FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Visualizing Protein Interactions in Corresponding Compartments

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Susanne Rinortner
Matrikelnummer 1526772

an der Fakultät für Informatik
der Technischen Universität Wien
Betreuung: Projektass. Hsiang-Yun Wu, PhD

Wien, 10. September 2019

Susanne Rinortner	Hsiang-Yun Wu

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Visualizing Protein in Corresponding Compartments

Hybrid Biological Images

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Susanne Rinortner
Registration Number 1526772

to the Faculty of Informatics

at the TU Wien

Advisor: Projektass. Hsiang-Yun Wu, PhD

Vienna, 10th September, 2019

Susanne Rinortner	Hsiang-Yun Wu

Erklärung zur Verfassung der Arbeit

Susanne Rinortner

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, 10. September 2019

Susanne Rinortner

Kurzfassung

Die Visualisierung von Protein-Protein Interaktionen ist ein wichtiger Schritt, um diese zu verstehen. Es gibt bereits viele Möglichkeiten, die dieses Problem lösen. Diese lassen aber meistens die Position der Proteine im Inneren der Zelle außer Acht. Jedoch ist dies eine wichtige Information, die in einer Visualisierung nicht verloren gehen soll. Diese Arbeit stellt einen neuen Ansatz vor,um die Position von Proteinen zu visualisieren. Das vorgestellte Programm nutzt ein dreidimensionales Modell einer Zelle als Basis. Dieses wird geschnitten, um eine zweidimensionale Oberfläche des Durchschnitts zu erzeugen. Die Oberfläche wird abgetastet und rekonstruiert um einen Algorithmus anzuwenden, welcher mit physikalischen Kräften einen Graphen erzeugt. Dieser Algorithmus wird auch dazu verwendet einzelne Zellteile zu skalieren um zu gewährleisten, dass alle innenliegenden Proteine darin Platz finden. Auf diese Weise wird sichergestellt, dass kein Teil der Zelle überfüllt wird. Mit dieser Methode ist es möglich nicht nur Protein-Protein Interaktion sondern auch die Position der Protein in der Zelle zu visualisieren.

Abstract

Background: The visualization of networks for protein interactions is an important step to understand them. There are already many approaches for this task, but most of them do not show any information about the compartment of the cell the proteins belong to. **Motivation:** Since the placement of proteins inside a cell is important information, because it helps to understand their interactions, this thesis proposes a method to visualize protein inside cell compartments. **Goal:** The objective of this project is a clear and understandable visualization of interactions between proteins and where these interactions or reactions happen inside the cell. **Method:** This project uses a three-dimensional model of a cell as a base and intersects it using cutting planes. Then the intersection surface is sampled and reconstructed using Delaunay triangulation. To the mesh created by the triangulation, a force-directed algorithm is applied. This algorithm is used to scale single cell parts in order to fit all proteins inside. This ensures that none of the cell parts gets overfilled. **Result:** The result is a new method which makes it possible to visualize not only protein-protein interactions but also in which compartment of the cell the proteins are located.

Contents

CHAPTER 1

Introduction

The following chapter will give an introduction to the thesis. It will provide information about the background and motivation. Furthermore the objective of the thesis will be explained. There will be information about the approach as well as the result. The last subsection describes how this thesis is structured.

1.1 Background

Visualizing networks for protein interactions is an important step in understanding them better. Because tabular data can be hard to read, especially when it comes to information about relationships, visualizing this date gives an easier insight. There are many different approaches for visualizing biological interaction data, ranging from graphs to matrices and heat maps. Most tools using a graph for displaying interaction data are focusing on showing the graph with the best and clearest layout possible and ignore the location of the proteins inside the cell completely to reach this goal.

1.2 Motivation

There are many approaches and tools for displaying protein interaction data that lack information about in which compartment of cell a protein is located. Because of that, there is a demand for a tool that displays interaction data in a clear way as well as showing where the interaction is happening within the cell. For example, locations of proteins inside the cell can help to understand patterns of relations or how cell compartments are related to different activities [Heberle et al., 2017]. For this reason it is important to develop a tool that gives the possibility to show in which compartments the interactions happen since this is valuable information that gets lost otherwise.

1.3 Objective

The objective of the thesis is to visualize which compartments of a cell a protein belongs to. The most important task is to make this information clearly visible. For this reason it has to be possible to scale cell compartments. The scaling is necessary because it is possible that there is not enough room for all proteins in a compartment. Since graphs are the most common approach for visualizing protein interactions, the interactions shall be represented by a graph. The reason for this is, that it will be easy to understand for biologists who are already used to this visualization type. In the resulting graph the nodes represent proteins and the edges are the interactions (in the following referred to as protein graph). This graph should be placed on an intersection of a cell (in the following referred to as intersection mesh) to show the location of the interactions. To better visualize the proteins inside the cell it is important that the different compartments of the cell are scalable individually. If there are too many nodes in one compartment of the cell to display them clearly it is necessary to scale that compartment, so all vertices that belong there can be displayed. This provides a technical challenge because the scaling of the compartments has to deform the compartment evenly. To reach this goal it is important to keep the number of edge crossing in the mesh representing the intersection surface as low as possible. The main objective of this thesis is to visualize proteins and interactions between proteins. But the same approach can be applied to other chemical compounds as well. This is possible by making the nodes symbolize genes, enzymes or other compounds instead of proteins.

1.4 Result

The result of this project is a prototypical version of a tool that not only shows the relationships between proteins but also their location. The input is a a three-dimensional model of a cell that is cut in half using cutting planes, then sampled and reconstructed using Delaunay triangulation. This results in a two-dimensional intersection of the cell. To that intersection surface a force-directed algorithm is applied that enables the scaling of single cell parts. The output is a visualization that displays a protein graph on top of a cell intersection and shows in which compartments the proteins are located. There is a high possibility of more proteins being located in a cell part than would fit in the original size of it. For that reason it is important to have the possibility to scale cell compartments, to ensure that all proteins can be placed in their assigned cell compartment.

1.5 Structure of the Thesis

This thesis starts with this introduction chapter 1, afterward in the chapter 2 “Related Work” important information about protein-protein interactions and visualization methods for those will be introduced. Existing approaches for the problem of mapping the protein graph onto an intersection of a cell will be discussed. In the "Methodology" chapter 3 the data set will be introduced and the theory of cutting planes, triangulation

and force-directed graph generation will be explained. The "Implementation" chapter 4 contains information about the importing, displaying and cutting of the cell with OpenGL and ASSIMP. Afterward the generation of sample points and the triangulation with CGAL will be explained. Furthermore the scaling of the cell parts with the force-directed graph algorithm as well as the positioning of the proteins onto the intersection mesh will be described. The thesis finishes with a description of the result in chapter 5 as well as a conclusion and a discussion about possible future work directions in chapter 6.

CHAPTER 2

Related Work

There are many ways for visualizing protein interactions and other biological pathways. The most common approach is to represent the relationship data as a network. Other approaches include matrices, heat maps and a nodetrix. The following chapter gives information about state-of-the-art visualization tools for protein interaction data.

2.1 Visualization Methods for Protein Interactions

Protein interactions are part of biological pathways that can be represented as a network. In this case, the nodes of a graph represent single proteins that interact with each other. These interactions are represented by edges connecting the proteins. These graph visualizations exist in two dimensions, as in the aforementioned examples, as well as in three dimensions.

2.1.1 Two-Dimensional Graph Visualization

Popular tools for this network visualization of biological relationship data include "Cytoscape" ,which is an open-source software for visualizing biomolecular interaction networks [Mario Cannataro, 2011] and "VisANT" where the back end supports data retrieval as well as visualization [Mario Cannataro, 2011]. Another tool is “Medusa” a open-source Java application that is optimized for protein interaction data. It supports weighted graphs and works best for smaller networks [Mario Cannataro, 2011] . “ProViz” is another tool that uses a two-dimensional graph visualization. It’s a scalable open source application that makes extensive use of plugins [Mario Cannataro, 2011]. Another application that visualizes in two dimensions is “Pivot” which is used to visualize protein interactions [Mario Cannataro, 2011]. “Pajek” which supports directed and weighted graphs falls into the category of two-dimensional graph visualizations as well [Mario Cannataro, 2011].

2.1.2 Three-Dimensional and Pseudo-Three-Dimensional Graph Visualization

The first two tools that fall in that category are not using real three-dimensional visualization but pseudo-three-dimensional which means that the visualization looks like it is three-dimensional but in reality it is not. The tools using this form of visualization are "ProViz" [Mario Cannataro, 2011] and "Pajek" [Mario Cannataro, 2011]. An approach that uses three dimensions is "Jak-Stat" [Ganesan et al., 2016] which is a framework based on modeling biochemical reactions in three-dimensional space. For this it exploits the parallelism of the GPU [Ganesan et al., 2016]. The second tool that uses real three-dimensional visualization is "BioLayout Express". Compared to the aforementioned "Jak-Stat" it provides the option to switch between a two- and three-dimensional view [Mario Cannataro, 2011].

