this work uses active agents, since the quantitative correctness is not necessary to understand the working principles of a molecular machine. Kolesar et. al. introduces a combination of three systems, including agents, that is capable of illustrating polymer emergence. However, the system used to describe the production rules for polymers by Kolesar et. al. is not suitable for more general molecular machines, e.g. enzymes [15].

3. Biological Background

All living things are made of cells. Cells are small, membrane-enclosed units filled with water and other chemicals. In an approximate bacterial cell, water makes up 70 % of the cell interior. Macromolecules like DNA, proteins and polysaccharides make up 24 %. The remaining 6 % are phospholipids, small molecules and ions. The composition of an animal cell is similar [16].

Macromolecules are polymers that are constructed by linking smaller subunits (called monomers) together. Especially proteins play a crucial role in cells as they are versatile and can perform thousands of distinct functions. The reason for this versatileness is that a protein is made of a chain of amino acids. Depending on the sequence and the number of amino acids in the chain, the resulting protein can be of very different size and shape. Each amino acid has unique physical properties. Some 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 is suitable for different

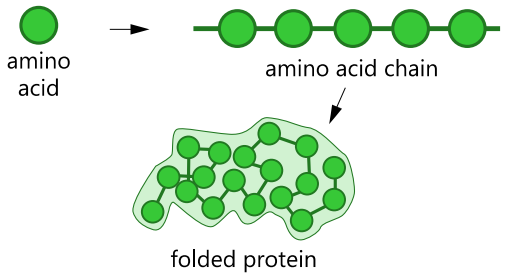

Figure 1: A Protein consists of a chain of linked together amino acids. In a watery environment the chain folds into a stable conformation [16].

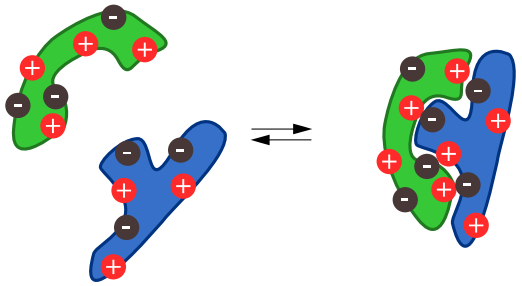

Figure 2: Binding of two molecules due to complementary shape and charges on their surface [16].

tasks. The cell has capabilities to produce proteins on demand, out of amino acid sequence templates encoded in the DNA [16].

In the water environment of a cell, the amino acid chain folds into a conformation of lowest energy, as illustrated in Figure 1. Normally each protein folds into a single stable conformation. This conformation, however, often changes slightly when the protein interacts with other molecules in the cell. These small shifts in shape are crucial for its functionality [16].

Large molecules can bind together through complementary shape and charges on their surfaces. A simple illustration of this effect is shown in Figure 2. This kind of binding can be very specific and allow only certain molecules to bind. When multiple macromolecules bind together, they can form large complexes with multiple moving parts that can perform specific tasks [16]. These complexes are called molecular machines [1].

4. Method

Biologists use 3D animations to communicate the working principles of molecular machines to fellow scientists and students. To create animations, biologists often use standard 3D software to model molecular environments according to their ideas [3]. 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 [7, 6]. This paper proposes another way of producing an animation. Instead of animating 3D objects directly, the biologist describes the behavior of molecular particles on a high level using a biologically inspired model. Each molecular particle interacts with other particles according to their behavior description, which causes the animation. 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.

Since the goal is to communicate working principles of molecular machines via animations, the biologist should not have to focus on low-level physical properties or forces. Hence, the framework aims to allow the biologist to describe the behavior on a level as high as possible. Simultaneously, the chosen level of abstraction also must not limit the biologist in achieving an animation that contains all details necessary to communicate the working principles. To achieve this, an abstract molecular model was created that is used for the behavior description. This model includes common molecular details and features that are discussed later.

To test the agent-based animation approach for molecular machines, a prototypic framework was created which implements the agent-based model. The methodology for developing the framework was driven by three examples of molecular machines. Three molecular machines were chosen and visualized by the authors to learn about the challenges of the animation process. The animations were created from scratch. No tool presented in the related work section was extended. The reason for this design choice is to get an in-depth understanding of the challenges involved and avoid being biased by possible solutions. Only cellVIEW is used for efficient rendering [13].

