FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Modeling Microorganisms

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Lukas Mitterhofer
Matrikelnummer 1226847

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ivan Viola, Associate Prof. Dipl.-Ing. Dr.techn.
Mitwirkung: Peter Mindek, Dr.techn.
Tobias Klein, M.Sc.

Wien, 14. März 2017

Lukas Mitterhofer

Ivan Viola

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Modeling Microorganisms

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Lukas Mitterhofer
Registration Number 1226847

to the Faculty of Informatics

at the TU Wien

Advisor: Ivan Viola, Associate Prof. Dipl.-Ing. Dr.techn.
Assistance: Peter Mindek, Dr.techn.
Tobias Klein, M.Sc.

Vienna, 14th March, 2017

Lukas Mitterhofer	Ivan Viola

Erklärung zur Verfassung der Arbeit

Lukas Mitterhofer
Kienmayergasse 45/14, 1140 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, 14. März 2017

Lukas Mitterhofer

Danksagung

Ich möchte mich an dieser Stelle bei Professor Ivan Viola für seine Hilfe im Zuge des Projekts, für die konstruktive Kritik und die Vorschläge für weitere Verbesserungen, bedanken. Des Weiteren möchte ich mich bei Peter Mindek und Tobias Klein bedanken, welche mich während der Implementierungs-Phase unterstützt haben. Als originäre Entwickler des Tools cellView konnten sie mir wichtige Informationen mitteilen, was zu höherer Qualität der Endversion wesentlich beigetragen hat.

Acknowledgements

I would like to thank Professor Ivan Viola for his guidance through the whole project, the constructive critics and the suggestions for further improvements. I also want to thank Peter Mindek and Tobias Klein for their support during the implementation phase, as contributors to the original cellView tool they provided additional information which greatly helped developing the final version of the cellView editor.

Kurzfassung

Die Modellierung von Mikroorganismen ist eine aufwändige Aufgabe, wenn Biologen visuelle Repräsentationen erstellen möchten. Um solche Strukturen zu modellieren, muss jedes Molekül an seiner entsprechenden Position platziert werden. Für komplexe Organismen kann diese Aufgabe sehr viel Zeit beanspruchen, daher ergibt sich die Notwendigkeit eines verbesserten Modellbildungsansatzes. Es wäre möglich, dieses Problem durch ein Regel-basiertes System zu lösen, allerdings folgt die Natur selten eindeutig festgelegten Regeln. Aus diesem Grund ergibt sich die dem Projekt zugrundeliegende Idee, molekulare Gebilde (zum Beispiel Proteine oder Lipide) basierend auf statistischen Schätzungen und Verteilungen zu platzieren. Ein Entscheidungsbaum evaluiert die Handlungen des Anwenders und lernt aus ihnen, getroffene Annahmen werden im Baum abgelegt und für gleichartige Moleküle neu evaluiert, was die komplette Struktur reorganisiert. Ziel ist es, dem Anwender zu ermöglichen, komplexe Zellteile mit minimaler Anzahl an notwendigen Schritten zu modellieren. Nicht nur die Platzierung ist ein Kriterium, auch die Orientierung in Richtung eines Referenzpunktes, auch Cluster-Bildung, unterschiedliche Verteilungen und andere Interaktionsmöglichkeiten in einem Echtzeit-Editor sollen die Modellierung von Mikroorganismen deutlich verbessern.

Abstract

Modeling of microorganisms is a cumbersome task, when biologists want to create a visual representation of a certain microorganism. To correctly model structures on atomic resolution, each molecule (for example proteins and lipids) has to be placed at its correct position. For microorganisms of larger dimensions the modeling process takes a very long time, at this point an improved modeling approach is required. It would be possible to create a rule-based modeling approach, but usually rules restrict the final outcome and produces repeating patterns. Therefore a tool that places molecules based on statistical evaluations and foresight was the main idea behind this project. The tool should allow for modeling complex organisms on molecular resolution by placing a minimal amount of examples and generalizing similar entities. A decision tree as learning structure evaluates the user's actions, learns from them and reorganizes the whole structure. With this approach the user should be able to model complex cellular structures in as few steps as possible, also more complex actions, such as orientation towards a certain reference point, clusters, varying distribution united a real-time editor should improve the modeling task significantly.

Contents

CHAPTER 1

Introduction

Current molecular modeling tools like cellPACK [JAAA+17] allow the manual placement of molecules to create a visual representation of a microorganism. This task is very cumbersome if the microorganism consists of thousands of molecules. In addition it is necessary to gather all the information to create a plausible visualization. To shorten the amount of time needed in order to create a model, the goal of this thesis is to create a tool, which interprets the user's actions, feeds the information into a learning structure, and applies the transformation to molecules of the same type. Typically microorganisms consist of many different macromolecules, but they also contain many of the same type, which may be found in similar locations. To precise the final position of the molecules several actions might have to be interpreted, but the adaption is still faster than the manual placement of each molecule on its own. To speed up the modeling process and provide an intuitive way to fulfill this task some general assumptions have been made. The user starts with the formation of compartments the cell consists of. Typically the microorganism's outermost compartment is the membrane, more compartments, such as for instance a capsid, can be added too. After creating the overall structure on mesoscale, the user can switch to nanoscale and add macromolecules to the microorganism. These molecules can be located in very specific regions of the microorganism, at this point a learning structure, namely a decision tree, interprets the user's actions and applies the transformations, in terms of relocation and reorientation, to molecules of the same type. The used data comes from the RCSB Protein Data Bank [RU17]. The basic components of microorganisms, for example proteins, are measured through procedures like the X-ray crystallography, which determines the atomic and molecular structure. The measured data of proteins is available at the RCSB Protein Data Bank as so called PDB files. These files can be downloaded within the tool and used to populate the microorganism and create a plausible representation, which may be used for educational purposes, but also to gather a better insight into cell routines and their composition.
There exist many rule-based modeling approaches, especially in the field of modeling and

simulating biochemical systems and cell signaling in particular. These approaches use a set of rules that indirectly specify a mathematical model. The relations are typically represented by—if this than that—rules. The definition of rules is hard, especially because rules typically result in repeating patterns, which usually do not occur in nature. There exist approaches, which define certain rules through algorithms, which aim at numerical approximation of functions [SBV98]. The resulting fuzzy systems require expert knowledge and operator training. A user is able to reach an optimal outcome through targeted interventions, but such tools are not usable without further knowledge about the tool itself and its behavior.
Rule-based approaches are well fitting in many domains, but due to the fact, that the nature does not always follow certain rules and those tools are hard to use for a layman. We wanted to support the process by analyzing the user input and generating statistical estimations about the target position of similar elements in the scene. The main benefit of a statistical approach is, that each action, performed by the user, has a weighted influence to the whole structure. As a simple example, if the user wants to place all proteins of a certain type inside a cellular compartment, the only thing he has to do, is move one of those proteins into this compartments, all the others will follow because there is no alternate information about the placement. If he then decides that a third of them is also found outside the membrane he has to place one outside and again one inside. This is a simple example for the placement, but also the rotation towards a point of reference, the formation of clusters between the same or different molecules is considered and evaluated in the background. With this approach the modeling process is sped up significantly. The original cellView shows a visualization of a HIV virion, after analyzing the composition of this HIV virion it was recreated with high similarities in about 15 minutes. The possibilities of extensions are widely spread and are very promising.