2.1.3 Other Approaches

Another possible way to visualize protein interaction data is by using adjacency matrices or heat maps. Tools using adjacency matrices like "MatLink" reduce the data to binary interactions between proteins and therefore making them easier to understand and analyze [Fekete, 2009]. “PathwayMatrix” is a tool that uses a matrix exclusively to display a network. It tackles the challenge of visualizing and analyzing complex pathways by visualizing binary relations between proteins with the use of an interactive adjacency matrix [Dang et al., 2015]. A special way of visualization is "NodeTrix" which uses a nodetrix and is therefore a mix between matrix- and graph visualization [Fekete, 2009].

All of these tools do not reference the position of the proteins inside of the cell for the visualization of the data which makes the approach presented in this thesis necessary.

2.2 Existing Approaches for Visualizing Compartments

There are already some approaches that take into account which compartment of a cell a protein belongs to, when visualizing protein interactions in a graph. Those approaches will be explained in the following. Compared to the approach presented in this thesis, those state-of-the-art tools do not deform cell compartments for better visibility of proteins.

2.2.1 CellNetViz

CellNetViz [Heberle et al., 2017] uses a force-directed layout to place networks into cellular compartments. It shows where network elements are located and concentrated and therefore visually organizes networks by cellular compartments. CellNetViz is written in Javascript and HTML making it a web-based approach [Heberle et al., 2017].

2.2.2 Mosaic

Mosaic [Salomonis et al., 2012] is a Cytoscape plugin that supports interactive network annotation, partitioning, layout and coloring. It uses a cell template that defines graphical regions that represent cellular compartments to create a cell-based layout. Nodes are positioned into cellular compartments based on annotations. After the positioning of the nodes a force-directed layout is applied within each region [Salomonis et al., 2012].

2.3 Contribution of this Thesis

1. State of the art approaches use a two-dimensional cell model as a base for their visualization. This thesis introduces an approach that uses a three-dimensional cell model as a base.

2. While most other approaches are web-based, this project is written as a desktop application in C++ and OpenGL.

3. The main contribution of this thesis is a method to scale cell parts by using a force-directed algorithm on the intersection of the cell. This method gives the option to enlarge single cell compartments to ensure that all belonging proteins can be displayed.

CHAPTER 3

Methodology

The following chapter gives an overview on all techniques used for this approach. It provides definitions and theoretical explanations of all techniques.

3.1 Data-Set

The data set used for the prototype consists of an three-dimensional cell object and a protein interaction graph. The following section gives details on those data sets.

3.1.1 Cell Model

The three-dimensional cell object consists of four individual spheres in different colors that represent the compartments of the cell. Those compartments are a core (or inner layer), a middle layer, the mitochondria and an outer layer. As shown in figure 3.1 the outer layer encases the mitochondria, the middle layer and the core. The middle layer encases the core. The center of each sphere is in the origin (0.0, 0.0, 0.0) except for the mitochondria which is positioned on the left side above the middle layer inside the outer layer. Each sphere has 32 segments and 16 rings which means it consists of 1984 vertices. Since the contour of each cell part is used for the intersection mesh generation it is important to keep the number of segments and rings low for better performance.

3.1.2 Protein Graph

The protein graph has 20 nodes that are connected by 20 edges. Each node represents a protein in a specified compartment of the cell and each edge represents a connection between two proteins. The positions of the nodes are two dimensional and each node and edge is rendered as a two dimensional quad on top of the intersection surface. These nodes have different colors, as shown in figure 3.2, symbolizing the cell compartment the vertex belongs to.

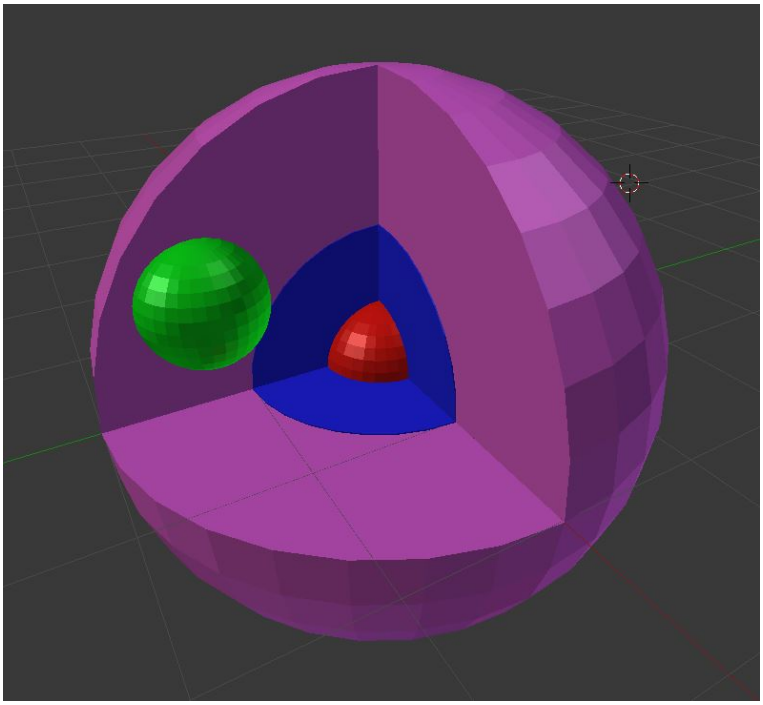


Figure 3.1: The 3D cell object.

The protein graph is a simple graph. Which means it is defined as a tuple $G = (V, E)$ that consists of a set of vertices $V = \{v_1, v_2, ..., v_n\}$ representing entities (in this case proteins) and a set of edges $E = \{e_1, e_2, ..., e_m\} \subseteq V \times V$ representing mutual connectivity [Wu et al., 2019] (in this case interactions).

3.2 Cutting Plane

Cutting planes are a method to cut objects. Additional to the object polygon mesh a plane mesh is needed. Generally, the object is cut into two parts per plane used. The plane has to be placed in a way that it intersects the object that will be cut, as shown in figure 3.3 on picture b. The first step is to sort the vertices of the object depending on which side of the cutting plane they are located. After that there are two options, either keep all vertices and generate two new meshes that represent the two parts of the object or remove the vertices from one side completely and keep only half of the original mesh.

3.3 Sampling and Triangulation

After the cell is cut using cutting planes the colors of the screen are sampled to create a two-dimensional reconstruction of the intersection. The sample points are generated

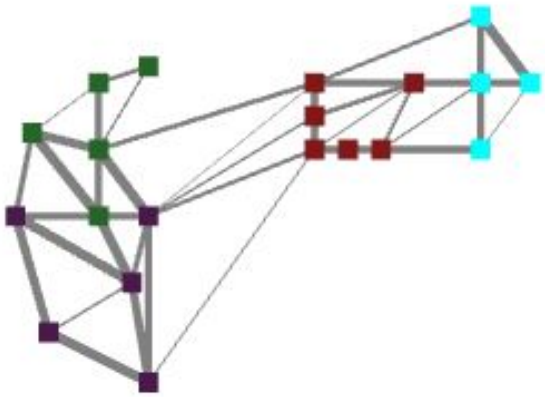


Figure 3.2: The protein graph with color coded nodes depending on the cell compartment they belong to.

using the Halton sequence and their coordinates are used as input for the Delaunay triangulation.

3.3.1 Halton Sequence

After the cutting of the cell, the colors on the intersection surface are sampled. For the generation of these sample points the Halton sequence is used. The Halton sequence is a generalization of the van der Corput sequence, where the idea is to express every value $n \in \mathbb{N}$ in a base b. Then the expanison is reflected into the unit interval $\mathbb{I} = [\,0,1\,] \subset \mathbb{R}$. In comparison, the main idea for the Halton sequence is to generate d one-dimensional sequences and form the corresponding d-dimensional vector sample points [Hokayem et al., 2003] . The defintion of the Halton sequence is based on the radical inverse function

$$\phi_p(n) \equiv \frac{b_0}{p} + \frac{b_1}{p^2} + ... + \frac{b_m}{p^{m+1}},$$

where p is a prime number, the p-ary expansion of n is given as $n = b_0 + b_1 p + ... + b_m p^m$ with integers $0 \leq b_j \leq p$. Compared to other low-discrepancy sequences, the Halton sequence is easy to implement because the implementation of the radical inverse function is relatively simple [Chi et al., 2005].

3.3.2 Delaunay Triangulation

Triangulation is a method for creating a mesh out of a set of points or polygons. The Delaunay triangulation of a point set S is characterized by the empty circumdisk property (explained in the definition below). This means that there is no point in S within any of the triangles created.