The framework was extended and improved 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 working principle of the chosen molecular machine is sufficiently visualized, choose another more complex one. Then repeat these steps with the new machine but instead of creating a new framework, extend and improve the existing one, if necessary, so it is capable of animating a broader range of molecular machines. When repeated for a few iterations, the resulting framework should support quite a variety of molecular machines.

4.1. Model

As discussed before, the lessons learned from animating the three molecular machines are used to create an abstract model that provides a suitable abstraction for molecular machines. This model can be used to animate a desired molecular machine in an agent-based way. Agents are suitable, since molecules can be seen as autonomous units that interact with each other in an environment [2]. An overview over the model is illustrated in Figure 3.

The environment in the model represents a small space with molecules in it. It can be seen as the equivalent to a scene in standard 3D software. In biology, the environment is inhabited by molecular particles. In the model, molecular particles

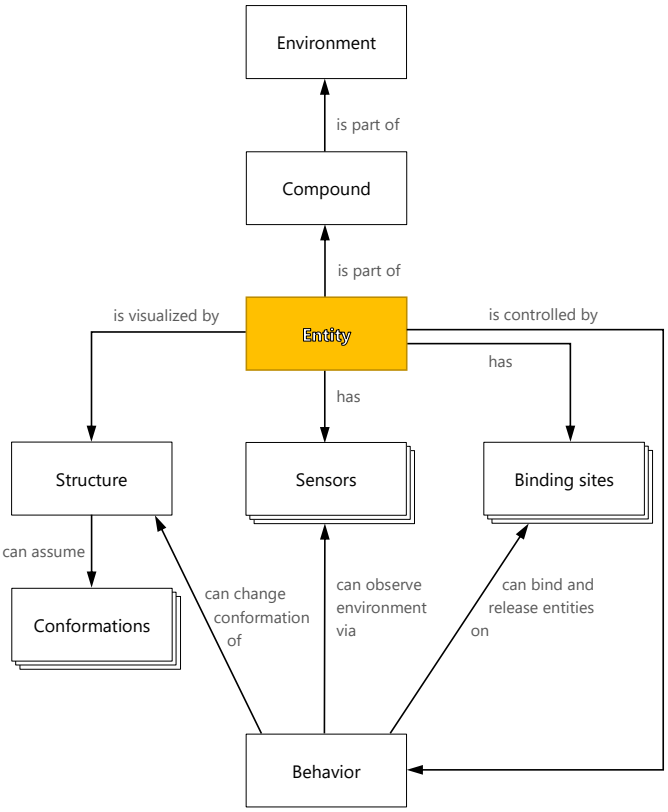

Figure 3: Interaction of model components.

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 autonomous as agents.

In biology, molecular particles move and interact with each other according to physical forces [16]. In existing solutions [7, 6], these effects have to be animated by the user. Instead of animating each molecular particle by hand, the proposed model requires the user to provide a behavior description for each type of entity. The animation is then generated according to the behavior description.

The atomic structure of a molecule is very important in biology. Depending on its size and shape, it can interact with other molecules [16]. This atomic structure is reflected in the model as entity structure. As described in Section 3, the shape of certain molecules can shift slightly, which is important for various processes [16]. Such shapes a molecule can assume, are represented in the model as entity conformations. Each entity type has a set of conformations that it can assume.

In biology, interaction of molecules plays a vital role in the functionality of molecular machines. The interactions are a result of physical attraction and repulsion forces. Since the molecules in the model are represented as active agents, they have to be able to observe the environment, to find interaction partners. Therefore, sensors are introduced to the model. A sensor allows the entity to detect other nearby entities in the environment. It can then decide if it wants to interact with one of them or not. As a result, the molecules can bind together to perform a task, which is also an important phenomena in biology. Molecules have sites with a specific shape and charge profile where other molecules with complementary shape and charge profile can bind [16], as described in Section 3. In the model, such sites are represented as binding sites. Entities that are bound together in the model form a compound.

The following paragraphs describe each concept of the model in more detail:

Entity: An entity represents either a molecule, a part of a molecule, an atom or an ion. Each entity acts as an autonomous agent and interacts with the environment according to its defined behavior. Entities also have defined binding sites and sensors, which are explained in more detail later. Its visual representation is defined by its structure.

