Universiteit Leiden
The Netherlands

Bio-informatics

Zebrafishualizer

Robert Gijsbers

Supervisors:
Lu Cao & Katy Wolstencroft

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

29-02-2024

Abstract

Cells migrate all throughout the human body. The migration of cells is controlled using multiple different genes. Unfortunately humans have to complicated gene structures for propper testing, this is where the zebra fish comes in. Zebra fish larvae are known for their regenerative ability. When the larvae are hurt the TRL2 and MyD88 genes regulate the macrophage and neutrophil cell migration. How these cells move can tell us a lot about the TRL2 and MyD88 genes and how their regulation works. In order to know who these genes influence the macrophage and neutrophils the movement of these cells needs to be observed. The movement of the cells were recorded using GFP as a marker and a picture on multiple layers. In order to be fully able to observe the movement of the neutrophils a 3D visualization is needed. This thesis created that 3D visualization. To start the visualization the raw data had to go trough some changes, interpolation being the first. The data was changed from TIFF into a point cloud with interior points and from a point cloud with interior points into a mesh using python. The meshes were smoothed after which they were used to make a 3D animation using Godot. The animation has a separate fully controllable camera which can be used to view the 3D visualization from all angles at all times. Having the movement of these cells visualised is very useful for our understanding of the TRL2 and MyD88 genes. Using mutants of these two genes and comparing the movement of the neutrophils can give great insight in the regenerative ability of zebra fish larvae and by extension humans.

Contents

1 Introduction

1.1 biological background

When a zebra fish larval is wounded macrophage and neutrophil cells migrate towards the wound. Macrophage and neutrophils are dependent on membrane-localized pattern recognition receptors (PRRs) in order to recognise invading microbes and tissue damage which comes from the invaders [Dag14]. Invading microbes are recognised by these PRRs because of their pathogen associated molecular patterns (PAMPs) and damage associated molecular patterns (DAMPs) [R02]. To modulate the migration of these macrophage and neutrophil cells TLR2 and MyD88 are involved [vSCLea21]. Neutrophils are a type of white blood cell, they are the first to respond to an infection. Neutrophils respond by traveling to the site of the infection and ingesting the microorganism. After which the neutrophils release enzymes which kill the microorganism [Neu]. MyD88 is a proximal signaling adaptor. It is the adaptor for inflammatory signaling pathways downstream of members of the Toll-like recepors (TLR) and interleukin-1 (IL-1) receptor families [Bar14]. "TLRs are the important mediators of inflammatory pathways in the gut which play a major role in mediating the immune responses towards a wide variety of pathogen-derived ligands and link adaptive immunity with the innate immunity." [Nis21]
The partial tail amputation was performed on larvae 3 days post fertilization (dpf). Using a 1 mm sapphire blade (World Precision Instruments) on 2% agarose-coated Petri dishes. The larvae were given anesthesia with 0.02% aminobenzoic acid ethyl ester [Xie19]. The time-lapse imaging was performed on the 3 dpf larvae. The amputation was performed on both unchallenged and wounded larvae. After the amputation the 3 dpf larvae were observed using confocal laser scanning microscopy (CLMS) with 1 minute time intervals for 2 hours [vSCLea21].

1.1.1 Zebrafish

Danio rerio or better known under its common name the zebra fish can be found in South-asia in Pakistan, Bangladesh, India and Nepal [Par15]; [ea07]. They commonly reside in slow moving or stagnant fresh water. This species of fish is often used in scientific research because of its many useful qualities. It reproduces quickly with loads of children, young zebrafish are almost transparent and they can regenerate body parts such as skin, fins, hearts and spinal cord [ea07]. Their ability to produce a lot of young and regenerate body parts makes them excellent for research in regenerative medicine. Zebra fish are used in the study for a large number of diseases such as: cancer, diabetes, cardiovascular diseases, neurodegeneration and ciliopathies [BT13]. 84% of human diseases have a zebra fish equivalent. This means that the genome of a zebra fish and a human are similar [Zfi]. This means that models made from zebra fish could be a valid tool for investigating the development of the brain in neurophysiological and neuropsychiatric studies [NR21]. For this research it was important that the neutrophils and macrophages were well visualised. Using double transgenic lines with fluorescent markers for both macrophages and neutrophils were used in all mutant and sibling zebrafish lines. In combination with the transparent 3 df zebra fish larvae this made for a good test subject [vSCLea21].

1.2 Transgenic reporter line

Transgenic reporter lines represent a sophisticated and powerful tool in molecular biology and genetics research, offering invaluable insights into the spatiotemporal regulation of gene expression within living organisms. These lines are generated by integrating a reporter gene, often encoding a fluorescent or luminescent protein, into the genome of an organism of interest. The reporter gene's expression is then driven by the regulatory elements of a target gene, allowing researchers to monitor the activity of these elements in real-time and in specific tissues or cell types. This technique enables the investigation of gene expression patterns, promoter activity, and cellular responses under various conditions, contributing significantly to our understanding of complex biological processes.

The incorporation of reporter genes into transgenic organisms has proven instrumental in studying development, disease mechanisms, and cellular responses to environmental stimuli. For instance, green fluorescent protein (GFP) and its derivatives have become widely adopted as reporter molecules due to their non-invasive nature and ability to emit visible light, facilitating live imaging studies. The use of transgenic reporter lines has become particularly prevalent in model organisms such as mice, zebrafish, and Drosophila, where the availability of well-characterized genomes and sophisticated genetic manipulation tools has enhanced the utility of this approach.