CHAPTER 2

Related Work

The project is an extension to the existing software cellView, developed at the TU Wien. cellView is a Unity-based tool [Tec17] that was designed to load recipes of microorganisms, modeled with cellPACK [JAAA+17]. These recipes were generated after the heavy modeling process to load these structures with visualization tools such as cellView. So the software was tailored for the visualization of such files and provides many exploration features. The modeling of new structures was not possible up to now, but there were already interfaces which made it possible to extend it. With that approach the tool becomes more interactive, a user has the possibility to create new structures by loading PDB files and insert certain amounts into the scene. By the placement of single proteins or lipids, in the following called ingredients, a decision tree evaluates actions and adapts its distribution functions. The other ingredients of the same type are then evaluated against the decision tree to simplify the modeling process.

2.1 Rendering Techniques

A very important requirement for a modeling tool is to support processing in real time. The tool is an extension to cellView, so it greatly benefits from its optimized rendering pipeline. Different rendering techniques are applied to boost the rendering performance, allowing to implement lots of new features without losing the real-time aspect. The level of detail technique is used to overcome the problem of processing each atom per frame by reducing the overall amount of atoms per ingredient depending on the distance to the viewer. This approach is depicted in Figure 2.1.

In addition the atom spheres are not represented by solid geometry, but with billboards, a technique, that replaces complex shapes by a 2D texture, placed at the atomic position in the scene and oriented towards the viewer. With this approach the amount of triangles needed for a perfectly round sphere is reduced from some hundreds to only two. To

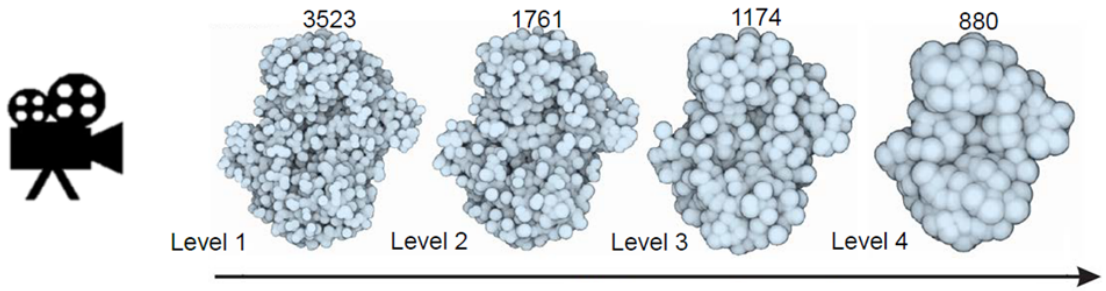

Figure 2.1: Illustration of a dynamic Level-of-Detail, where the amount of atoms is reduced with increasing distance to the viewer.

improve the visual output and enrich it by a better depth perception the ambient occlusion shading technique is applied. With ambient occlusion the amount of light that reaches a given surface point is calculated. This technique enhances the final quality, structures such as tubes or a deformed surface are better comprehensible by the viewer. With these important rendering techniques scenes consisting of millions of atoms can be rendered in real time.

2.2 Machine Learning

Michie, Spiegelhalter and Taylor wrote an interesting paper in the field of statistical machine learning in 1994. Although many things have changed in the past 20 years and therefore the discussion about the performance is not as important as back then, the comparison of different classification approaches is still very useful. As stated in their article, the usage of decision trees for the task of machine learning is widely spread, if a decision tree would be allowed to grow without limits to any number of leaves, the tree would possibly classify the given data with maximal accuracy. Another motivation for the usage of decision trees was their understandability, the state is much easier to analyze than for example a neural network [MST94].
Engineers have been trying for a long time to understand and emulate the human brain. The so called artificial neural networks are inspired by the human brain and try to approximate it roughly. These networks consist of many interconnected nodes which are the so called neurons (from the neurons of the human brain). These graphs propagate an input based on the edges and their weights to the next neurons and can be interpreted as—if this then that—rules. They are very popular in the field of machine learning and would have been a valuable candidate for our tool, therefore it had to be mentioned. For further interest the paper of Michie et al. [MST94] is highly recommendable.

2.3 Procedural Modeling

Procedural modeling is a powerful technique in computer graphics. It is currently used to model trees and buildings, but also large objects or collections of similar objects such as terrains and cities. Procedural modeling makes use of a predefined set of rules, which produce a certain outcome. These rules may be changed and adapted to produce the desired output. Procedural modeling is based on the observation, that things, such as plants, can be generalized and generated with slight variations. The so called Lindemayer-System (L-system) [PL12] is basically a formal grammar, which contains a set of production rules and perform a parallel rewriting process to generate similar objects. This system is used in example for trees and works really well. The generalization of microorganisms is not necessarily rational, although similar procedural modeling approaches could also be applicable for molecular modeling, the drawbacks are crucial. There is a known lack of control due to the cryptic description of the rules and the modeled relations between the models [BŠMM11], in addition the results are hardly predictable. In molecular modeling these rules would be similarly hard to formalize, and would not be valid for any microorganism that may be modeled in the future. To avoid these restrictions a statistical modeling approach is appealing, mainly because we do not have to make any assumptions about the final outcome, the system is trained during the modeling process. This process needs user input, depending on the desired outcome it may need much input, but the final system is adapted and tailored to the current model.

CHAPTER 3

Methodology

In this chapter some of the most important approaches, applied in the modeling tool are discussed, benefits and drawbacks are shown and also possible improvements are stated. The structure of a microorganism can be described on several scales. On mesoscale large structures, so called compartments, and the hierarchy they form, can be described by geometric shape modeling. On nanoscale the positions and orientation of individual macromolecules can be described by probabilistic modeling. Our approach deals specifically with the modeling on nanoscale, which depends on the provided geometric description of the compartments.

3.1 Compartments

A microorganism's outermost component is the membrane, the content of the cell is completely within the membrane. To further subdivide a cell in smaller regions new compartments can be introduced, which form a hierarchy. There exist several physical compartments in the literature, for example the eukaryotic cell, which has a nucleus and organelles as subunits in the cell, and cytoplasm within the membrane, but outside the nucleus. With this knowledge the first step in the creation process was clear, a user should initially model only these compartments to get a feeling of the basic structure before filling these compartments with different proteins or fluids such as the cytoplasm.