Typically molecules are modeled as entities. When a molecule has the ability to split into smaller parts, those parts are modeled as entity. Thus, entities are the smallest, indivisible molecular particles in the environment. What the smallest, indivisible molecular particles are, is dependent on the environment. For example, an environment that visualizes a biological process that reduces adenosine triphosphate (ATP) to adenosine diphosphate (ADP) for power [16], could represent these molecules using two entities: ADP and phosphate. ATP would be represented as an ADP entity bound to a phosphate entity.

Structure: The structure contains the visual representation of an entity. It contains information about all the atoms that the entity is made of. As discussed in Section 3, certain molecules can change their conformation [16]. Predefined conformations an entity can assume are also defined in its structure. A conformation encloses the spatial location of all entity atoms, as well as the locations and directions of binding sites and sensors. An entity structure can change its color depending on the current conformation for visual guidance.

Conformation changes can be initiated by the behavior. To avoid jumpy conformation 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 color are also interpolated linearly.

Sensor: Sensors are used by the entity behavior to find other entities that are nearby. To achieve this, the sensor checks for each entity in the environment if it is located inside a cone at the sensor site. The cone is defined by the sensor position and direction that are relative to the origin of the entity. The apex of this cone is located at the specified position, the direction reflects the sensing direction. The size of the cone is dependent on additional aperture and range properties. Aperture is the angle at the apex and range is the height of the cone.

Binding site: At a binding site, an entity can establish a temporary or lasting bond to a binding site of another entity. Therefore, entities with the ability to bind to other entities must have one or more binding sites.

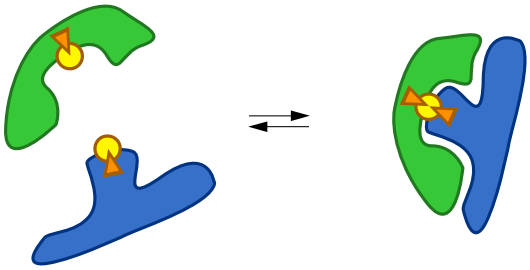

Figure 4: The binding site positions are depicted as yellow circles, the directions as orange arrows. When bound, the positions match and the directions are pointing at each other.

In biology, the binding sites are sites on the molecule surface with a specific shape and charge profile [16], as described in Section 3. In this model, the complex biological representation is reduced to a single point and a direction. When two entities bind together, they form a compound and are aligned to each other depending on the location and direction of their binding sites. The location and direction of a binding site is stored in each conformation of the entity. As a consequence, binding sites can change their location and direction as a result of a conformation change.

The behavior can bind a binding site of the controlled entity to a binding site of another entity and release the bond again. When the binding sites of two entities bind together, 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 only bind to compatible binding sites.

Compound: All entities that are bound together are organized in compounds. Compounds move together and act like one big entity, but in contrast to an entity, a compound can split again into its individual parts. As soon as one or more entities split, they leave the current compound and form a new one. In the model, any entity is always part of a compound. When an entity is not bound to any another entity, it is still part of a compound, but it 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. To enforce these constraints, the position and rotation of each entity is recalculated in every frame. Each compound has an arbitrary root entity. Starting from there, an algorithm traverses along all bindings and recalculates the positions and rotations for each entity.

A binding site is located at a certain position relative to the entity origin. In addition to that position, a direction is provided. When two binding sites are bound, the entities are aligned in such a way that those directions point at each other, and the positions match, as illustrated in Figure 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.

Compounds are not only used to hold entities together but also to locate them spatially in the environment. Since each entity, and also unbound entities, are always part of a compound, the location is defined for compounds and not for entities.

There are three different types how a compound can be located: The first is ”fixed”. The compound is at a static position and cannot be moved. Fixed is used, for example, when a compound should be placed at a certain position in the membrane. The second location type is ”floating in compartment”. The environment can be divided into separated areas called compartments. Compounds that are floating, move randomly inside such a compartment to mimicking Brownian motion. The third location type is ”trajectory”. During compounds are attracted or repulsed by an entity, compounds move along a trajectory. Behaviors can change the location type to move around entities in the environment.