A seminal study by Chalfie et al. [Cha94] in 1994 introduced GFP as a viable reporter for gene expression studies, paving the way for the development of transgenic reporter lines across various species [Cha94]. Since then, numerous advancements have been made in the field, with researchers employing a variety of reporter genes and imaging techniques to delve deeper into the intricacies of gene regulation. For instance, the use of bioluminescent reporters, such as luciferase, has enabled researchers to perform non-invasive and quantitative monitoring of gene expression in vivo, offering a valuable complement to fluorescence-based approaches [Con95].

Transgenic reporter lines have been instrumental in uncovering the dynamic nature of gene expression during embryonic development. In a landmark study, Hama et al. [Ham11] utilized a transgenic zebrafish line expressing GFP under the control of the flk1 promoter to visualize and analyze vascular development in real-time, shedding light on the intricate processes underlying angiogenesis [Ham11]. Similarly, in the context of mammalian development, transgenic reporter lines have provided essential insights into the spatiotemporal regulation of key signaling pathways and developmental events [Mad10].

Moreover, the utility of transgenic reporter lines extends beyond developmental biology to encompass the study of various diseases, including cancer. By coupling reporter gene expression with disease models, researchers can gain a deeper understanding of the molecular mechanisms driving pathogenesis. Notably, the development of transgenic mouse models expressing fluorescence reporters driven by cancer-specific promoters has allowed for the visualization and tracking of tumor progression, facilitating the identification of potential therapeutic targets [Mar09].

In conclusion, transgenic reporter lines have become indispensable tools in the molecular biologist's arsenal, providing a means to dissect the complexities of gene regulation with unparalleled precision. The continuous refinement of reporter genes and imaging technologies, coupled with their application in diverse model systems, ensures that transgenic reporter lines will remain at the forefront of biological research, driving discoveries that extend our understanding of fundamental biological processes and informing the development of innovative therapeutic strategies.

1.3 3D tracking

Three-dimensional (3D) tracking plays a pivotal role in contemporary scientific research, particularly when utilizing cutting-edge technologies such as confocal laser scanning microscopy [Paw06]. In the specific context of this study, 3D tracking involves capturing and scrutinizing the spatial trajectory of neutrophils marked with transgenic responder lines within a zebrafish larvae. Employing confocal laser scanning microscopy enables the precise imaging of fluorescently labeled neutrophils at multiple focal planes, providing a comprehensive representation of their movements in three-dimensional space over time [Paw06]. This sophisticated imaging approach facilitates high-resolution observations within living organisms, offering insights into dynamic processes at the cellular level. Through the integration of advanced tracking algorithms, researchers can analyze the behavior and migration patterns of neutrophils, thereby gaining a deeper understanding of immune responses and inflammatory processes [Yoo10]. The combination of confocal laser scanning microscopy and 3D tracking not only enhances our understanding of complex biological phenomena but also underscores the interdisciplinary nature of modern scientific investigations.

1.4 Visualization background

1.4.1 Godot

Godot, an open-source game engine, has gained immense popularity among developers for its versatility and user-friendly features. Initially designed for 2D game development, Godot has evolved to provide powerful tools for 3D visualization, including free camera movement.

Godot is a free and open-source game engine that was created by Juan Linietsky and is now developed and maintained by the Godot community. This engine offers a robust and accessible platform for game and application development, suitable for both 2D and 3D projects. It boasts a highly customizable and user-friendly interface, scripting support, and an active community, making it an ideal choice for both beginners and experienced developers [God].

Godot's versatility extends far beyond traditional game development. While it excels in that domain, it can be employed for a wide range of applications, including:

1. Game Development: Godot is widely used for creating 2D and 3D games of varying complexities. With a built-in visual editor, extensive animation tools, and a powerful scripting language, developers can efficiently bring their gaming visions to life.

2. Interactive Simulations: The engine's capability to handle 3D graphics makes it suitable for creating interactive simulations and educational applications. Its easy-to-use tools enable developers to craft immersive, informative experiences.

3. Architectural Visualization: Godot is increasingly popular in the field of architectural visualization. Designers and architects can use it to create realistic 3D renderings of buildings and spaces, allowing clients to explore designs in a more interactive manner.

4. Product Prototyping: For businesses, Godot offers a cost-effective solution for prototyping product designs in a 3D environment. This can be particularly valuable in industries like automotive, aerospace, and industrial design.

Godot's power in 3D visualization, especially concerning free camera movement, can be attributed to several key features and capabilities:

1. Efficient Rendering Engine: Godot employs a highly efficient 3D rendering engine that can handle complex scenes and assets. This engine allows for the seamless display of 3D models, textures, and animations, essential for realistic 3D visualization.

2. Extensive Scripting Support: The engine supports multiple programming languages, including GDScript, C#, and VisualScript. This scripting flexibility enables developers to create custom camera control scripts, giving them the freedom to implement free camera movement as per their requirements.

3. Scene System: Godot uses a scene-based structure that makes it easy to organize complex 3D environments. Developers can create separate scenes for different areas, objects, or levels, enhancing the manageability of large projects and enabling seamless camera transitions.

4. Built-in Physics: Godot includes a robust physics engine, which is vital for accurate object interactions in 3D environments. This feature is particularly beneficial for architectural visualization and simulations, where realistic physics can greatly enhance the user experience.

5. Community Resources: The Godot community has produced an abundance of tutorials, documentation, and open-source assets, making it easy for developers to find the guidance and resources they need to implement 3D visualization and camera movement effectively Official Godot Engine Website.

6. Export Options: Godot supports a variety of export platforms, including Windows, macOS, Linux, Android, iOS, and HTML5. This flexibility allows developers to reach a broad audience, ensuring that their 3D visualization projects can be accessed on a wide range of devices.