The membrane of the prokaryotic- and the eukaryotic cell in Figure 3.1 have an overall shape of a sphere, also the nucleus of the eukaryotic cell has the shape of a sphere. In general it would be necessary to provide and support a large variety of shapes, therefore we made use of the signed distance function, which can be adapted to mathematically describe geometric surfaces. As reference implementation we provide the two shapes of the prokaryotic- and the eukaryotic cell. The compartments are completely represented by these signed distance functions and form a hierarchy in the scene, the membrane is the so called root compartment and contains the complete cell interior. The exterior

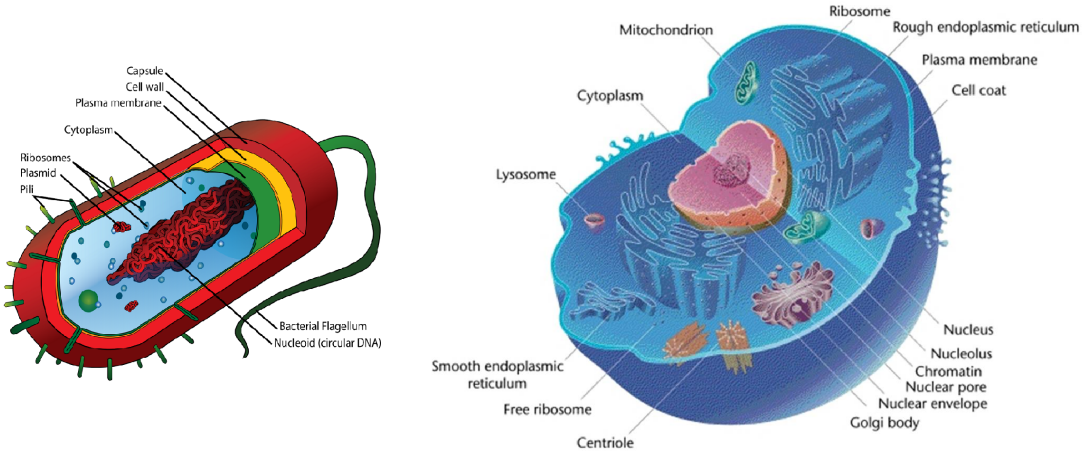

Figure 3.1: Exemplary image of a prokaryotic cell (left) [Med17] and an eukaryotic cell (right) [Wik17]. Note that these cells are normally not equally sized, the prokaryotic cell typically only has about 1/10.000 of the volume of an eukaryotic cell.

can be modeled too to show the region of occurrence, such as for instance the human blood. Each compartment inside the membrane is a child of the membrane, the space between the membrane and other compartments such as the nucleus, belongs to the root compartment and each space can be filled individually.
In the beginning a different approach was used, the compartment was represented by a semi-transparent mesh. This approach led to several inconveniences, in the initial step of the modeling process the user had to create the compartments with meshes. After the compartment creation the user had to place membrane molecules on the surface to populate it. The calculation of intersections with the mesh were unhandy, which led to the usage of signed distance functions. With the introduction of these functions in combination with meshes it was hard to keep the functions in line with the mesh in terms of scale and rotation. Therefore the approach was dropped and the compartment was fully represented by the signed distance function, forcing the user to select a membrane molecule in the beginning of the modeling process, which made it also easier and faster for the user to model certain structures. The signed distance functions also have the benefit that they are really fast in terms of evaluation time. Intersections with surfaces and the distance to the surface are important to adapt the correct leaf nodes of the decision tree, therefore this approach has been used.

3.2 Implicit Surfaces

When modeling the compartments of a microorganism we needed an efficient way to represent those compartments, in terms of visual representation, but also in terms of intersection testing and distance evaluation. The main reasons why implicit representations are used, are the fast evaluation for intersection and distance calculation and inside/outside tests for any given point. In addition implicit representations enable computing of smooth transitions and blending between different types of surfaces, simply by linear interpolation between two surfaces [BW90]. By preserving the continuity of two different surfaces seamless transitions between different molecular surfaces can be achieved. These benefits are hard to achieve in real time with boundary representations between geometric models [PJR$^+$14]. Although we did not make use of the full potential of implicit surfaces yet, a good foundation is laid for future improvements and extensions.

In general an implicit surface is a surface in Euclidean space, defined by an equation $F(X, Y, Z) = 0$. An implicit surface consists of the whole set of zeros of the function, so it is basically a mathematical representation of a given surface.

Examples for implicit surface equations:

- plane: $x + 2y - 3z + 1 = 0$
- sphere: $x^2 + y^2 + z^2 - 4 = 0$
- torus: $(x^2 + y^2 + z^2 + R^2 - a^2)^2 - 4R^2(x^2 + y2) = 0$

3.2.1 The Signed Distance Function

Distance functions can theoretically approximate any shape. Primitive shapes such as spheres, ellipsoids, boxes and cones can be represented by a simple and exact signed distance function. For more complex shapes the distance functions have to be adapted or combined with union, intersection and subtraction [PV], also deformations are possible. For some complex shapes the distance function is only an approximation.
As previously discussed the compartments are not represented by a solid geometry, but by a signed distance function. In general, distance functions define the distance between two elements. Signed distance functions can be used to evaluate for any given point in space if it lies either inside an object, on the surface or outside. The signed distance functions also have the benefit that they are really fast in terms of evaluation. To be more general, we can describe a shape by a function $f(p)$. The function result indicates for each point p if $f(p) > 0$ the point lies outside the object, if $f(p) < 0$ the point is within the object and if $f(p) = 0$ the point lies exactly on the surface.

The usage of implicit surface descriptions is simplifying the implementation significantly, when the user wants to place certain molecules in certain areas, the algorithm simply has to check the result of the compartment's signed distance function, intersections with the children (or sub-compartments) are fast and easy to calculate. The signed distance functions have to be adapted to correctly handle also scaled and rotated shapes.

3.3 Surface Points of Implicit Surfaces

After introducing the signed distance functions, which can be evaluated to check whether a point lies inside, outside or on an implicit surface, there is still the problem of finding a point which lies on the surface. The task of finding a surface point is important for instance when a compartment is created and populated by a certain macromolecule. The initial approach to find surface points was to take random samples inside the bounding box of the compartment, evaluate the signed distance function for that certain point, if the result is zero or within some threshold epsilon, the point is returned, otherwise continue with the random sampling process. Although this approach did not result in a significant performance loss, it had to be optimized, if the bounding box was large and the epsilon was small it resulted in a noticeable delay. This approach was optimized by taking a random sample within the bounding box and calculating the direction vector from the compartments center to the point. By using the result of the signed distance function and multiplying the value with the normalized direction vector the point approaches the surface of the compartment. After a point within a certain threshold epsilon to the surface is reached a surface point is found. To be sure that the resulting surface point is actually the closest point the same procedure is evaluated for each point in the direct neighborhood of the initially found surface point. If the distance to our point decreases a closer point has been found, otherwise the algorithm terminates.
This approach works very well with spherical shapes, for the capsule, which is also used as a compartment, the resulting population of the surface is not normally distributed. The reason is that a point is more likely to be in the cylinder of the capsule than in the hemispheres. With the application of a collision detection algorithm the final output is indistinguishable from the random sampling approach, without a collision detection the resulting images may contain holes and clusters.
The random sampling approach finds a surface position for a given ingredient with an average of about 250 samples with a high precision. This results in 250 evaluations of the signed distance function of the corresponding compartment to find a destination for one ingredient. The before mentioned algorithm to find the closest surface point finds a position with higher precision with the first attempt, making it much faster than the random sampling.

3.4 Decision Tree

After describing the methodology of the compartment structure and their core functionality on mesoscale, the transformations of the macromolecules have to be handled. The modeling process cannot be done molecule by molecule, as there can be several thousands of molecules in a microorganism. Therefore we need a learning structure, which takes the transformations of the modeling process as input and adapts to these transformations, so they can be considered and applied to macromolecules of the same type. The main reason the decision tree has been used as a machine learning approach, is that the decision tree is easier to understand and interpret as for example a neural network, moreover it was a

convenient way to map the hierarchy, formed by the compartments, to the decision tree. Another benefit is, that decision trees allow the addition of new evaluation criteria, which is an important aspect for a tool, which may in future be extended. Debugging the tree is also comparatively easy, the complete structure can be logged in an understandable manner, which makes the localization of errors easier than for instance in neural networks. The decision tree is the core feature of the modeling tool, storing the gathered information from the user's actions and applying this information to ingredients of the same type. With its help the modeling process should be shortened, enabling to model complex microorganisms in comparatively short time. The decision tree is initially an empty node. After the first interpretable user action a subtree for that specific ingredient is created and appended to the root node of the decision tree. Depending on the model's compartment structure a hierarchy is created. Each node of the tree holds information about a certain ingredient type relative to the compartments in the scene. When a new action occurs the decision tree is modified by the new value, but only for those, which are inside the same compartment. Afterwards all affected ingredients, which are either the ingredients with the same id, or ingredients, which are in some kind of relationship to the changed one, are updated as well, receiving a new position and/or orientation. The decision tree is depicted in Figure 3.2.