Behavior: The behavior is the agent logic of the entity. Each entity is linked to a behavior which controls its interaction with the environment. The interaction results in movements and conformation changes that make up the animation. The animator describes the entity behavior by using program code. Details are provided in Section 5. 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 determine the concentration of a specific ligand or to find nearby entities. The information obtained from the sensors must then be processed and evaluated. Based on this information, the behavior may updates its inner state and decides 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. Bonds with other entities on the 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. Often, the molecule behavior depends on its current conformation. For example, enzymes that are in an activated conformation behave different than the same enzyme that is not activated. Therefore, the behavior can often be described as state machine.

5. Implementation Details

The framework is based on Unity 5.4.3 [17] and is implemented using the programming language C#. In the framework, the behavior is described using C# program code. For each type of entity a separate behavior description can be provided. The behavior is described in a class that extends the `EntityBehavior` base class. When the animation is started, for each entity a corresponding behavior object is instantiated, that controls the entity.

The behavior can be described by using the model components introduced in Section 4.1. To access the model components in code, the framework includes a dependency injector 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 1.

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

Listing 1: 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 another entity is bound, or to initiate and release bonds. An example definition is shown in Listing 2.

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

Listing 2: 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 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 3.

```
// inject entity
[BoundEntity("rotorSite")]
public Entity RotorEntity { get; set; }

// inject behavior
[BoundEntity("rotorSite")]
public RotorBehavior RotorBehavior { get; set;
    }
```

Listing 3: 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.

5.1. Managing Inner State

Entity behaviors are implemented as state machines. Each state of the state machine has a dedicated code block. 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 every frame. The current state of the state machine can be changed with the `SetState` method. A simple example is shown in Listing 4.

```
[EntityBehaviorId("atpSynthase.f0c")]
public class F0cBehavior : EntityBehavior
{
  [BindingSite("protonSite")]
  public BindingSite ProtonSite { get; set; }

  [State(InitialState = true, Conformation = "
      tensed")]
  public void Tense()
  {
    // Change to Relax-State as soon a proton
        is bound
    if (ProtonSite.IsBound)
    { SetState(Relax); }
  }

  [State(Conformation = "relaxed")]
  public void Relax()
  {
    // Change to Tense-State as soon the
        proton leaves
    if (!ProtonSite.IsBound)
    { SetState(Tense); }
  }
}
```

Listing 4: Simple example of a behavior description.

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. 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.

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.

5.2. Observing the Environment

The surrounding of the entity can be observed by using the entity's sensors. To make a sensor accessible in code, the sensor object must be injected into a property using the `Sensor` attribute, as shown in Listing 1. The injected sensor object provides various methods to find nearby entities. The available methods are shown in Listing 5.

`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 method 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 method that finds the entity that is the nearest to the sensor and

```
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: Methods that are implemented by a sensor.

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 return 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.

5.3. 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 5.2. To move around compounds, the behavior can define a trajectory. 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 6.

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 7.

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

```
// 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 6: Moving a proton entity to a local binding site and then bind.

Listing 7: 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 entities 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 provided, to move the released entity away.

6. Results

The result of this work is a framework that implements the model proposed in Section 4.1. The framework is implemented using the Unity game engine [17] and utilizes cellVIEW to render the environment [13]. Furthermore the implemented framework is used to visualize and animate three molecular machines.

In order to being able to animate more and more complex molecular machines, new features were added to the framework during its emergence. Three molecular machines animated to test the framework and simultaneously serve 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 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 conformation is induced. This structural change makes it easier for the next oxygen to bind. Thus, binding the first oxygen is the hardest, but from there it gets easier and easier [16].

In the lung, 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 oxygen level 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 [16].

Framework Model

The Hemoglobin is modeled as one entity with 5 conformations: deoxygenated, 25 % oxygenated, 50 % oxygenated, 75 % oxygenated and 100 % oxygenated. The structure is imported from the two PDB entries 1HHO (100 % oxygenated) and 2HHB (deoxygenated) [18, 19]. Intermediate conformations are linearly interpolated. The Hemoglobin model contains four binding sites and four sensors. One sensor for each binding site.

The behavior is programmed to evaluate the oxygen concentration on each sensor. If the concentration exceeds a certain bind-threshold, the nearest oxygen molecule is attracted and bound to the binding site. The conformation changes depending on the sum of oxygen molecules bound on all binding sites. Moreover, the higher the number of bound oxygen molecules, the lower is the bind-threshold for the next one.