In conclusion, Godot is a versatile and powerful game engine that has extended its capabilities into 3D visualization, including free camera movement. Its efficient rendering engine, scripting support, and community resources make it a valuable tool for developers across various industries. Whether you're designing games, architectural simulations, or product prototypes, Godot provides the means to bring your 3D visualization projects to life, offering a high degree of customization and creative control.

1.4.2 Structure of Godot project

In the realm of Godot's game development environment, scenes form the foundation of interactive experiences. These scenes encapsulate a hierarchy of nodes, which are the fundamental building blocks dictating the game's structure. Within this architectural framework, a 3D scene assumes the role of the primary canvas, hosting elements such as cameras, lights, and 3D models. Simultaneously, a 2D node overlays this 3D environment, providing a platform for the creation of user interfaces. Integral to the 3D scene is the Spatial node, a cornerstone that represents a point in three-dimensional space. This node serves as a container for other nodes, establishing a hierarchical structure within the scene. Cameras, as pivotal components of the 3D experience, are added as nodes to define viewpoints and perspectives. These cameras contribute to the creation of a multi-dimensional and immersive virtual environment.

In the intricate orchestration of 3D visuals, the MeshInstance node takes a central role. This node is instrumental in the instantiation of 3D models, including those derived from external sources in formats such as .obj files. By incorporating MeshInstance nodes into the scene, developers seamlessly integrate custom 3D models, enriching the visual landscape.
To navigate and explore this digital expanse, a free-moving camera becomes indispensable. Attached to a parent Spatial node, the camera's position and orientation are controlled dynamically through GDScript, Godot's scripting language. User inputs, such as arrow key strokes and mouse movements, are translated into script-driven actions, enabling fluid movement within the virtual space.
In tandem with the 3D environment, the user interface is addressed through a separate 2D node. This node accommodates the creation of menus, buttons, labels, and other UI elements. Through scripting, scene transitions are facilitated, allowing for the seamless shift between different scenes. A button press, for instance, triggers a script-driven signal that initiates the loading of a new scene, contributing to the overall navigational structure of the game.
Scripts, implemented through GDScript, emerge as pivotal agents in this interactive narrative. They encapsulate the logic governing node behaviors, user interactions, and scene transitions. The synergy of nodes, models, cameras, and scripts brings forth a cohesive and dynamic gaming experience, underscoring the technical intricacies and seamless interactions within the Godot game development environment.

1.5 Main Goal

The main goal of this thesis can be summarised to one sentence: To visualize the migration behavior of neutrophils through the tail of a zebra fish. A very clear way to visualize movement is through a video of some sort, but not just an ordinary video. A video which you can look around as it is playing in order to see the migration behavior of the neutrophils from all directions. Knowing how neutrophils move under normal circumstances alone will not give to much information. The migration behavior of different types of mutants can give insight into the workings of for example neutrophils. A TLR2 mutants neutrophils might move at different speeds and directions. These differences in direction and speed can give insight in the working of TLR2. The same can be said for MyD88 mutants. The final visualization product will show a video of the neutrophils moving though the zebra fish larvea tail. It will allow full camera movement and freedom during the video.

1.6 From TIFF to animation

In order to visualize the migration behavior of neutrophils through the tail of a zebra fish from the original TIFF file some data resampling needs to be done. The original data is under sampled so some interpolation needed to be done. After the data is interpolated and no longer under sampled it can be used to make a point cloud with interior points. Using a triangulation method the point cloud with interior points can be transformed into a mesh. The mesh made from the point cloud with interior points needed to me smooth in order for the meshes to have realistic shapes. Having all the meshes in the correct shape the animation can be made.

1.7 Research question

What method can be used to visualize the migration behavior of neutrophils through a zebra fish larvae tail in 3D?

2 Related Work

2.1 Interpolation

Interpolation is a mathematical and computational technique used to estimate values that fall between known or measured data points. The goal of interpolation is to construct a function or curve that passes through the given data points, allowing the determination of values at non-data points within the range of the data. There are multiple different ways of interpolating data such as:

2.1.1 Linear Interpolation

Linear interpolation is a method for estimating values between two adjacent data points by assuming a linear relationship. It models the relationship between the independent variable (x) and the dependent variable (y) as a straight line. The interpolated value (y) at a point x lies on the line connecting the two nearest data points (x_1, y_1) and (x_2, y_2)
Linear interpolation computes the slope of the line connecting the two data points and uses it to find the value at the desired point along the line [Fla07].

$$y = y_1 + \frac{(x - x_1)(y_2 - y_1)}{x_2 - x_1}$$

Linear interpolation is suitable when the variations between layers in the TIFF file are expected to change linearly or show a relatively smooth transition. For example, if each layer represents a time frame in a sequence of images, linear interpolation can help create smooth transitions between frames, assuming that changes between consecutive frames occur linearly over time.

2.1.2 Polynomial Interpolation (Lagrange Interpolation)

Lagrange interpolation constructs a polynomial that passes through given data points. The Lagrange polynomial is a weighted sum of basis polynomials, where each basis polynomial is associated with a specific data point. The resulting polynomial is then evaluated to find the interpolated value.

$$P(x) = \sum_{i=0}^{n} y_i \prod_{j=0, j \neq i}^{n} \frac{x - x_i}{x_i - x_j}$$

Lagrange interpolation avoids solving a system of linear equations by directly forming the polynomial using basis polynomials. The basis polynomials are designed to be 1 at the corresponding data point and 0 at other points [Bur11].
Polynomial interpolation, such as Lagrange interpolation, can be useful when there is a desire for a higher-degree polynomial fit between layers. This method might be chosen if the variations between

layers exhibit more complex patterns that can be captured by a polynomial function. However, caution is needed to avoid overfitting and oscillations.