As there can be seen in Figure 3.2 the decision tree takes the compartment hierarchy as a subtree, modifications of an ingredient inside a compartment does not affect the probabilities the same ingredient contained in another tree node (another compartment). When modeling microorganisms the depth of the decision tree is limited by the depth of the compartment hierarchy. The amount of compartments in a microorganism is typically relatively small, compared to the amount of ingredients the microorganism contains. Therefore also the depth of the decision tree is quite small, making the transition of the tree and the evaluation for a specific ingredient type and position fast. A tree node holds their respective child nodes, the amount of placed ingredients (inside the compartment or on the surface), the rotation probabilities and the compartment it belongs to. Each modification of the decision tree must be reversible in order to allow undo the last modification and go back to the initial state. The constraints are considered when ingredients of the referenced type are updated and a new position and/or orientation is requested.

3.5 The Problem of Overlapping Molecules

The process of modeling microorganisms, consisting of several compartments, filled with different molecules and proteins, can result in very crowded scenes. Without application of a collision detection several molecules are overlapping, creating slightly disruptive result. Therefore some kind of collision detection is needed to create more plausible results. Since there are so many molecules consisting of thousands of atoms a precise collision detection would result in a significant performance loss. Therefore an approximation is taken, which takes the largest radius of an ingredient. This approach works very well

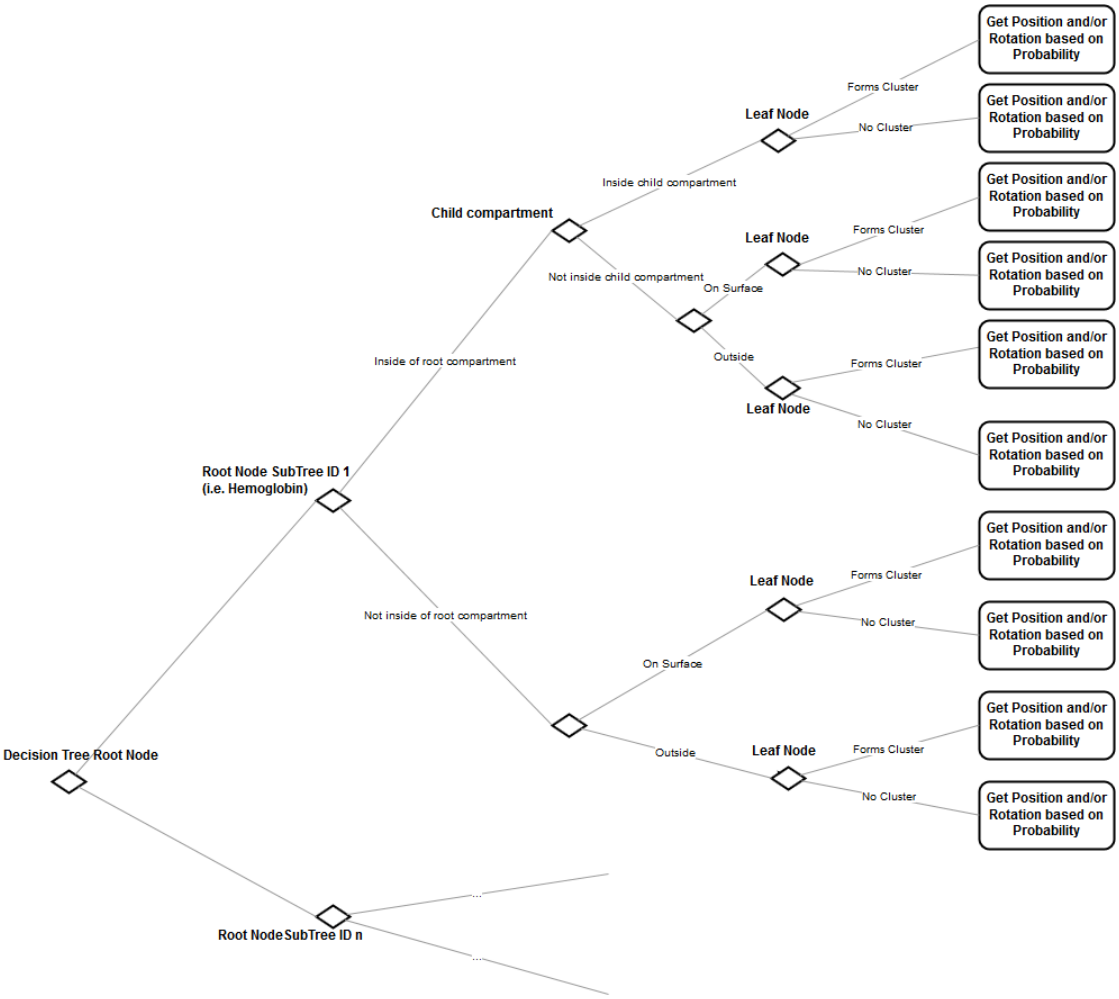

Figure 3.2: The decision tree as implemented in the tool. The tree grows with the more information it has, gathered through the user's actions. Each different ingredient type has its own subtree, which is added to the root node.

with spherical proteins, but with elongated substances the bounding sphere is too large, leading to large holes, as depicted in Figure 3.3.

The conflicting ingredients are assigned to their corresponding compartment, based on the probability. Then it is distinguished whether they are on the surface, inside or outside this specific compartment. Then they move into a random direction without crossing the border of other compartments until they find an empty spot with no intersections. A counter counts the amount of trials and errors, finally removing the substance if no empty space could have been found after a certain amount of relocations.
The implemented collision detection approach is not optimal, as stated previously the usage of bounding spheres for collision detection works only well for spherical shapes.