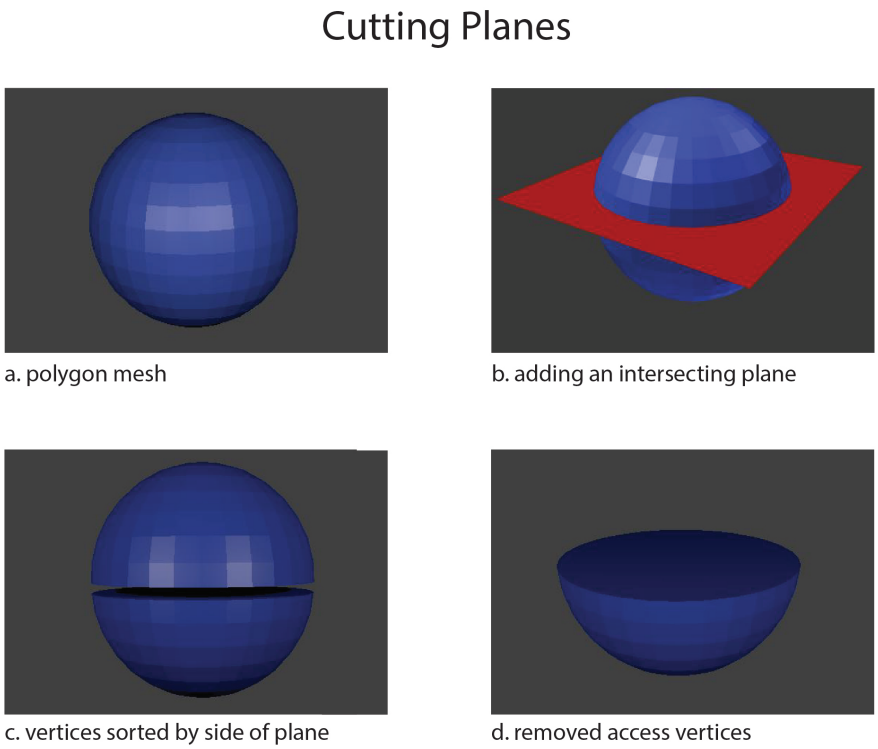


Figure 3.3: Steps of cutting a mesh using a cutting plane.

Definition: *"In the context of a finite point set S, a triangle is Delaunay if its vertices are in S and its open circhumdisk is empty - i.e. contains no point in S. Note that any number of points in S can lie on a Delaunay triangle's circumcircle. An edge is Delaunay if its vertices are in S and it has at least one empty open circumdisk. A Delaunay triangulation of S, denoted $DelS$, is a triangulation of S in which every triangle is Delaunay."*[Cheng et al., 2012, p.32]

Every point set has a Delaunay triangulation. In two dimensions the Delaunay triangulation has the advantage that it maximizes the minimum angle and also optimizes several other geometric criteria.[Cheng et al., 2012].

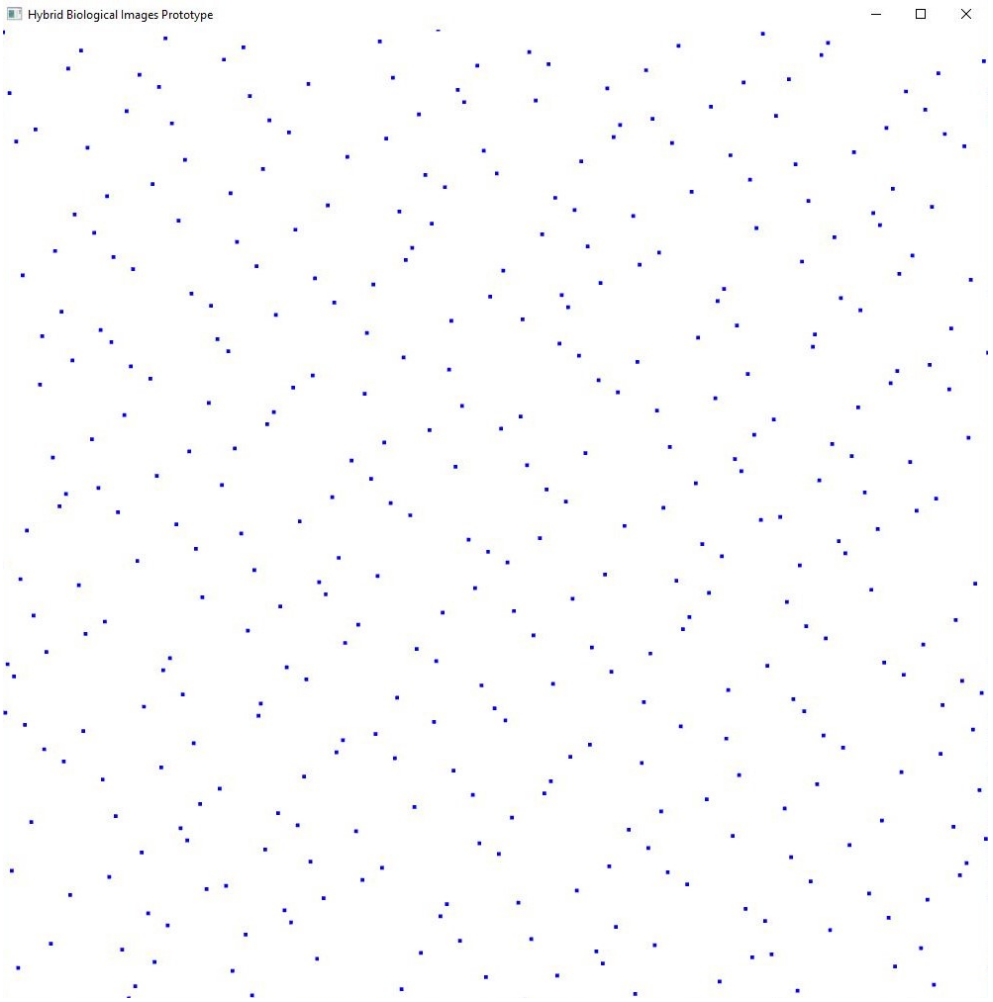


Figure 3.4: Sample Points with x-axis Halton sequence base 2, y-axis Halton sequence base 3

3.4 Force Directed Graph Generation

3.4.1 Graph Layouts Based on Physical Analogies

For the scaling of single compartments of the intersection graph a force-directed layout algorithm was used. While many other graph layout algorithms are based on structural characteristics of the graph, this approach sees the vertices of the graph as physical objects and uses their characteristics for layout generation. The advantages of methods based on physical analogies like this one are that they are very intuitive, easy to understand and that they generate fairly good results, especially on medium-sized graphs up to 50 vertices. A force-directed graph generation has two main components.The model of the graph, consisting of physical objects that represent vertices and edges and an algorithm

that computes an equilibrium configuration of said graph [Hutchison and Mitchell, 1973].

3.4.2 Basic Spring Embedder

The algorithm of Eades uses a mechanical model to produce two-dimensional layouts. This model replaces the vertices of a graph with steel rings and the edges with springs so they form a mechanical system. The graph has a random initial layout and when the springs are let go their forces on the rings move the system to a minimal energy state [Eades, 1984].

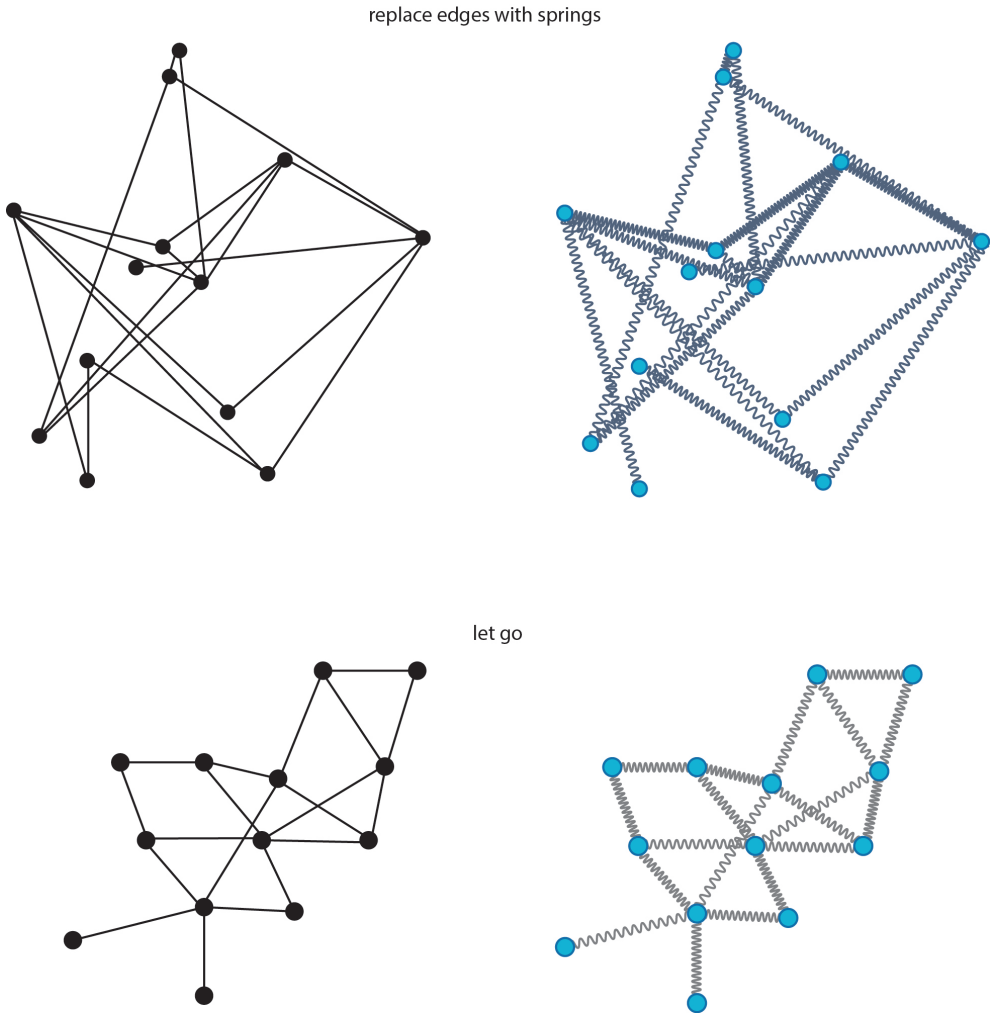


Figure 3.5: The input and output of the basic spring embedder.