2.1.3 Spline Interpolation (Cubic Splines)

Spline interpolation divides the data range into intervals and fits a low-degree polynomial (typically cubic) to each interval. Cubic splines provide a smooth and continuous interpolation by ensuring continuity up to the second derivative at the points where the polynomials connect.
The cubic spline equation for the interval $[x_i, x_{i+1}]$ is a cubic polynomal $S_i(x)$
Cubic spline interpolation involves determining the coefficients of cubic polynomials for each interval, ensuring that the function and its first two derivatives are continuous across adjacent intervals[dB01].
Cubic spline interpolation is well-suited for creating smooth transitions between layers. In the context of a TIFF file representing images, cubic splines can help maintain the overall shape of the image and ensure continuity in intensity changes. This method is particularly useful when there is a need for visually pleasing, smoothly interpolated frames in animations or image sequences.

2.1.4 Nearest-Neighbor Interpolation

Nearest-neighbor interpolation assigns the value of the nearest data point to the location where interpolation is needed. It is a simple and computationally efficient method, often used in scenarios where speed is crucial.
$y = y_i$, where i is the index of the nearest data point. Nearest-neighbor interpolation identifies the closest data point and assigns its value to the interpolated point, making it piece wise constant [Gon18].
Nearest-neighbor interpolation is appropriate when the data changes abruptly between layers, and a piece wise constant interpolation is acceptable. In the case of a TIFF file with distinct layers representing different states or conditions, nearest-neighbor interpolation can help preserve the discrete nature of the data. It is suitable when maintaining sharp transitions without introducing intermediate values is critical.

2.2 Point cloud reconstruction

The algorithms Delaunay triangulation, Poisson surface reconstruction, Marching Cubes, and Moving Least Squares (MLS) surface reconstruction, are well-suited for transitioning from a point cloud to a mesh due to their distinct capabilities in handling different aspects of 3D data. Delaunay triangulation excels in creating non-overlapping tetrahedra connecting 3D points, forming a mesh that preserves the geometric structure of the original point cloud. This method is particularly useful when a simple and well-defined mesh is desired [dB08]. Poisson surface reconstruction proves effective in capturing fine details and intricate features present in the point cloud. By solving a Poisson equation over the volumetric representation of the data, it generates a smooth surface that accurately represents the underlying geometry [KMB06]. Marching Cubes is adept at extracting

surfaces from volumetric data by dividing the 3D space into cubes and determining the surface configuration within each cube. This algorithm is versatile and widely used in applications like medical imaging and computer graphics [Lor87]. MLS surface reconstruction offers adaptability to irregularly sampled or noisy point clouds. By locally fitting polynomial surfaces to neighborhoods of points, it generates a smooth and continuous mesh, making it suitable for data with varying point densities [Lev04]. In summary, these algorithms provide diverse solutions for mesh reconstruction, catering to different data characteristics and application requirements in 3D space.

2.2.1 Delaunay triangulation

Given a set of points in a plane, the Delaunay triangulation is a triangulation that satisfies the Delaunay criterion: For every triangle in the triangulation, no point from the given set lies inside the circumcircle of that triangle. In simpler terms, if you imagine the points as vertices of a polygon, the Delaunay triangulation connects these points with edges to form triangles in such a way that the circumcircle of each triangle doesn't contain any other points inside it. This makes it the perfect candidate to make a mesh from a point cloud, there will be no overlapping triangles on the surface and the points inside of the cell will disappear [dB08]. Step for step the method works as follows:

1. Initial Setup: Start with a set of points in a plane.

2. Triangulation: Connect these points with edges to form triangles. There can be multiple valid triangulations for a given set of points, but the Delaunay triangulation stands out due to its geometric optimality.

3. Delaunay Criterion: For each triangle in the triangulation, check if no other points from the given set fall within the circumcircle of that triangle. The circumcircle is the circle passing through the three vertices of the triangle.

4. Adjustment: If a point is found inside the circumcircle of a triangle, "flip" the diagonal edge of that triangle. This edge flipping operation reconfigures the triangulation locally while maintaining the Delaunay criterion. Repeat this step until all triangles satisfy the Delaunay criterion.

5. Final Result: The resulting triangulation is the Delaunay triangulation for the given set of points. It ensures that no point lies inside the circumcircle of any triangle, which implies a certain level of "optimal" arrangement in terms of triangle shapes and angles.

When Delaunay triangulation is used on a 3D point cloud with interior points it still operates succesful. The algorithm inherently considers both exterior and interior points. Delaunay triangulation is not limited to the convex hull of the point cloud; rather, it constructs tetrahedra that encapsulate both the surface and interior points in three-dimensional space. The algorithm aims to create a triangulation that maximizes the minimum angle of all triangles, ensuring that no point falls inside the circumcircle of any tetrahedron. As a result, interior points are an integral part of the Delaunay triangulation process, and the mesh generated will naturally include both exterior and interior vertices. This property makes Delaunay triangulation well-suited for applications where the point cloud represents a volumetric distribution of points, such as in medical imaging, computational physics, or any scenario where the data extends beyond the surface into the interior of a region. The

algorithm provides a comprehensive representation of the spatial distribution of points, capturing both surface and internal features in the resulting mesh.
Delaunay triangulation does not explicitly create a mesh that represents the interior of the point cloud. The resulting mesh primarily represents the convex hull of the input points, forming the outer surface of the spatial distribution. Interior points are encompassed within the tetrahedra, but the triangulation itself doesn't generate triangles that explicitly define the interior structure.