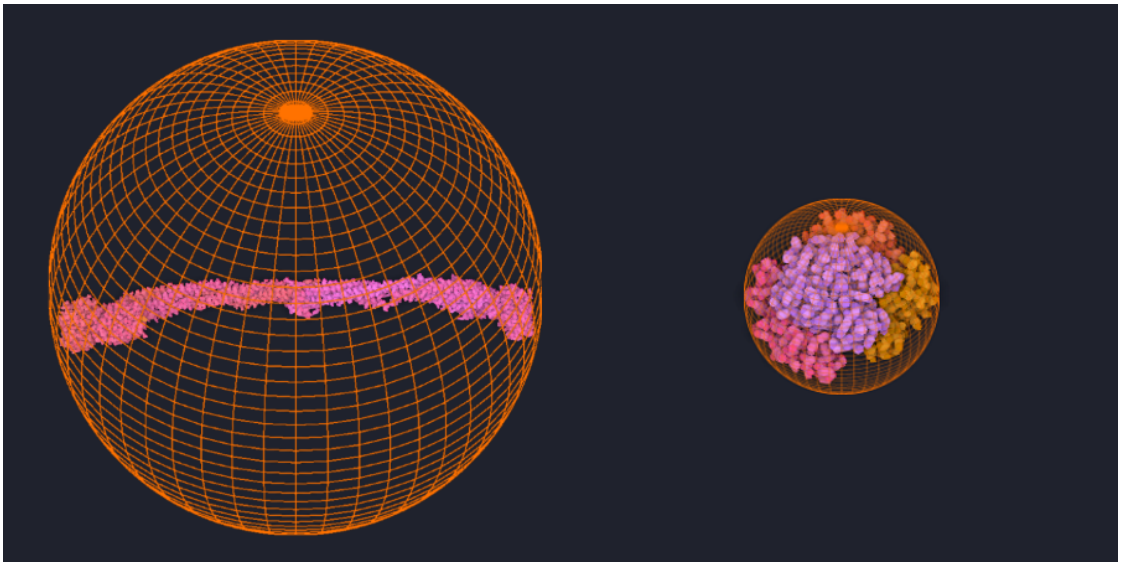

Figure 3.3: The currently used collision detection approach takes only the bounding sphere. For almost spherical objects this approach works very well, for non-spherical objects the collision detection may lead to holes in the final model.

The system is implemented in a modular manner, allowing the exchange of the current collision detection approach by a more precise one by simply exchanging the compute shader, which handles the collision detection itself, the resolution of the collisions is implemented for either approach. A major improvement would be, to use an axis aligned bounding box for the ingredients, which would only result in insignificantly small holes. The reason, why this simple approach was used in our tool, is that it was not the focus of our thesis to implement those algorithms, but rather come up with a functioning and modular system and to provide a reference implementation to test the system and to provide a proof of concept, the adaption to more robust approaches is ensured by the modular system.

3.6 Import and Export

After creating a model of a microorganism it might be interesting for the user to have more possibilities than only making screenshots of the model. Therefore we provide the functionality to import and export the current scene, including all components such as the macromolecules, the compartments, the clusters and the decision tree. Similar to the cellPACK [JAAA+17] software, which allows modeling microorganisms and produces a recipe for the model, which again can be loaded with a visualization tool, our modeling tool should provide the functionality of exporting an intermediate version of a microorganism in a way that the user can continue with the modeling process. Our tool makes use of several different data structures, such as the decision tree, which contains all probabilities,

the clusters, which can be formed and their composition. The compartment hierarchy should be restored, each ingredient should be reloaded at its very position without information loss. Currently we do not export the scene in a format, that an existing visualization tool can interpret, but with a specification of the file format it would be possible to visualize the microorganism also with other tools.

CHAPTER 4

Implementation

In this section the implementation of the features are discussed. Benefits and drawbacks are showed up and the difficulties, which had to be overcome are briefly discussed.

4.1 Compartments

The initial step in the creation of a microorganism is to create the basic structure. As first step the user has to add a root compartment, which usually is the cell membrane, formed by a certain lipid or macromolecule. The membrane is populated on the surface of the compartment, defined by its distance function. When creating a compartment the user has the possibility to deform these compartments and add inner compartments such as the capsid or organelles. An initially set cutting plane helps modeling also the interior of the cell. If the needed ingredient is not present the user has the possibility to download files anytime during runtime and make use of them. Compartments have their own collision-detection implementation, which is valid for any shape which would be added in the future as long as the distance function is correct. In the following the algorithms that calculate the signed distance of the ellipsoid 4.1 and the capsule 4.2 are described. In the first lines of both algorithms there can be seen, that the point gets rotated by the inverse of the current rotation around the center of the compartment to bring it to the correct position. For the capsule also the reference points a and b have to be transformed. The radii of both the ellipsoid and the capsule are manipulated during scale operations, translation does not change these values. This is necessary in order to support transformations of the original shape.

With the distance function and a bounding box intersections are calculated before the positions are updated, stopping the process if intersections are occurring. Therefore the inner compartments cannot be scaled or moved out of the membrane. This is supporting the modeling process by not allowing illogical situations. The inner representation of these compartments is only the signed distance function. When adding new compartments

Algorithm 4.1: Signed distance function for a rotated ellipsoid [Qui17]

Input: A point in space $\vec{point}$, the current rotation quaternion
Output: Float x, if $x == 0$ the point is on the surface, if $x < 0$ the point lies inside, if $x > 0$ the point lies outside

1 Vector3 $\vec{pos}$ = QuaternionTransform(Quaternion.Inverse(rotation), point);
2 **return** $(Vector3.Scale(\vec{pos}, 1/\vec{radii}).magnitude - 1) * \min \vec{radii}$;

Algorithm 4.2: Signed distance function for a rotated capsule [Qui17]

Input: A point in space $\vec{point}$, the current rotation quaternion
Output: Float x, if $x == 0$ the point is on the surface, if $x < 0$ the point lies inside, if $x > 0$ the point lies outside

1 Vector3 $\vec{pos}$ = QuaternionTransform(Quaternion. Inverse(rotation), $\vec{point}$);
2 Vector3 $\vec{capsA}$ = QuaternionTransform(Quaternion. Inverse(rotation), $\vec{capsuleA}$);
3 Vector3 $\vec{capsB}$ = QuaternionTransform(Quaternion. Inverse(rotation),$\vec{capsuleB}$);
4 Vector3 $\vec{pA}$ = $\vec{pos}$-$\vec{capsA}$, $\vec{bA}$ = $\vec{capsB}$-$\vec{capsA}$;
5 Float h = clamp(dot($\vec{pA}$, $\vec{bA}$)/ dot($\vec{bA}$, $\vec{bA}$), 0, 1);
6 Vector3 $\vec{v} = \vec{pA} - \vec{bA} * h$;
7 **return** $(Vector3.Scale(\vec{v}, 1/\vec{capsuleRadius}).magnitude - 1) * \min \vec{capsuleRadius}$;

to the tool, the only thing needed is a valid distance function, handling also possible rotations of the original shape. The developer has to handle the different transformations by applying rotations, scale and translation only for the compartment-specific member variables, such as the center point. For example the translation in the ellipsoid does only translate the center point. The scale modifies the radii of the ellipsoid, and the rotation is simply stored and used in the signed distance function to provide the correct output also for rotated compartments. Everything else is handled in the abstract compartment class, when a developer wants to add a new compartment shape, the new class has to extend the abstract compartment class. The methods that have to be implemented in order to get a working compartment shape are the following:

- The method GetName should provide an unique name.
- The method GetPossibleChildOrigin should provide a valid position inside the current compartment for child compartments, which is not trivial for shapes such as the torus.
- The method HalfSize should halve the size of the shape based on the size of its parent, so the compartment does fit inside its parent without intersections.
- The method GetSignedDistance is the signed distance function, which handles also the rotation of the compartment.

- Lastly the method AdaptCompartmentToTransform has to adapt the member variables, such as the radii or the dimensions, based on the transform type.