If the measured oxygen concentration on a sensor drops below a release-threshold, the oxygen on this binding site is released again. Like the bind-threshold, the release-threshold is also dependent on the number of oxygen molecules currently bound. A screenshot of the visualization is shown in Figure 5.

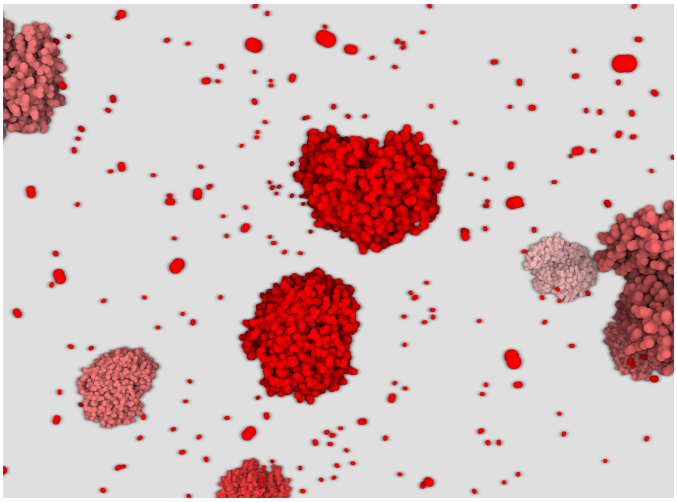

Figure 5: Hemoglobin and oxygen environment. The redder the Hemoglobin, the more oxygen is bound.

6.2. ATP-Synthase

Most cellular processes are powered by adenosine triphosphate (ATP). When energy is needed, ATP breaks into adenosine diphosphate (ADP) and phosphate. This liberates energy that was stored in the bond. The liberated energy is then used to power the 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 gradient over a membrane [2].

ATP-Synthase 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 [2]. A frame of the created animation and a short explanation of the parts is shown in Figure 6.

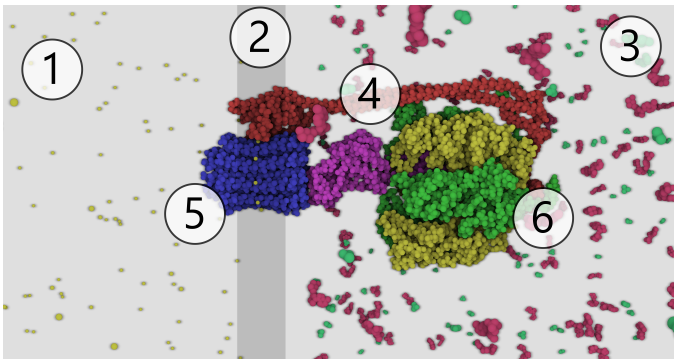

Figure 6: ATP-Synthase. (1) Protons outside membrane, (2) membrane, (3) ADP and phosphate inside membrane, (4) stator, (5) rotor consisting of F_0c proteins, (6) $F1$ part consisting of $F_1\alpha$ and $F_1\beta$ proteins.

Framework Model

Since this machine is too complex to model using a single entity, the model is composed of different parts. One entity forms the basic frame for the machine. It provides binding sites for the stator, rotor, three $F_1\alpha$ entities and three $F_1\beta$ entities. The parts are shown in Figure 6. The frame also has a sensor attached that senses to the outside of the membrane enclosed area. As ATP is not atomic, it is not modeled as a single entity, instead ATP is represented as ADP bound to phosphate.