2.2.2 Poission surface reconstruction

Poisson surface reconstruction is invaluable in mesh generation from 3D point clouds due to its ability to capture fine details and create smooth surfaces. By constructing a volumetric scalar field from the input points and solving a Poisson equation, the algorithm produces a continuous surface that faithfully represents the underlying geometry. This method is particularly useful when dealing with unorganized and noisy point clouds, providing a robust solution for reconstructing surfaces with intricate features. The resulting mesh is not only visually appealing but also accurately reflects the original point cloud, making it suitable for applications requiring high-quality surface representations [KMB06].

2.2.3 Marching cubes

Marching Cubes is a versatile algorithm employed for transitioning from point clouds to meshes by extracting surfaces from volumetric data. In 3D space, the algorithm divides the space into cubes, determining the surface configuration within each cube based on density values. This method is well-suited for visualizing complex 3D structures and is widely used in medical imaging and computer graphics. Marching Cubes generates a mesh that faithfully represents the input data, offering adaptability to various densities within the point cloud. Its flexibility and efficiency make it a popular choice for applications where a detailed and accurate mesh representation is essential [Lor87].

2.2.4 Moving least squares

MLS is a powerful technique for transitioning from point clouds to meshes, particularly in scenarios involving irregularly sampled or noisy data. MLS fits local polynomial surfaces to points within a specified neighborhood, creating a smooth and continuous representation of the surface. This adaptability is crucial for handling variations in point density, ensuring that the resulting mesh accurately reflects the underlying geometry. MLS is effective in providing a flexible and robust solution for surface reconstruction, making it well-suited for applications where the input point cloud may exhibit non-uniform sampling or contain noise. By iteratively applying MLS to each point, a cohesive and accurate mesh can be generated from the point cloud data [Lev04].

2.3 Smoothing

2.3.1 Taubin smoothing

Taubin smoothing is a mesh smoothing algorithm used in computer graphics and geometry processing to improve the quality of 3D surface meshes. It was introduced by Gabriel Taubin in the paper titled "A Signal Processing Approach to Fair Surface Design" published in 1995. The algorithm aims to iteratively smooth the mesh while preserving sharp features and avoiding the loss of fine details. It is based on the concept of applying low-pass and high-pass filters to the mesh vertices, similar to signal processing techniques. The low-pass filter helps in reducing noise and smoothing the overall surface, while the high-pass filter helps in preserving sharp edges and geometric details.

1. Initialize the mesh: Start with the original mesh representation.
2. Compute the Laplacian: Calculate the Laplacian matrix of the mesh. The Laplacian describes the curvature of the mesh and helps in determining the amount of smoothing to be applied to each vertex.
3. Smooth the mesh: Apply a low-pass filter to the mesh vertices by moving each vertex towards the average position of its neighboring vertices. This step smooths the mesh but can also blur sharp features.
4. Correct sharp features: To preserve sharp edges and fine details, apply a high-pass filter to the mesh vertices by moving each vertex away from the average position of its neighbors. This step helps in restoring the sharpness of features.
5. Weighted combination: Adjust the smoothing effect by applying a weighted combination of the smoothed vertices obtained from steps 3 and 4. The weights are controlled by two parameters: lambda (λ) and mu (μ).
6. Iterate: Repeat steps 3 to 5 for several iterations until the desired level of smoothness is achieved.

The lambda (λ) and mu (μ) parameters control the amount of smoothing applied in each iteration. Properly tuning these parameters is essential to achieving a good balance between surface smoothness and feature preservation.

2.3.2 Laplacian Smoothing

Laplacian smoothing is a classical technique for mesh smoothing that operates by iteratively adjusting vertex positions based on the Laplacian operator. In this context, the Laplacian of a vertex is computed as the average position of its neighboring vertices. By updating each vertex towards this average, Laplacian smoothing effectively reduces irregularities and sharp features in the mesh.

This method is computationally efficient and easy to implement, making it a popular choice in various fields such as computer graphics, computer-aided design, and finite element analysis. However, it may struggle to preserve fine details and may lead to over-smoothing on certain types

of meshes. Researchers and practitioners often customize Laplacian smoothing by incorporating additional constraints or combining it with other algorithms to address specific challenges in mesh processing [Hop93].

2.3.3 Implicit Fairing

Implicit fairing is a smoothing method that focuses on evolving a surface over time to minimize a predefined energy functional. This technique operates by representing the surface as an implicit function and utilizing gradient flow to drive the evolution of the surface toward a state of minimal energy. The energy functional typically includes terms related to curvature, surface area, and other geometric properties.

Implicit fairing is advantageous in preserving fine details and important geometric features during the smoothing process. It finds applications in fields such as computer-aided design, medical imaging, and surface reconstruction. However, the implementation of implicit fairing involves solving partial differential equations, making it computationally demanding [Des00].

2.3.4 Bilateral Filtering

Bilateral filtering is a non-linear, edge-preserving smoothing method that considers both spatial and intensity information during the filtering process. It is particularly effective in applications where preserving sharp edges and fine details is crucial. In the context of mesh smoothing, bilateral filtering operates by convolving the mesh with a spatial kernel and a range kernel.

The spatial kernel determines the weights based on geometric proximity, while the range kernel considers intensity differences between neighboring vertices. This dual consideration allows bilateral filtering to smooth the mesh while retaining important geometric features. Bilateral filtering has been successfully applied in computer graphics, image processing, and computer vision tasks where noise reduction is desired without sacrificing essential details [Tom02].

3 Methodology

The visualization of this project boils down to one thing: go from a TIFF file to a 3D representation of that file.