4.2 Real-Time Configuration

cellView loads a recipe containing all macromolecules, each of these ingredients is assigned to a specific group or compartment and has a certain color. When a user wants to model a microorganism there are initially no assumptions about the ingredient group they belong to, in fact, if the user downloads a protein from PDB database the tool knows nothing about this new protein. For already existing ones we have a common name and a description, which tells the viewer many things about the macromolecule itself, but also about the composition of the microorganism. For downloaded files we only have a four-digit alphanumeric identifier of the protein. To enable this benefit also for the modeling tool we created a JSON file, which stores this four-digit identifier of the ingredient, a common name, a description and a color, for each ingredient in the scene. A new entry is created automatically when downloading a file from PDB database. Now the user has the possibility to change this document with any text editor, also during runtime. With that, a user is able to create the desired visual representation of the structure, by communicating certain information via the color (such as blood proteins, which are usually assumed to be red, or viral proteins which are recognizable by an unhealthy looking color, such as green). Moreover the creator is able to write a description that is tailored to the modeled microorganism, for example a certain protein has a specific purpose, but in an infected cell this purpose is, due to the infecting substances, destroyed. This is implemented by including this functionality into the already existing info-text controller. The user has a “Refresh Configuration” button, which parses the JSON file and replaces the currently stored values for the color, the common name and the description by the fields in the file. The usage of a JSON file to edit this information is not very user-friendly, mainly because this file is not editable in the software itself and because JSON files have a strict syntax, but errors are not shown with general text editors. The idea behind it, was to show up the possibility, it could basically also be included in the tool itself, but an intuitive graphical user interface would have to be implemented.

4.3 Import and Export

The importing and exporting process happens through serialization and deserialization of all needed scene components with JSON. In large scenes, with many thousands of proteins, a complex decision tree, many compartments and clusters the file size may get quite big. When exporting the whole information is written in a cfv file (cellView-file). These files can be imported on any other computer without losing any information, the whole structure is restored and the modeling process may be refined or also changed, a limitation of this process is, that already made decisions are hard to invalidate. If incorrect decisions have been made earlier the resulting image may always show influences

Table 4.1: Comparison of the import and export of models with increasing amount of scene objects.

File size (scene objects)	Desktop with Intel i5-3570K 3,4 GHz and NVIDIA GeForce GTX 750		Laptop with Intel i5-6200U 2,4 GHz and NVIDIA GeForce 940M	
	Import	Export	Import	Export
7,95 MB (5.000)	3,416 sec	4,009 sec	4,656 sec	5,078 sec
15,6 MB (10.000)	6,458 sec	6,552 sec	9,250	8,375 sec
156 MB (100.000)	64,553 sec	65,692 sec	92,121	83,850 sec

of this mistake, for example some ingredients may be found at a point where they should not be. Therefore a redo and undo functionality has been implemented, allowing the user to undo or redo up to three actions, also removing modifications of the decision tree. This enables the user to try a certain step, if the outcome is not satisfactory or decreases the quality of the model, it is possible to restore the previous state without saving the scene after each step. More steps would be possible, but the storage of many memory intense actions are very demanding and may lead to errors when restoring the previous state.

In Table 4.1 there can be seen that the duration for the import and export as well as the file size increase linearly by the amount of objects in the scene, this is due to fact that no optimization steps have been applied, the file size as well as the duration could be decreased significantly by storing the necessary values more efficiently.

CHAPTER 5

Results

After explaining the concepts and the implementation of some core features of our tool, in this chapter some visual examples are provided.

5.1 Compartments

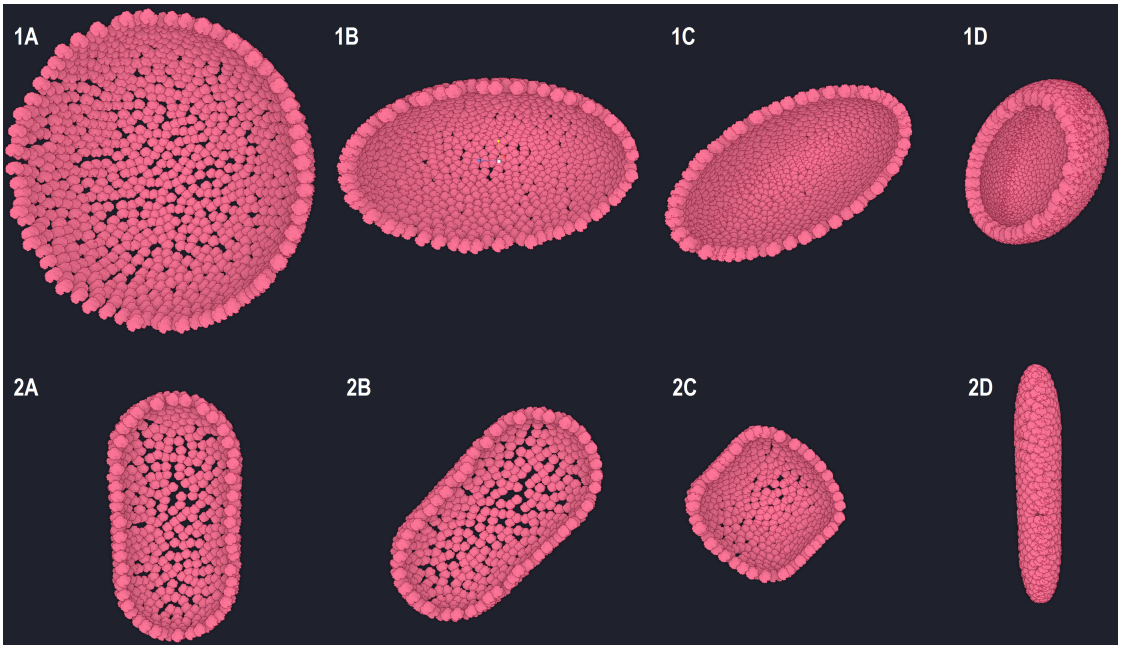

Figure 5.1: Different possible compartment shapes, created through scaling and rotating the primitive shape. 1A-1D are shapes, created from the ellipsoid, 2A-2D are created from the capsule.

As mentioned in previous sections we implemented the two compartment shapes ellipsoid and capsule. These shapes are freely rotatable and scalable, so many different shapes are already possible to create, as shown in Figure 5.1. These shapes would already be sufficient to create a prokaryotic- and a eukaryotic cell, in the future more different shapes or free-form surfaces could be implemented, to allow the modeling of more complex microorganisms too.