In this spring embedder, the repelling forces used between every pair of non-adjacent vertices are defined as

$$f_{rep}(p_u, p_v) = \frac{c_\varrho}{||p_v - p_u||^2} \cdot \overrightarrow{p_u p_v}$$

,

where c_ϱ is the repulsion constant.

The spring forces between adjacent vertices are defined as

$$f_{spring}(p_u, p_v) = c_\sigma \cdot \log \frac{||p_u - p_v||}{l} \cdot \overrightarrow{p_v p_u},$$

where l is the natural length of the spring, the c_σ is a constant defining the strength of the spring. The direction of the vertex displacement depends on whether the actual distance between two vertices is greater or smaller than l [Hutchison and Mitchell, 1973].

3.5 Protein Graph Placement

For the placement of the protein graph, a simple algorithm is used. The graph has an initial layout where vertices that are assigned the same cell compartment are already clustered. For each compartment boundaries are calculated. For this the program searches for the vertices inside this compartment that have the highest and lowest x-coordinate and the highest and lowest y-coordinate. Then the boundaries are set by using the highest and lowest value on the y- and x-axis. This gives a rough idea of the size of this compartment without using the contour. Then the centre of each protein-cluster is calculated using a similar method. The algorithm checks for every cluster if it is positioned within the boundaries of the assigned cell compartment. If it is not placed in the right compartment the whole cluster is moved to its assigned compartment (a detailed description of this algorithm is presented in section 4.5.

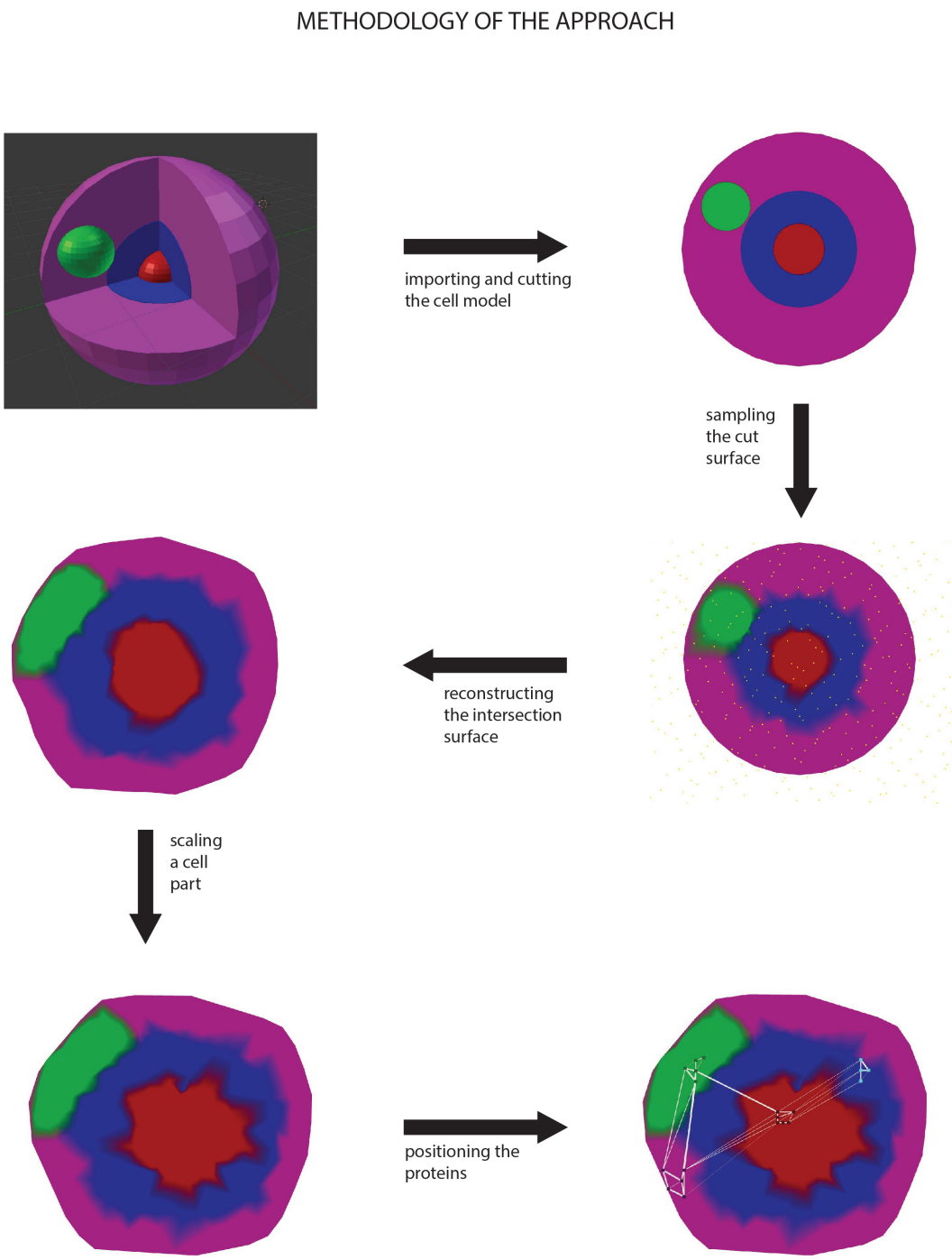


Figure 3.6: An overview of the methodology of the approach showing the pipeline.

CHAPTER 4

Implementation

The following chapter gives details on the implementation of the different techniques explained in chapter 3. Furthermore there is information about the system requirements as well as all used libraries that the program uses.

4.1 Program Environment and Installation

The program is written in C++ and OpenGL/GLSL. It was developed and tested in Visual Studio 2015 in Debug x86 mode, with an NVIDIA GTX950M and 8GB RAM.

Additionally it uses the following libraries which need to be installed and linked for it to compile and run properly. GLEW32 Version 2.1.0 for determining which OpenGL extensions are supported [gle, 2019] and GLFW3 Version 3.2.1 [glf, 2019] for creating windows. GLM Version 0.9.9.4 [glm, 2019] because it provides classes with the same naming conventions used by glsl which is important for the shaders to work. CGAL Version 4.13.1 [The CGAL Project, 2019] for the Delaunay traingulation and boost Version 1.66.0 [boo, 2019] for CGAL to work.

4.2 Importing and Cutting the Cell

The first step in the pipeline is to import a three-dimensional cell model and cut it using cutting planes. The following sections will explain the implementation of this steps in detail.

4.2.1 Generating the Cell in Blender

The cell model used in the project was generated using Blender[ble, 2019], a free 3D-modeling program. First the four spheres were generated and each of them was assigned

distance	vertex position
$= 0.0$	on plane
< 0.0	outside of plane
> 0.0	inside of plane

Table 4.1: Vertex distances to the clipping plane.

a different material and therefore different color. Four spheres are used to keep the performance high by not adding too many compartments into the prototype. Any number lower than four would not have shown the cases of compartments being encased in other compartments, therefore four is the chosen number. In this case, the main goal of the color selection was to get contrasting colors, because it is important that the differences between the cell parts are very visible in the final intersection. In the next step the position, rotation, and scale of each sphere were included in the coordinates of the vertices. Since the model is exported to a collada file this means that there is no transformation matrix included in this file and the transformations are included in the coordinates of the vertices instead. The advantage of that is, that it is not necessary to implement transformations in the program, the disadvantage is that it makes the positioning, rotating and scaling of the three-dimensional object less flexible. Since it is not necessary to move the object in the program, the lacking flexibility is not an issue.

4.2.2 Importing the Cell Using ASSIMP

The collada file was imported into the program using ASSIMP(Open Asset Import Library) which is an importing library for most three-dimensional-model formats. First the whole object is imported as a scene object and split into the individual meshes. For each mesh the vertices, indices of the vertices and the material are imported. The color is derived from the material and for each part of the cell an individual mesh object is generated. Each mesh is rendered individually when the draw method for the cell object is called.

4.2.3 Cutting Planes in OpenGL

The cutting plane is implemented using OpenGL (Open Graphics Language) and the `gl_ClipDistance` function in the vertex shader. The function provides a mechanism for vertex clipping. It specifies a distance to the clipping plane P. The distance for each vertex to P is calculated and all vertices with a distance smaller than 0.0 are clipped and therefore no longer rendered [Khronos, 2019]. The orientation of the clipping plane for the cell object is defined as

$$\overrightarrow{P} = (\,0, 0, -1, 0\,)\,.$$

The cutting of the cell creates 4 hollow semi spheres that are rendered. This rendering is the base for the sampling and reconstruction of the cell intersection surface.