3.0.1 Data received

This bachelor thesis uses data received from a previous research [CV22]. The data is composed of green fluorescent protein (GFP) infused neutrophils within a zebra fish. These neutrophils were tracked for 120 minutes and the position was photographed every minute. From these images an analysis of the dynamic process of neutrophil migration was made [CV22]. The photographs were in high detail considering the size of the zebra fishes and neutrophils. The photographs were taken from different dephts within the zebra fish. As a result it was possible to render a 3D reconstruction. The original TIFF files had a size of 8x512x512. This means that it consists of 8 layers in the z-axis (or depth) each of size 512x512 in the x- and y-axis. Having 8 layers does not make for a good 3D

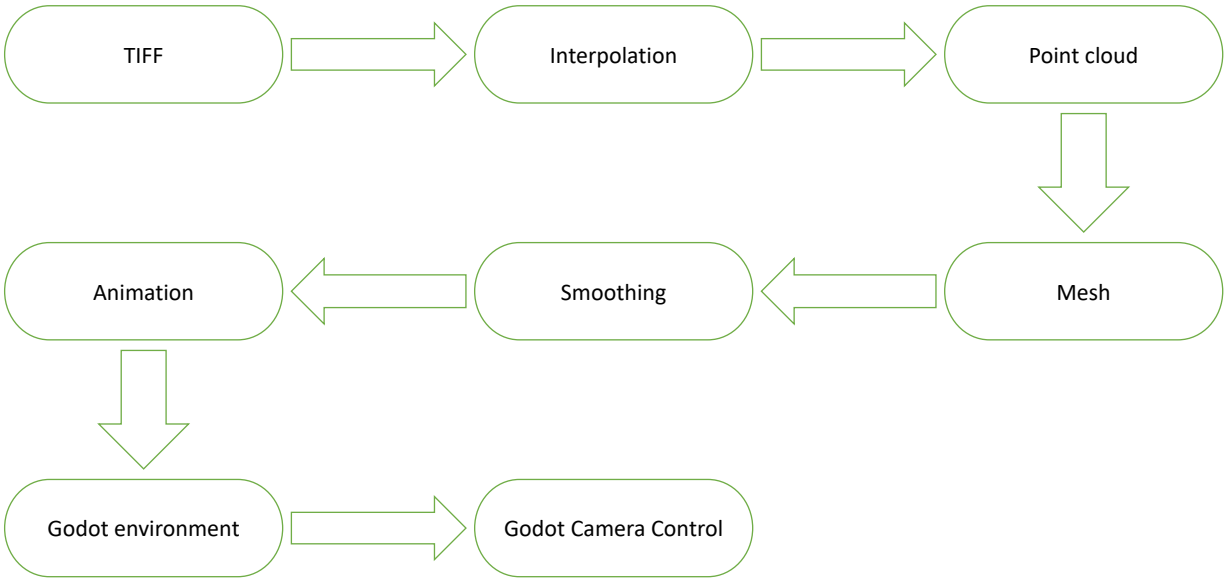

Figure 1: This figure shows a visualization of the workflow of the methodology of this project.

image, unfortunately the original data was under sampled. As a consequence of this under sampled data the visualization will appear to have big noticeable holes in between the layers. When there are more layers in the z-axis these holes will become less noticeable. There are multiple ways to get data which fits in between other data. A simple approach is to take the average from two adjacent layers and put that new average layer in between. Doing this for every pair of layers gives a new shape of 15x512x512 [CV22]. These new layers were not within the original data and are a guess of what shape the cells should have.

3.0.2 TIFF format

The data used was formatted in Tag Image File Format or TIFF. TIFF files are used because they can store high-quality photographs. This way you won't lose any quality when saving a photograph to a format which does not support high-quality photographs. TIFF is also able to support 3D images, in a way. TIFF is not able to show a 3D image but it is able to show multiple layers in the z-axis in order to reconstruct a 3D image.

In order to go from a TIFF file to a 3D representation of that file a few things have to be done first. The TIFF file which should be visualized is first checked if it is a TIFF file. If it is its dimensions are saved as variables. If the cells are labeled numerically the maximum and minimum (excluding empty spaces) values are saved as well. Each different cell within the TIFF file should have a different value for its points. So the points of cell 1 should all have the value 1, while the point of cell 7 should all have the value 7. This ensures that the program puts the cells together correctly even when they are close to each other. The TIFF files were from another earlier research [vSCLea21].

This research focused on cell tracking, the data already has segmentation labels in each frame. The segmentation labels are to make sure that each individual cell will be labeled through each frame. This way when cells are close to each other will not get mixed up.

3.1 Converting TIFF to point cloud

The data received were 120 TIFF files each representing a different point in time. The original TIFF files have 8 layers which are meant to be on top of each other The individual layers can be seen in figure 2. Every individual layer is made up from 512 by 512 pixels.

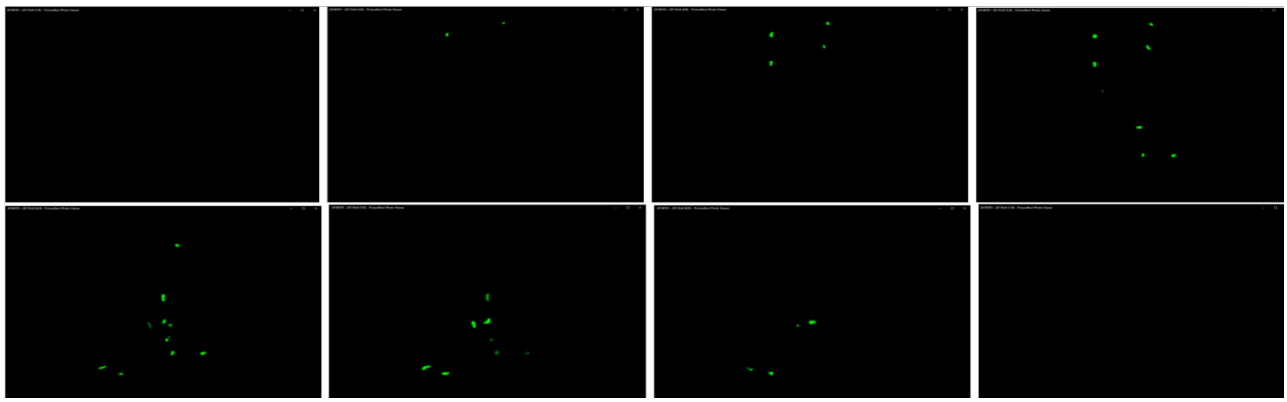

Figure 2: The 8 layers of the first time step of the original TIFF files. These layers should be on top of each other, the highest layer is the top left going right and then down.