5.2 Real-Time Configuration

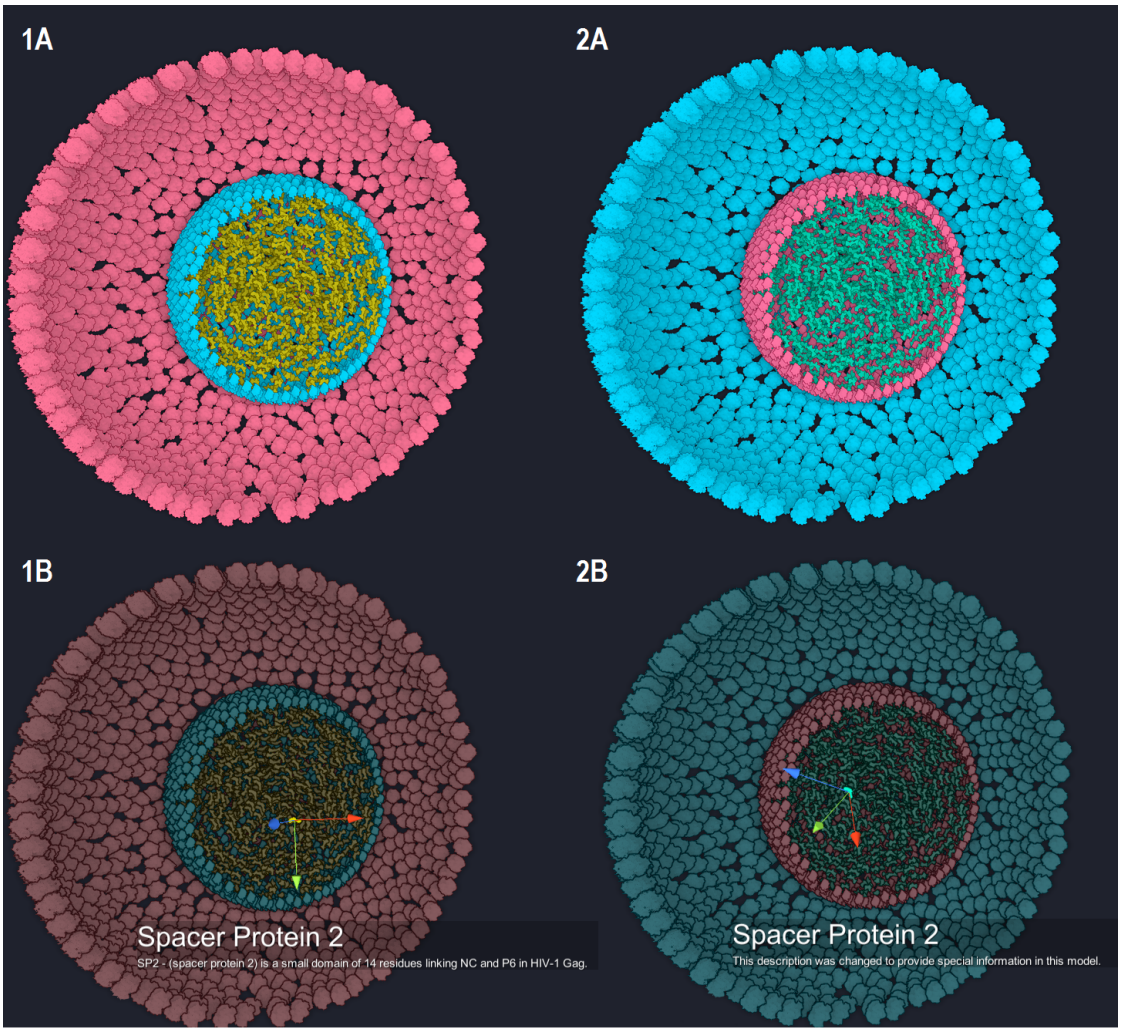

Figure 5.2: 1A and 1B show the initial state when modeling a cell, in 1B the last description is set, but if this description would not clarify the purpose of this ingredient in this certain model in 2A the color values have been exchanged to provide a better visual representation and in 2B the description of this ingredient has been changed.

When modeling a microorganism certain macromolecules are used, which may serve a specific purpose in this microorganism. It is assumed, that it may be desired to modify the description of the used macromolecules. In addition to create a better understandable visual representation it may be desired to use a specific color for some ingredient, either to highlight it, or to pack more information in the final outcome, as for example a toxic substance could be colored in green or for blood proteins the most intuitive color would be red. These values can be modified during runtime by editing the configuration file, as depicted in Figure 5.2. Currently this feature is only usable when editing the configuration file, which is yet not possible within the tool. A user-interface, allowing the modification in the tool could be a future work.

5.3 Collision-Detection

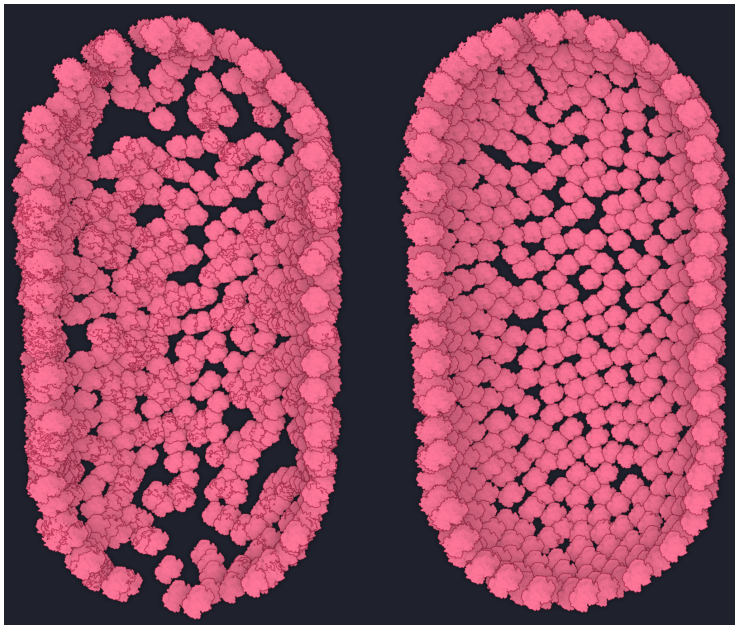

Figure 5.3: This image shows the differences between a compartment, created without collision detection (left) and the exactly same compartment with collision detection (right).

Initially no collision detection has been implemented in our tool. The resulting images had many overlapping molecules, which led to unsatisfactory results. After implementing the general modeling tool a simple collision detection approach has been implemented. Although the approach is very simple and only suitable for spherical molecules, the resulting images were more plausible. A more precise approach may be implemented in the future. In Figure 5.3 the differences between the initially created compartment without collision detection and the exactly same compartment with applied collision detection are shown.

CHAPTER 6

Reflecting

The final outcome of the tool we created was better than expected. With the available compartment shapes it is already possible to produce good results. Also the decision tree, which is yet only able to detect few different user actions, is strongly supporting the creation process. Depending on the structure a user wants to model one may encounter different problems, because some actions may not be taken into account. It is yet not simply possible to model certain patterns, such as a certain surface structure of the compartment hull (for example the capsid). There are also other limitations, such as the placement of components in some subsections of the cell, which do not form a compartment. To be more precise, the current implementation is strongly focused on the compartments and the through compartments formed hierarchy. The calculations for the positions and orientations of the components, as well as the adaption of the decision tree and the resolution of the collisions is implemented on the CPU, only the detection of the collisions is implemented on the GPU. These calculations are possible in real-time and could still be optimized and extended to handle even more complex user actions without significant performance loss. Heavy calculations, such as the collisions, or to be more precise the resolution of the collisions lead to slight performance issues, especially when there are several thousands of colliding ingredients. This problem could be avoided by resolving the collisions on the GPU. Another limitation of the tool is the correct placement of the ingredients. The algorithm takes the final position of the component as a reference and adapts the decision tree according to the placement. If it is not correct, the adaption made to the decision tree may lead to undesired errors which are hard to readjust. Therefore we implemented the possibility to undo the last three actions, which removes these previous modifications from the tree and also the placement and orientation is restored. Although this eases the problem of the placement, it is still impractical to place an ingredient at a desired position. The movement in three dimensions makes it hard to see where exactly the ingredient is in correspondence to other compartments and also other ingredients.