4.3 Generating the Intersection Mesh

For the generation of the intersection mesh the original cut and rendered cell object is sampled. The coordinates of the sample points are generated using the Halton sequence and the color of the screen at those coordinates is sampled. Those sample points are used to create the vertex positions for the intersection mesh. The edges were created using the two-dimensional Delaunay triangulation function of CGAL (The Computational Geometry Algorithm Library)[The CGAL Project, 2019].

4.3.1 Halton Sequence

For the Halton sequence the algorithm 4.1 is used. It is called two times per sample point that is created. Once for the x-coordinate with a base of 2 and once for the y-coordinate with a base of 3. Those numbers where used because they are prime numbers. The base represents b from the radical inverse function (for a more detailed explanation see section 3.3.1). The algorithm is called in a loop and the counter variable of this loop is used as the index, therefore creating different coordinates for every sample point.

Algorithm 4.1: generateHaltonSequence

Input: An integer *index*, an integer *base*
Output: A double *result*

$f \leftarrow 1$;
$result \leftarrow 0$;
while $index > 0$ **do**
  $f \leftarrow \frac{f}{base}$;
  $result \leftarrow result + f \cdot (index \mod base)$;
  $index \leftarrow \frac{index}{base}$;
end
return *result*

4.3.2 Delaunay Triangulation

For the triangulation, a point set created from the sample point coordinates is used. Then the `Delaunay_triangulation_2` by CGAL is used to create the mesh via Delaunay triangulation. This class represents the Delaunay triangulation of a point set in a plane. It creates a triangulation that fulfills the empty circumdisk property (explained in section 3.3.2) [The CGAL Project, 2019].

4.4 Scaling Cell Compartments

For the scaling of the cell parts a force-directed algorithm 4.2 is used. This algorithm is an implementation of the simple spring embedder explained in chapter 3. The main

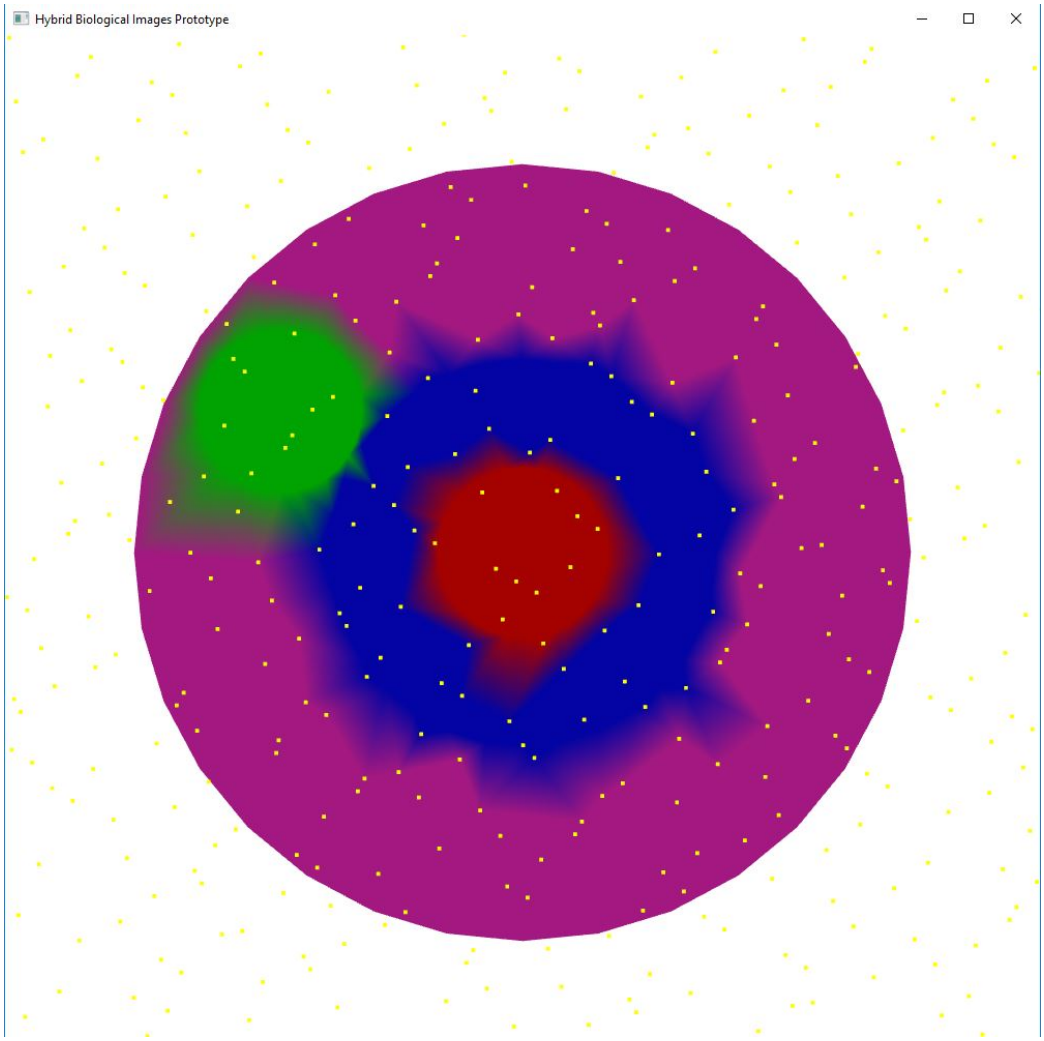


Figure 4.1: Sample points generated with the Halton sequence ontop of the cell model

adjustment to the algorithm for using it for scaling is that the ideal spring length is changed according to the scaling factor. Then the constant for the calculation of the repulsive force (in the following `c_rep`) is enlarged every iteration for all vertices that are included in the cell compartment that is scaled. Therefore `c_rep` is different for vertices that are part of the scaled cell compartment and the rest of the vertices. This makes the repulsive force onto the scaled vertices stronger which causes them to drift apart. Therefore it creates a larger surface of the cell part. The Delaunay triangulation is applied before and after the scaling to make the mesh more stable. This results in a reduction of edge crossings. Figure 4.2 shows a comparison of the final mesh with and without remeshing after every iteration.

Algorithm 4.2: scaleCellCompartment

Input: An undirected graph G = (V,E) , an undirected graph $G_{scaled} = (V_{scaled}, E_{scaled})$, number of iterations $K \in \mathbb{N}$.
Output: void

$t \leftarrow 0$;
$crep \leftarrow 0.001$;
$crep_{scaled} \leftarrow 0.001$;
$cspring \leftarrow 0.1$;
$threshold \leftarrow 0.1$;
while $t < K$ **do**
  for $u \leftarrow 0$ **to** $number\,of\,vertices$ **do**
    for $v \leftarrow 0$ **to** $number\,of\,vertices$ **do**
      $distance \leftarrow euklidean\,distance(p_u, p_v)$;
      if $u \neq v \wedge distance > 0$ **then**
        $l \leftarrow idealSpringLength(p_u, p_v)$;
        if $u, v \in E$ **then**
          $f_{spring}(p_u, p_v) = cspring \cdot \log \frac{||p_u p_v||}{l} \cdot \overrightarrow{p_v p_u}$;
        else
          if $u \in V_{scaled}$ **then**
            $f_{rep}(p_u, p_v) = \frac{crep_{scaled}}{||p_v - p_u||^2} \cdot \overrightarrow{p_u p_v}$;
          else
            $f_{rep}(p_u, p_v) = \frac{crep}{||p_v - p_u||^2} \cdot \overrightarrow{p_u p_v}$;
          end
        end
      end
    end
    $displacement_u \leftarrow \sum_{u:(u,v) \notin E} f_{rep}(p_u, p_v) + \sum_{u:(u,v) \in E} f_{spring}(p_u, p_v)$;
    $p_u \leftarrow p_u + displacement_u$;
  end
  $t \leftarrow t + 1$;
end
return