In order to get these separate images on top of each other the edges of each cells need to be extracted. These edges are then converted into a point cloud with interior points. Each individual images was gray scaled and turned into a array. So if there was nothing visible on the pixel of the image the corresponding array number is 0. If there is something visible on the pixel of the image the corresponding array number is 1. The important thing about making every image an array is that one can make sure that every pixel is aligned perfectly. This can be done by taking all 8 arrays and layering them into a 3D array. From this 3D array a point cloud with interior points of all the cells is made. This is done using the Polydata from the pyvista library along with concatenate from the numpy library. There are 8 different 2D arrays each showing one layer of the data as seen in figure 2. Using the concatenate function from the numpy library all 8 2D arrays will be converted to one 3D array. Because of the way the layers are put on top of each other, the new 3D array converted into a point cloud will have interior points. To visualize these point each is given a colour depended on which layer they are in, so each point with the same colour are from the same layer. When it comes to transforming these point clouds with interior points into manipulable and visualizable surfaces, the Visualization Toolkit (VTK) in Python proves to be a valuable tool. In the context of mesh generation and manipulation, converting a point cloud into vtkPolyData becomes essential. vtkPolyData is a versatile data structure within VTK that accommodates various geometric elements such as points, lines, and polygons. This conversion provides a foundation for further processing, visualization, and mesh generation.

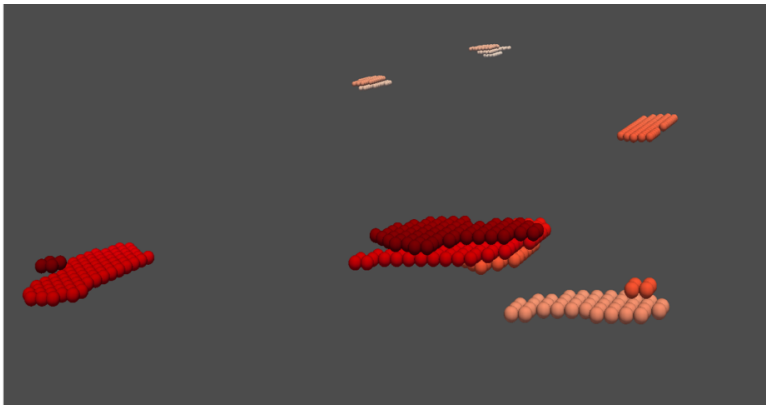

Figure 3: A zoomed in picture of the point cloud. Each orb with the same colour originates from the same layer within the TIFF file.

The visualized point are purely for the understanding of how the point cloud looks. In reality the points are points and thus infinitely small. These point can be used to construct a mesh.

3.2 Converting point cloud to mesh

In order to go from a point could with interior points to a mesh multiple ways are possible. One of these ways is surface reconstruction. The problem with using surface reconstruction is that the python library will try to make a surface from all the cells over the entire data set. This normally would be what one wants in order to reconstruct the surface of the object within the data set. However as seen from figure 2 the data set has multiple separate cells. So the pyvista surface reconstruction is not the solution to the mesh problem. The pyvista library has more options then just the surface reconstruction, it also has the Delaunay triangulation method 2. Delaunay triangulation uses the points in the point cloud to make a lot of triangles. It makes these triangles between the points, but only points close enough to each other in order to prevent the same problem as with surface reconstruction to happen. From figure 3 and 4 it can be seen that a lot of details

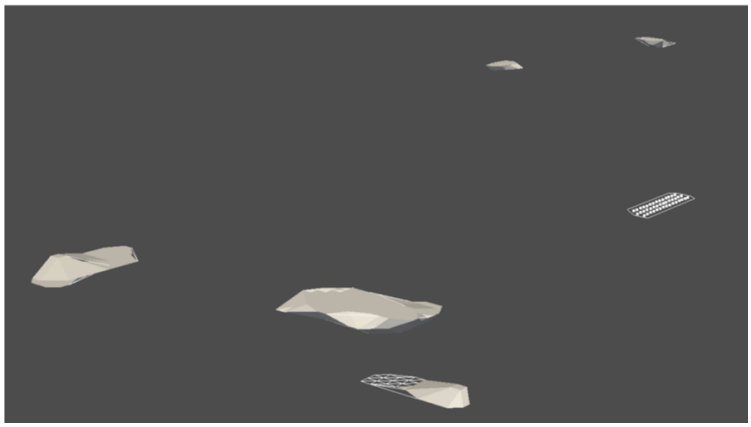

Figure 4: A zoomed in picture of the Delaunay mesh.

from the cells are lost. In order to offset this more layers were implemented. These layers were

put within the excising 3D array, going from eight to fifteen layers. These new cells in the new layers will get the average number of the two adjacent original layers. This entire process was done once more giving a total of 29 layers. Doing this interpolation of the layers helped a lot with the final result of the cells. They became vertically larger giving them a shape one would expect from living cells. A new problem which arises now is how the Delaunay triangulation makes the cells look. Because the original data is under sampled, the visualization fall flat on the top and bottom. The shape of the neutrophils is not properly shown in the mesh at all. To get closer to the original shape smoothing is used.