The behavior for the ATP-synthase is described like this: First, use the sensor to find a proton from outside the membrane. Bind it to the F_0c next to the stator. Rotate the rotor by one step. Release the proton bound to the F_0c that is now next to the stator and release it to the inside of the membrane. These simple steps cause the rotor to rotate. The F_1 parts act dependent on the rotation progress. First, they bind to separated ADP and P_i entities. In the next step, these two entities are bound

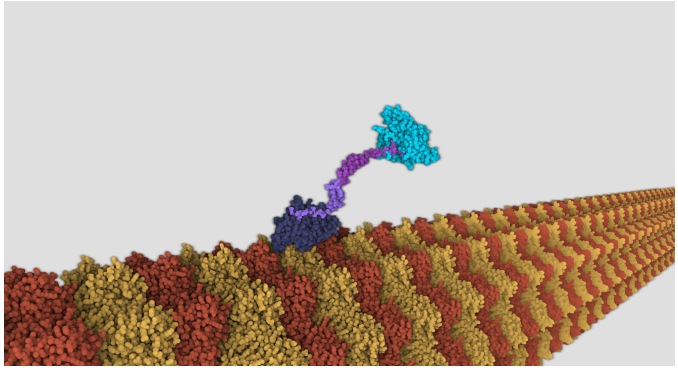

Figure 7: Frame of the kinesin animation. The kinesin walks along the microtubule.

together. In the third step, the regenerated ATP is released and the cycle repeats.

The PDB entries used for the ATP-synthase are 5T4O, 5T4P and FT4Q [20].

6.3. Kinesin

Kinesin is a molecular machine that moves along a microtubule in a walking-like movement. In biology, cells use kinesin to transport a cargo across the cell. For simplicity reasons, cargo is omitted in this animation. A frame of the animation is shown in Figure 7.

Framework Model

The microtubule structure is imported from the PDB structure 5SYC [21], and the kinesin from the PDB structure 3KIN [22].

The kinesin behavior is controlled by its two heads. Each head has a sensor attached, that looks for the microtubule. As soon as a head finds a microtubule binding site, a series of states are run through: When 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. Then, the bound head the neck-linker, that connects the two heads, is pulled towards the bound head by rotating the binding site where the neck-linker is bound, which causes the other head to be thrown forward. The unbound head now is near enough to microtubule to detect and bind to it, at a site that is more forward. Then all steps repeat with the other head. Both heads alternate performing these steps, which results in the walking-like movement.

7. Discussion

The framework introduces a novel approach for animating molecular machines. In already existing tools, the animator animates the molecular environment via key-frame animation [7, 6]. Other agent-based approaches focus on other goals than illustrating working principles of molecular machines [14, 15]. In this 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 types is small compared to the total number of entities. Since the behavior description only has to be formulated once for each entity type, the animation scales easily, 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 prototypic 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-base animation is a viable option for visualizing molecular machines.

7.1. Level of Abstraction

The model proposed in this paper, provides an abstraction of biological phenomena to simplify the animation process. 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 [7, 6]. 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 result from physical properties. Such a description is very abstract and highly reusable. However, it gives very less and 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 an 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 model that allows the animator 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, 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.

7.2. Limitations

The framework does not include a graphical user interface (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 application programming interface (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 [23].

7.3. 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 [16].

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.

Acknowledgments

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.

References

[1] Ballardini, R., Balzani, V., Credi, A., Gandolfi, M.T., Venturi, M.. Artificial Molecular-Level Machines: Which Energy To Make Them Work? ACCOUNTS OF CHEMICAL RESEARCH 2001;34(6):445–455.

[2] Goodsell, D.S.. Bionanotechnology: Lessons from Nature. John Wiley & Sons, Inc.; 2004. ISBN 9780471469575.

[3] Iwasa, J.H.. Animating the model figure. Trends in Cell Biology 2010;20(12):699 – 704. Special issue - CellBio-X; URL http://www.sciencedirect.com/science/article/pii/S0962892410001558.

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

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

[6] Waldon, S.M., Thompson, P.M., Hahn, P.J., Taylor, R.M.. SketchBio: a scientist's 3D interface for molecular modeling and animation. BMC Bioinformatics 2014;15(1):1–17. 1407.3145.

[7] Iwasa, J., McGill, G., Sliz, P., Mourant, R., Pan, M., Riyo, R.. Molecular flipbook. https://www.molecularflipbook.org/; 2017. Accessed: 2017-01-31.

[8] Pettersen, E.F., Goddard, T.D., Huang, C.C., Couch, G.S., Greenblatt, D.M., Meng, E.C., et al. UCSF Chimera A visualization system for exploratory research and analysis. Journal of Computational Chemistry 2004;25(13):1605–1612.

[9] Berman, H.M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T.N., Weissig, H., et al. The protein data bank. Nucleic acids research 2000;28(1):235–242.