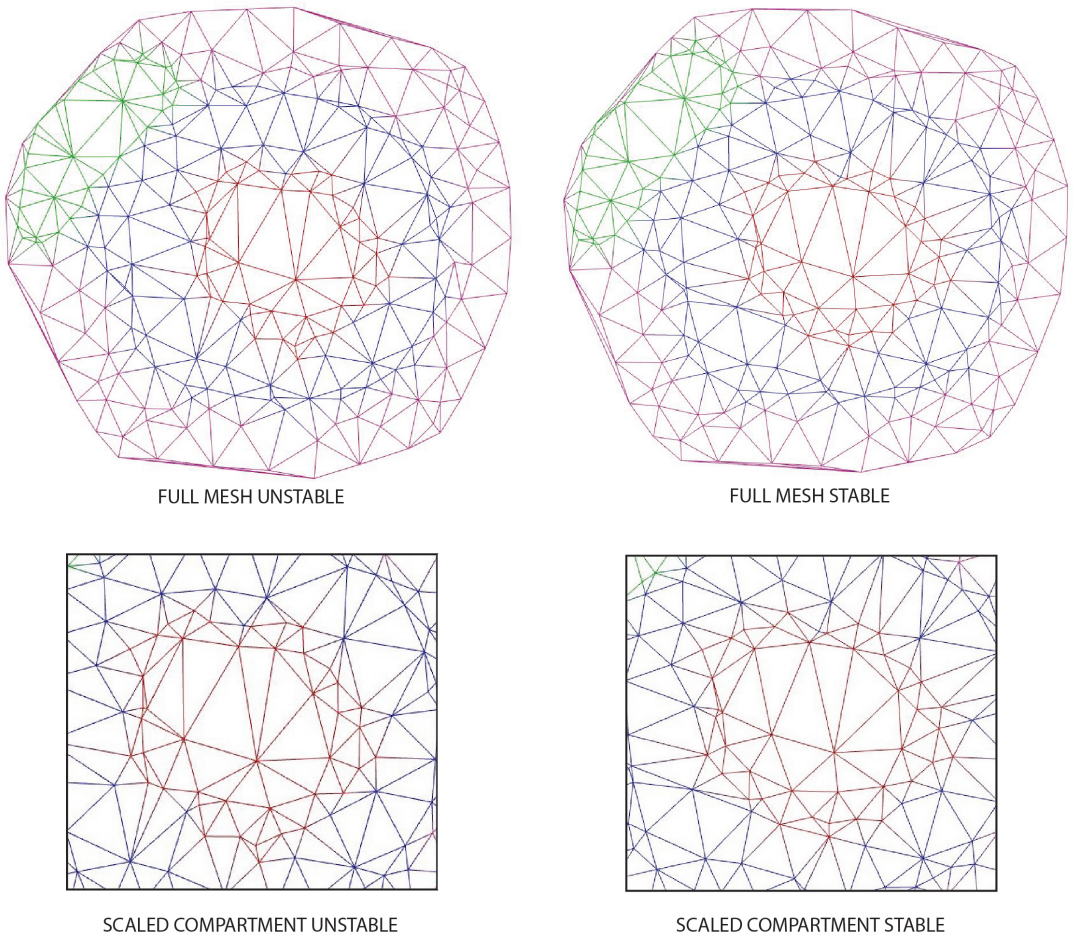


Figure 4.2: The mesh without remeshing (unstable) and with remeshing (stable).

4.5 Positioning Protein Graph Vertices

The last step is to position the vertices of the protein graph in their assigned cell parts. For this task multiple approaches were tested.

The first one was to render the protein graph and then two algorithms were applied. These algorithms used the minimal and maximal values of the coordinates of the cell compartment as boundaries to check if the vertex is in the right part of the cell. If the vertex is placed outside of its assigned cell compartment a random position inside the cell compartment is calculated and the vertex is moved. This was applied to each vertex and after all, vertices are placed inside their cell compartments the protein graph was rendered again. This approach placed the vertices of the protein graph in the right compartments, but it led to a lot of edge crossings which made the visualization very confusing. Therefore, another approach was chosen.

The new approach uses the clustering of the protein graph. Because of this it only works with protein graphs where the protein nodes are already clustered by their assigned cell compartment. Then the algorithm 4.3 is applied. First for each cluster the centre of the cluster is calculated by getting the maximum x- and y- coordinates and calculating the middle. After this the program checks if the cluster is already inside the assigned cell compartment. For this the maximum x- and y-coordinates of the compartment mesh are used. If the cluster is not inside the assigned cell compartment the centre of the cluster is positioned in either the centre of the compartment (for compartments that do not enclose other compartments) or on one side of the compartment (for compartments that enclose other compartments). If the centre of the cluster is repositioned all the vertices belonging to the cluster are moved to the compartment as well. This method gives a clearer layout with less edge crossings than just randomly positioning the vertices which is why it was the one used in the visualization in the end.

Algorithm 4.3: positionProteinVertices

Input: An undirected Graph $G_{cell} = (V_{cell}, E_{cell})$ that is sorted by compartments an undirected Graph $G_{protein} = (V_{protein}, E_{protein})$ that is sorted by clusters

Output: void

```
1 for Cluster_0 to Cluster_n do
2 |   clusterCentre ← calculateCenterOfCluster() ;
3 |   repositionedClusterCentre ← moveClusterCenterToCompartment() ;
4 |   if clusterCentre ≠ repositionedClusterCentre then
5 |   |   moveClusterNodes() ;
6 |   end
7 end
8 return
```

CHAPTER 5

Result

This chapter presents the results of the thesis. It will also describe the finished approach in detail and give examples for the scaling of the cell compartments.

5.1 Sampling and Reconstruction

The usage of the Halton sequence to calculate the coordinates for the sample points gives a good pseudo-random distribution of sample points. For performance reasons, only 500 sample points are used. Because this leads to a very imprecise reconstruction of the different compartments of the cell it is necessary to include the cell compartment contour vertices of the original three dimensional cell object in the reconstruction as well. Since the whole intersection mesh is rendered as one mesh with different colored vertices the edges of the cell compartments are not sharp but gradually changing color from one cell compartment to the other. Since the original intersection has sharp contours the reconstruction would be closer to the original if the compartments were rendered individually. But because it is necessary to have only one mesh for the force-directed algorithm, all compartments are rendered as one mesh with different colored vertices.

5.2 Force Directed Algorithm For Scaling Compartments

The force-directed algorithm works well for the scaling of the single cell compartments. It works by not only changing the length of the ideal spring length but also gradually enlarging the repulsive force. This leads to a gradual resizing of the cell compartment. A main issue was the problem of the contours becoming a zig-zag line instead of an even circle when scaling. A solution to this is repeated application of the Delauney triangulation (in the following re-meshing). The introduction of re-meshing after every iteration improved the results significantly. It did however not solve the problem perfectly. The middle cell compartment (blue) still has some vertices that stray too far off the

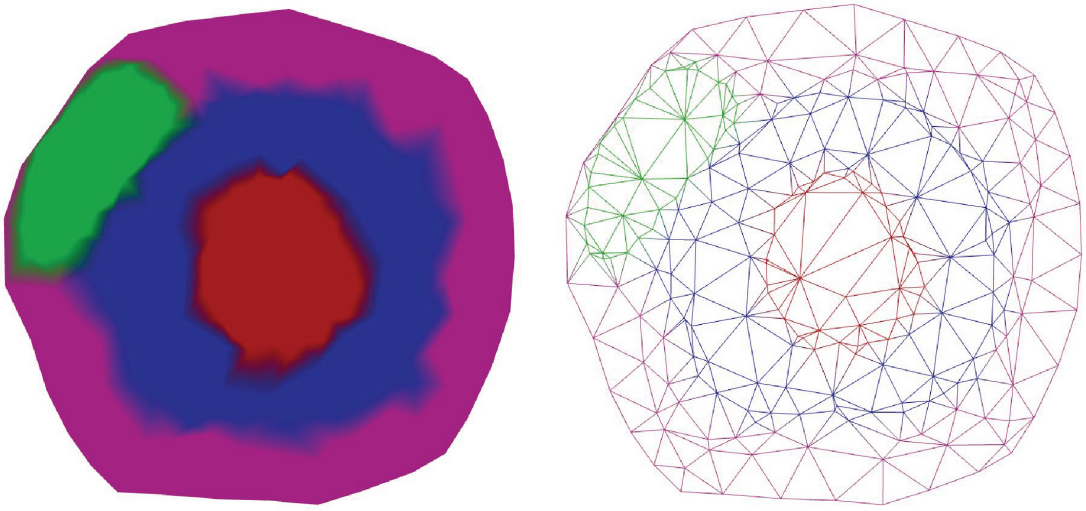


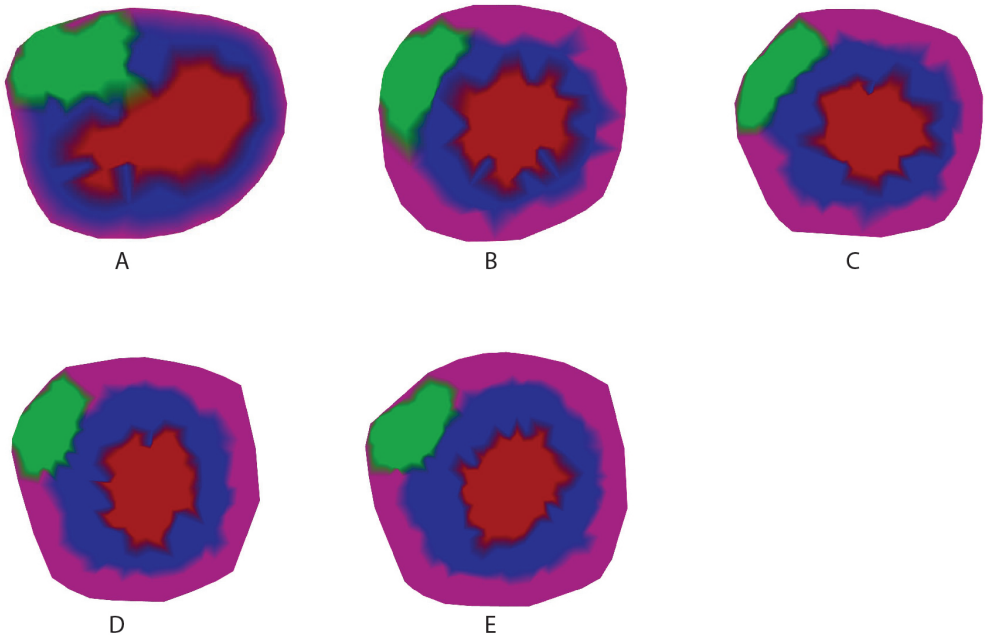
Figure 5.1: The sampled mesh with the force directed algorithm applied. All cell compartments have their original size.

contour after scaling. This can be seen in figure 5.4 . A problem of the re-meshing is that it leaves holes in cell compartments when vertices that belong to an outer cell compartment are encased in another one. This issue is solved by checking for these holes after the scaling is done and closing them. The closing is done by coloring these vertices the same color as the cell part that encases them. A major issue of the force-directed algorithm is the runtime. Since it runs in $\theta(n^2)$ it gets really slow when dealing with a larger number of vertices (100 or more). Therefore the prototype runs up to 20 minutes when scaling cell compartments on the testing hardware. For testing a notebook with 8 GB RAM, an intel core i5 and a NVIDIA GTX950M was used.