The previously shown screenshots are entirely created with the tool in short time periods. The focus of this experiment was to create a tool, which reacts to the users input in an appropriate way. This task was fulfilled and the modeling process is much faster in comparision to the manual placement of each molecule. The movements of the ingredients and the overall behavior are much more interactive. By providing the possibility to undo recent actions mistakes can be rectified, making the modeling process almost entertaining. The tool was entirely created with Unity3D [Tec17] and C#. In the beginning it was quite hard to work with Unity, because some actions seemed to be arbitrarily, in the end the tool was built on top of cellView without harming the previous functionality.
For statistical analysis of the user's actions we make only basic assumptions about the placement and the orientation, but these assumptions could in the future be extended to understand also more complex actions. An idea would be to not only check the placement relative to a given compartment, but also to check the distance to the compartment's border (for example the membrane). We took a closer look at the HIV virion, which is loaded through a cellPACK [JAAA$^+$17] recipe, and collected the information about its composition. This HIV virion was modeled by hand by placing each component at its very position. With the gathered information we tried to reproduce this structure with our tool, the resulting image, compared to the original virus is depicted in Figure 6.1. The resulting image is close to the original, containing the same amount of components. To create this model it took us about 15 minutes of time, making it really fast compared to the manual placement of about 20.000 components. With further improvement steps and a better analyzation of the user input, the needed time might even be reduced and the resulting structure may be even better.

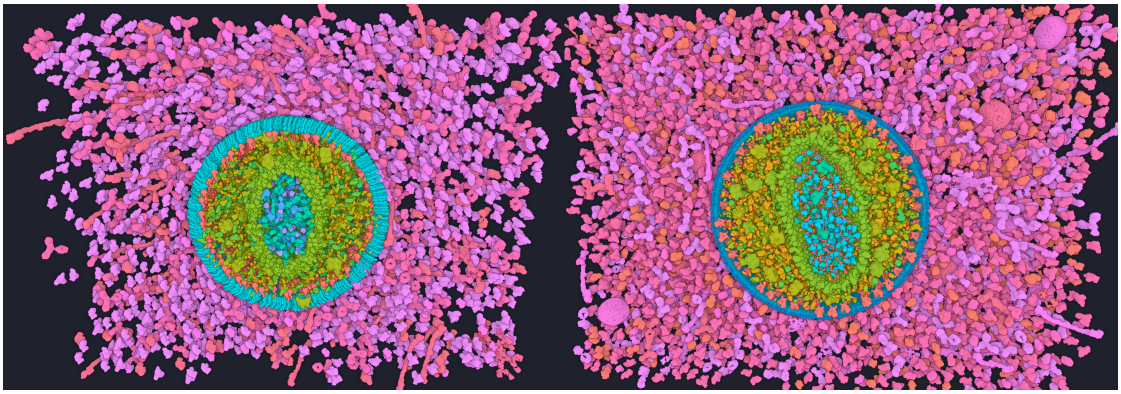

Figure 6.1: HIV viria, the HIV virion on the left side is modeled with the tool, the right one is the original model of the HIV virion, loaded with cellView. The models have the same composition.

CHAPTER 7

Conclusion and Future Work

An approach for statistical modeling has great potential for the future, the composition of microorganisms follows certain patterns, such as the occurrence of certain proteins in a cellular compartment. Although an initial classification may be inaccurate, a learning algorithm is able to predict possible positions with increasing precision. The development of a tool, that does not take a rule based approach, but a statistical approach has great potential to become very powerful. In statistics many different distributions exist, in the development process we only used a Gaussian distribution, but with additional distributions more possiblities in the creation process may show up. The possibilities to extend our tool are given. Future improvements would be an implementation of a more precise collision detection, to represent such crowded scenes with no overlapping molecules, which improves the perception of the final image. Another possibility would be to create templates, which could be later used for various different microorganisms. The task of implementing possible cluster formations was cumbersome, because it leads to many new tasks, on how to handle such clusters. But basically they could be also represented by a certain template, by the means of a combination of proteins, which are very likely to be found in a certain constellation. Another application would be to use such templates for compartment boundaries, such as the capsid of the HIV virion, which has a special star-shaped pattern that is hard to learn, but would be easy to be modeled as a template. To further improve the modeling the addition of more different compartment shapes would lead to more possibilities in order to create cells and compartments of any shape. The tool provides interfaces where new compartments could easily be added with only implementing the necessary methods and modifying the parameters, such as the radii, based on the transformations. Currently we take advantage of the signed distance functions. In order to support freely formable compartment shapes adaptions of the distance functions would have to be defined, or could be described by combining different primitive shapes and their respective distance functions.
Regarding the import and export in general it would not be necessary to parse the

data with JSON, it is a convenient way and is easy to implement, but for example the export in XML format would have been possible as well. The size of the output file is, depending on the amount of ingredients, comparatively big, making both the import and the export protracted. A very simple optimization step would be, to overwrite the JSON properties of certain objects, such as the vector. Currently the vector holds the normalized position, the magnitude and the squared magnitude, which results in much larger file sizes for information, which is not directly needed. Other data, such as the compartment hierarchy and the decision tree, occupy only a very small portion of the file, compared to the proteins.
The research area of molecular visualization is a very interesting field, with many possibilities for improvements and extensions. This tool would provide enough work for many more people in order to create a reliable and powerful tool in the process of modeling complex microbiological structures in a very short amount of time. The learning process could be extended to recognize many more patterns and the science would clearly appreciate such a tool, not only for visualization purposes, but also for students to make microbiology more comprehensible and almost tangible.

Glossary

compartment A compartment is a subsection of the cell, which is usually surrounded by a single or double lipid layer membrane. In the tool the compartments form a hierarchy, the membrane, the capsid and other subsections are represented through a compartment. 5

ingredient Ingredients in this context refer to any molecular structure, which can be added in the modeling process. For the tool mainly proteins have been used.

Bibliography

[BŠMM11] Bedrich Beneš, Ondrej Šťava, R Měch, and Gavin Miller. Guided procedural modeling. In *Computer graphics forum*, volume 30, pages 325–334. Wiley Online Library, 2011.

[BW90] Jules Bloomenthal and Brian Wyvill. Interactive techniques for implicit modeling. *ACM SIGGRAPH Computer Graphics*, 24(2):109–116, 1990.

[JAAA+17] G. Johnson, L. Autin, M. Al-Alusi, D. Goodsell, M. Sanner, and A. Olson. cellpack, March 2017.

[Med17] Mediran. Eukaryotic cell (animal). `https://creativecommons.org/licenses/by-sa/3.0/deed.en`, February 2017.

[MST94] Donald Michie, David J Spiegelhalter, and Charles C Taylor. Machine learning, neural and statistical classification. 1994.

[PJR+14] Julius Parulek, Daniel Jönsson, Timo Ropinski, Stefan Bruckner, Anders Ynnerman, and Ivan Viola. Continuous levels-of-detail and visual abstraction for seamless molecular visualization. *Computer Graphics forum*, 33:276–287, 2014.

[PL12] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The algorithmic beauty of plants*. Springer Science & Business Media, 2012.

[PV] Julius Parulek and Ivan Viola. Implicit representation of molecular surfaces. Discussion paper.

[Qui17] Inigo Quilez. Modeling with distance functions, February 2017.

[RU17] Rutgers and UCSD/SDSC. The protein data bank, February 2017.

[SBV98] Magne Setnes, Robert Babuska, and Henk B Verbruggen. Rule-based modeling: Precision and transparency. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 28(1):165–169, 1998.

[Tec17] Unity Technologies. Unity3d engine, February 2017.

[Wik17] Wikipedia. Average prokaryote cell, February 2017.