[10] Johnson, G.T., Autin, L., Goodsell, D.S., Sanner, M.F., Olson, A.J.. ePMV Embeds Molecular Modeling into Professional Animation Software Environments. Structure 2011;19(3):293 – 303. URL http://www.sciencedirect.com/science/article/pii/S0969212611000608.

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

[12] Andrei, R.M., Callieri, M., Zini, M.F., Loni, T., Maraziti, G., Pan, M.C., et al. Intuitive representation of surface properties of biomolecules using BioBlender. BMC Bioinformatics 2012;13(4):S16.

[13] Muzic, M.L., Autin, L., Parulek, J., Viola, I.. cellview: a tool for illustrative and multi-scale rendering of large biomolecular datasets. In: Bühler, K., Linsen, L., John, N.W., editors. Eurographics Workshop on Visual Computing for Biology and Medicine. EG Digital Library; The Eurographics Association. ISBN 978-3-905674-82-8; 2015, p. 61–70. URL https://www.cg.tuwien.ac.at/research/publications/2015/cellVIEW_2015/.

[14] Muzic, M.L., Parulek, J., Stavrum, A.K., Viola, I.. Illustrative visualization of molecular reactions using omniscient intelligence and passive agents. Computer Graphics Forum 2014;33(3):141–150. Article first published online: 12 JUL 2014; URL https://www.cg.tuwien.ac.at/research/publications/2014/lemuzic-2014-ivm/.

[15] Kolesar, I., Parulek, J., Viola, I., Bruckner, S., Stavrum, A.K., Hauser, H.. Interactively illustrating polymerization using three-level model fusion. BMC Bioinformatics 2014;15(1):345. doi:\bibinfo{doi}{10.1186/1471-2105-15-345}. URL http://dx.doi.org/10.1186/1471-2105-15-345.

[16] Alberts, B., Johnson, A., Lewis, J., Morgan, D., Raff, M., Roberts, K., et al. Molecular Biology of the Cell; vol. 6. Garland Science; 2014.

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

[18] Shaanan, B.. Structure of human oxyhaemoglobin at 2.1 resolution (PDB ID: 1HHO). Journal of molecular biology 1983;171(1):31–59.

[19] Fermi, G., Perutz, M., Shaanan, B., Fourme, R.. The crystal structure of human deoxyhaemoglobin at 1.74 resolution (PDB ID: 2HHB). Journal of Molecular Biology 1984;175(2):159 – 174. doi:\bibinfo{doi}{http://dx.doi.org/10.1016/0022-2836(84)90472-8}. URL http://www.sciencedirect.com/science/article/pii/0022283684904728.

[20] Sobti, M., Smits, C., Wong, A.S., Ishmukhametov, R., Stock, D., Sandin, S., et al. Cryo-EM structures of the autoinhibited E. coli ATP synthase in three rotational states (PDB ID: 5T4O, 5T4P, 5T4Q). eLife 2016;5:e21598. doi:\bibinfo{doi}{10.7554/eLife.21598}. URL https://dx.doi.org/10.7554/eLife.21598.

[21] Kellogg, E.H., Hejab, N.M., Howes, S., Northcote, P., Miller, J.H., Daz, J.F., et al. 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 2017;429(5):633 – 646. doi:\bibinfo{doi}{http://dx.doi.org/10.1016/j.jmb.2017.01.001}. URL http://www.sciencedirect.com/science/article/pii/S0022283617300153.

[22] Kozielski, F., Sack, S., Marx, A., Thormhlen, M., Schnbrunn, E., Biou, V., et al. The Crystal Structure of Dimeric Kinesin and Implications for Microtubule-Dependent Motility (PDB ID: 3KIN). Cell 1997;91(7):985 – 994. doi:\bibinfo{doi}{http://dx.doi.org/10.1016/S0092-8674(00)80489-4}. URL http://www.sciencedirect.com/science/article/pii/S0092867400804894.

[23] Skjaerven, L., Hollup, S.M., Reuter, N.. Normal mode analysis for proteins. Journal of Molecular Structure: THEOCHEM 2009;898(1-3):42–48. doi:\bibinfo{doi}{10.1016/j.theochem.2008.09.024}. URL http://dx.doi.org/10.1016/j.theochem.2008.09.024.