Table 5.1 shows a comparison of runtimes and results. These data is for 251 iterations where 40 are just the force directed algorithm without scaling and 211 are scaling the inner cell part with a factor of 0.2 . As shown in figure 5.2 the result improves with a higher number of sample points. Therefore reducing the sample points or the size of the original cell model impairs the outcome. Since using more sample points also increases the time until the determination of the algorithm it is important to use enough sample points to get a sufficient result in reasonable time. For this reason 500 sample points are a good setting to achieve a good result.

5.3 Positioning the Protein Graph

The positioning of the protein graph works by using boundaries to check if the proteins are rendered in the right compartment. Since it uses the minimal and maximal position of the vertices of a cell compartment as boundaries it is not very precise at the moment.



A) Cell model 1/2 size, 500 sample points B) Cell model, 250 sample points C) Cell model, 500 sample points D) Cell model, 750 sample points E) Cell model, 1000 sample points

Figure 5.2: Results of the performance comparison in table 5.1

Model	Sample Points	Mesh Vertices	Time (mm:ss)	Accuracy
Cell (scaled down)	500	141	05:48	low
Cell	250	185	10:26	low
Cell	500	241	18:15	average
Cell	750	395	28:31	average
Cell	1000	350	41:12	good

Table 5.1: Comparison of algorithm performance with different models and different numbers of sample points. All of these options were tested on a notebook with 8 GB RAM, an intel core i5 and a NVIDIA GTX950M.

A threshold helped to solve this issue. This threshold is applied to the boundaries before checking if the protein is positioned correctly. Another issue is that even though the proteins are placed correctly it may not look like it on the final cell. For long edges, the gradual change of color from one cell part to another can be a problem if the protein is placed too close to a contour. This problem could be improved by using more sample points and therefore having shorter edges. Despite these issues the algorithm produces a decent placement of the protein vertices.

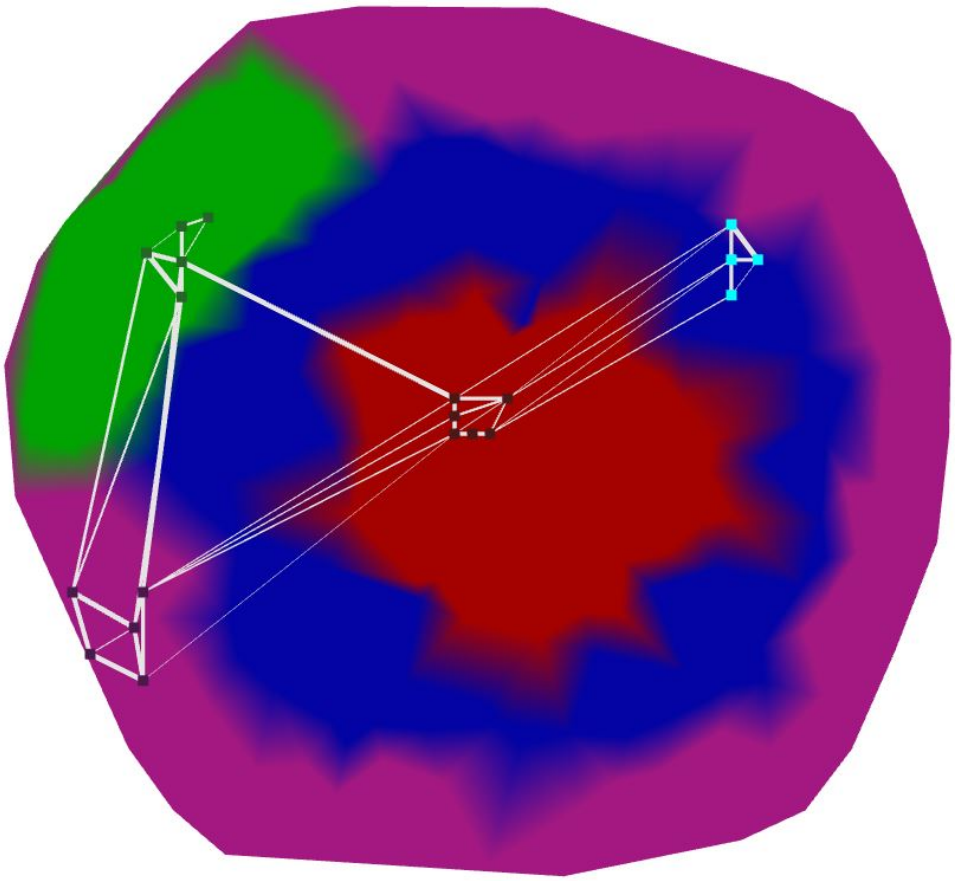


Figure 5.3: The final result with the inner compartment of the cell scaled and the protein graph applied.

5.4 Limitations

The program supports cells with up to four compartments right now. The reason for this is that those are really suitable for testing since they have compartments enclosed in other compartments but the performance is decent as well. This could be easily changed in future work on this project by adding a method that counts the different colors that are sampled and therefore concludes how many compartments there are.

Right now the scaling factor has to be set manually. This is done based on experimental values. A future approach could be calculating the area of a compartment and then calculate how many vertices of the protein graph would fit in that area. Based on that the program could decide how much to scale a certain compartment in order for all corresponding proteins to fit.

Another limitation is the size of the constant for the calculation of the repulsive force `c_rep`. When it gets too big the algorithm gets unstable. Therefore it is important to use small scaling factors for that constant. In the results shown in this thesis, `c_rep` was multiplied with 1.01 every iteration. But the combination of scaling the ideal spring length as well as `c_rep` makes sufficient scaling possible without the need of a huge value for `c_rep`.

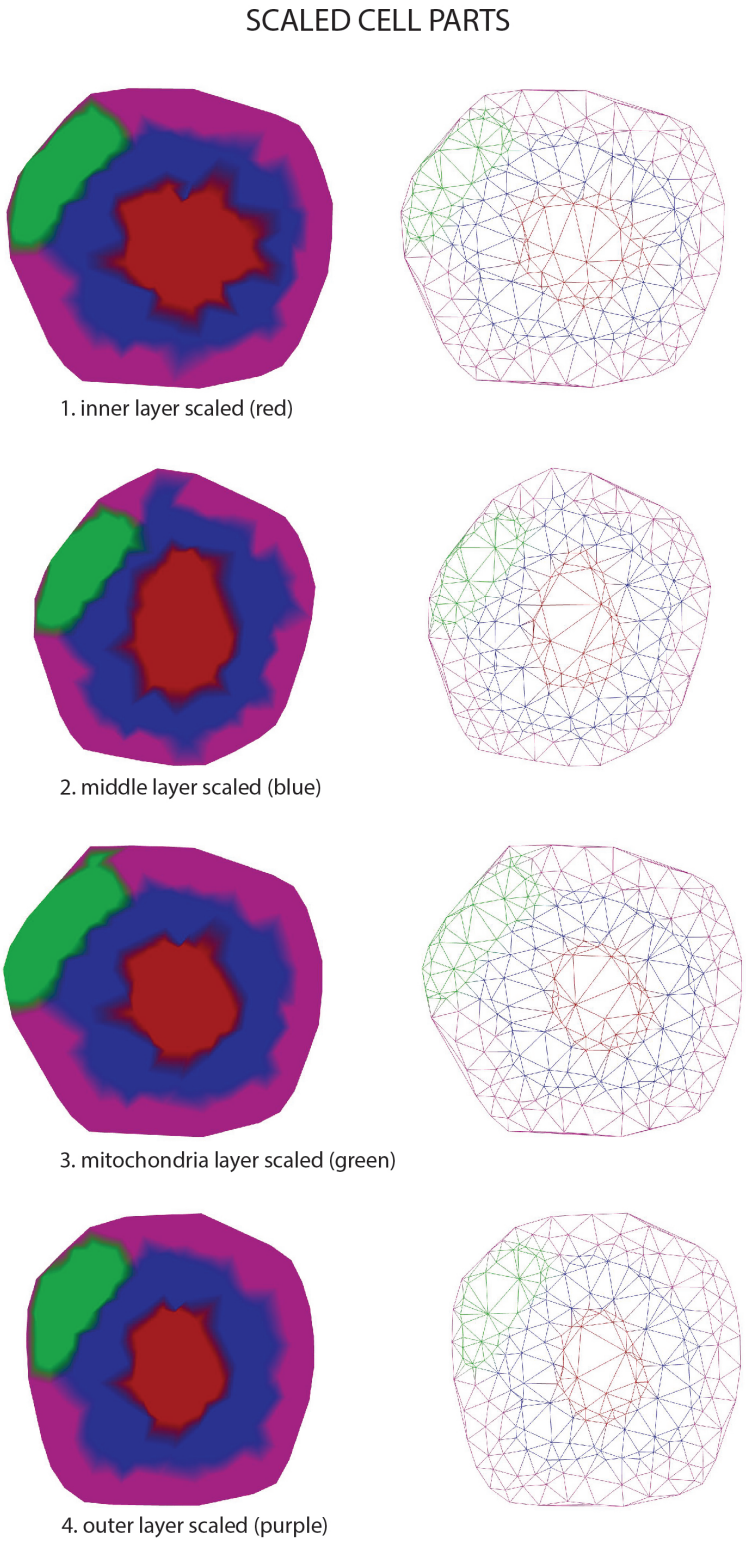


Figure 5.4: The scaled cell compartments.

CHAPTER 6

Conclusion and Future Work

The following chapter will give a conclusion on the work presented in this thesis. Furthermore it will explain future approaches that can be taken.

6.1 Conclusion

In this thesis, a prototype for visualizing protein positions in cells is introduced. The methodology of the approach centers around the force-directed algorithm that enables the scaling of single cell compartments. This is an important step when the original size of the cell compartment is too small for its proteins to fit. The scaling ensures that all proteins can be placed in their assigned compartment of the cell. For the force-directed algorithm to work it is necessary to sample the cut surface of the cell. With the coordinates and colors of the sample points it is possible to reconstruct the intersection surface as one mesh. To this mesh the force direct algorithm is applied to scale cell compartments. Then a protein graph is rendered and the proteins are placed in their assigned cell compartments. This placement is implemented using the maximum and minimum positions of the vertices of the intersection mesh inside that cell compartment. The algorithm checks if the proteins are in their assigned part and if not it moves them. Together the cutting planes, force-directed algorithm and placing of the protein graph achieve the goal of the project to visualize protein interactions in their assigned cell compartments.

6.2 Future Work

Since this project implements a basic version of the pipeline of this approach there are many possibilities for future work. There is the possibility to apply this method to the cellView[cel, 2019] framework. The framework visualizes large biomolecular data sets like large viruses and bacterial organisms [Le Muzic et al., 2015]. Since the prototype

uses a very simplified cell model that only has four parts using this framework would make the visualization more realistic.

Another possible approach in the future is to implement a more sophisticated force-directed algorithm. Right now the project uses a simple spring embedder. This algorithm is not optimal for large graphs. That's why using an algorithm that works with larger graphs, may improve the result significantly. The main issue is the performance of the force-directed algorithm. Since it is very slow this should be a focus point in future work to make the method more user-friendly. Another way of approaching the improvement of the performance would be to keep the simple spring embedder but implementing it on the GPU. Since right now it is implemented on the CPU the displacement of the vertices is calculated one after another. By using, for example, a compute shader in OpenGL it would be possible to calculate the displacement for all vertices at the same time. This would lead to improved performance of the algorithm. When the performance improves it is also possible to use more sample points and make the reconstruction of the intersection more accurate.

The protein graph can be improved by using the contour of the cell compartment as a boundary instead of the minimal and maximal position. Right now, the protein graph is positioned by positioning the clusters within these boundaries. To make it possible to use the contour of the cell compartments as boundaries, it would be necessary to access the triangles of the intersection mesh to check for every triangle if the current protein node is inside. An option for this is adding an algorithm that calculates the triangles, as well as the edge positions and checks if a vertex is contained in the triangles. Another option would be to change the implementation of the protein graph completely and use CGAL for the generation, which would make the triangles accessible. Another possibility that was tested for the positioning of the protein graph was the usage of the same force-directed algorithm that was used for the scaling of the compartments of the intersection mesh as well. The tests showed that it worked for the core, the middle compartment as well as the outer layer. But the mitochondria was a problem since there is no way to ensure that the force-directed algorithm will move the proteins to their according cell compartments. If the displacement goes in a different direction than the assigned cell compartment the proteins will never be placed there. This would make great changes to the algorithm necessary to navigate the displacement in the direction of the assigned cell compartments. For those reasons the simpler option of placing the clusters by boundaries was chosen to show the placement of the proteins on the intersection surface even though there are other options available.

List of Figures

List of Algorithms

Bibliography

[ble, 2019] (2019). blender.org. url: https://www.blender.org/ , date accessed: 2019-09-27.

[boo, 2019] (2019). Boost C++ Libraries. url: https://www.boost.org/, date accessed: 2019-09-29.

[cel, 2019] (2019). cellVIEW. url: https://www.cg.tuwien.ac.at/cellview/, date accessed: 2019-09-27.

[gle, 2019] (2019). GLEW: The OpenGL Extensions Wrangler Library. url: http://glew.sourceforge.net/ , date accessed: 2019-09-29.

[glf, 2019] (2019). GLFW - An OpenGL Library. url: https://www.glfw.org/, date accessed: 2019-09-29.

[glm, 2019] (2019). OpenGL Mathematics. url: https://glm.g-truc.net/0.9.9/index.html, date accessed: 2019-09-29.

[Cheng et al., 2012] Cheng, S.-W., Dey, T. K., and Shewchuk, J. R. (2012). *Delaunay mesh generation*. CRC Press.

[Chi et al., 2005] Chi, H., Mascagni, M., and Warnock, T. (2005). On the optimal Halton sequence. *Mathematics and Computers in Simulation*, 70(1):9–21.

[Dang et al., 2015] Dang, T. N., Murray, P., and Forbes, A. G. (2015). PathwayMatrix: Visualizing binary relationships between proteins in biological pathways. *BMC Proceedings*, 9(Suppl 6):1–13.

[Eades, 1984] Eades, P. (1984). A Heuristic for Graph Drawing. *Congressus Numerantium*, 42:149–160.

[Fekete, 2009] Fekete, J. D. (2009). Visualizing networks using adjacency matrices: Progresses and challenges. *Proceedings - 2009 11th IEEE International Conference on Computer-Aided Design and Computer Graphics, CAD/Graphics 2009*, pages 636–638.

[Ganesan et al., 2016] Ganesan, N., Li, J., Sharma, V., Jiang, H., and Compagnoni, A. (2016). Process Simulation of Complex Biological Pathways in Physical Reactive Space and Reformulated for Massively Parallel Computing Platforms. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 13(2):365–379.

[Heberle et al., 2017] Heberle, H., Carazzolle, M. F., Telles, G. P., Meirelles, G. V., and Minghim, R. (2017). CellNetVis: A web tool for visualization of biological networks using force-directed layout constrained by cellular components. *BMC Bioinformatics*, 18(Suppl 10).

[Hokayem et al., 2003] Hokayem, P. F., Abdallah, C. T., and Dorato, P. (2003). Quasi-Monte Carlo Methods in Robust Control Design. *Proceedings of the IEEE Conference on Decision and Control*, 3(December):2435–2440.

[Hutchison and Mitchell, 1973] Hutchison, D. and Mitchell, J. C. (1973). *Lecture Notes in Computer Science*, volume 9.

[Khronos, 2019] Khronos (2019). OpenGL-Refpages. url: https://www.khronos.org/registry/OpenGL-Refpages/gl4/html/gl_ClipDistance.xhtml , date accessed: 2019-09-10.

[Le Muzic et al., 2015] Le Muzic, M., Autin, L., Parulek, J., and Viola, I. (2015). cellVIEW: a Tool for Illustrative and Multi-Scale Rendering of Large Biomolecular Datasets. *Eurographics Workshop on Visual Computing for Biology and Medicine*.

[Mario Cannataro, 2011] Mario Cannataro, P. H. G. (2011). Visualization of Protein Deposits. In *Data Management of Protein Interaction Networks, First Edition*, page 11.

[Salomonis et al., 2012] Salomonis, N., Xu, D., Pico, A. R., Kuchinsky, A., Hanspers, K., and Zhang, C. (2012). Mosaic: making biological sense of complex networks. *Bioinformatics*, 28(14):1943–1944.

[The CGAL Project, 2019] The CGAL Project (2019). *{CGAL} User and Reference Manual*. CGAL Editorial Board, 4.14 edition.

[Wu et al., 2019] Wu, H.-y., Viola, I., and Nöllenburg, M. (2019). Graph Models for Biological Pathway Visualization : State of the Art and Future Challenges.