3.2.1 vtk smooth poly data

The vtkSmoothPolyDataFilter is a module in the Visualization Toolkit (VTK) designed to enhance the visual appearance of three-dimensional polygonal meshes by smoothing their surfaces. This filter utilizes a weighted Laplacian smoothing algorithm, which iteratively redistributes the positions of vertices based on the average position of their neighbors [VTK]. The smoothing process aims to reduce surface irregularities and enhance the overall geometry of the mesh. Users can control the level of smoothing by adjusting parameters such as the number of iterations and the relaxation factor.
During each iteration, the filter computes a weighted average of vertex positions, allowing the mesh to converge toward a smoother representation. This iterative approach is particularly useful for refining noisy or jagged surfaces, common in computer-aided design (CAD) models or medical imaging data. The vtkSmoothPolyDataFilter is versatile and can be applied to a wide range of polygonal datasets, providing users with a means to improve the visual quality of their 3D models for applications in scientific visualization, simulation, and virtual reality.

3.3 From mesh to animation

Each of the steps up until now has been done for every step of the 120 time steps. These time steps will have to be put in the same visualizer but shown one by one. Doing this in python using one of the popular visualization libraries like matplotlib is not an easy task. These visualization libraries are very good for making graphs in 2D and even in 3D. However it is not made to alternate quickly between showing different meshes. Another problem with using matplotlib is that the controls of moving around a 3D object leave a lot to be desired. Matplotlib is not the only visualization library which has these problems. So in order to prevent all these problems another approach is needed. Moving around using a camera freely while a video is playing is a difficult task, but not something which has not been done before. When playing a video game these things are already being done simultaneously. The program to solve all the python problems is Godot 2. Godot is used to make video games, these very often have the same qualities which are needed for good visualization. Some of these qualities are a free moving camera, lighting options and the ability to quickly change what is showed within the environment. Within Godot the first things which need to be made is an environment, lights and a camera. The environment is needed to place objects in in the first place, the objects in this case are the meshes of each time step. The lights are needed in order to see the objects placed within the environment, without lights the environment will be completely dark and nothing can be seen. Lastly the camera is needed in order for the user to be able to see the objects within the environment. Just having a camera is not enough to see the 3D

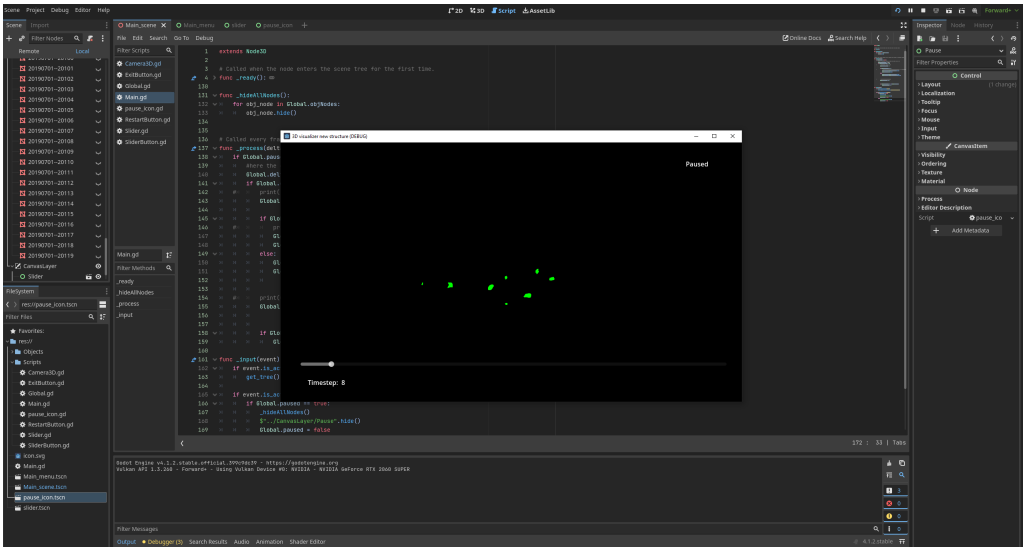

Figure 5: The Godot editor with the project running.

mesh from every angle, the camera needs to be moved. This can be done in two ways. Using the arrow keys or wasd keys to move the camera forward, backward, left or right. Using the space bar to move the camera up and the left shift in order to move the camera down. The mouse can be used to change the rotation of the camera, the way the mouse moves is also the way the pitch of the camera will move. If the mouse moves down the camera will also pitch down. The different time steps are shown in succession of each other. To indicate where in the different time steps the animation is a slider is shown at the bottom of the screen. The slider indicates at which time step the animation is along with the number of the time step. There is also a possibility to pause the animation in order to look at the cells in a specific moment in the process.

3.3.1 zebra fish tail representation

Having free camera control at all times can make it difficult for the used to stay oriented. Keeping the user oriented is achieved by having an opaque outline of the fin of the zebra fish. The outline of the fish can be seen in figure 6. This outline of the zebra fish is an approximation of what the tail of the zebra fish larvae would look like. It is shaped in such a way that it looks like a common fish tail, because of this the user will be able to tell which part of the tail has been cut and which part of the approximation is the rest of the fish.
In figure 7 the right side of the fish tail is the side which has been cut and the left side is the rest of the fish.
An extra feature the approximate representation of the zebra fish tail has is: the side which continues into the fish is intentionally left open as seen in figure 8. The open side is meant to show that the environment might not show more of the fish itself, but the fish itself continues even outside of the scope of the environment.

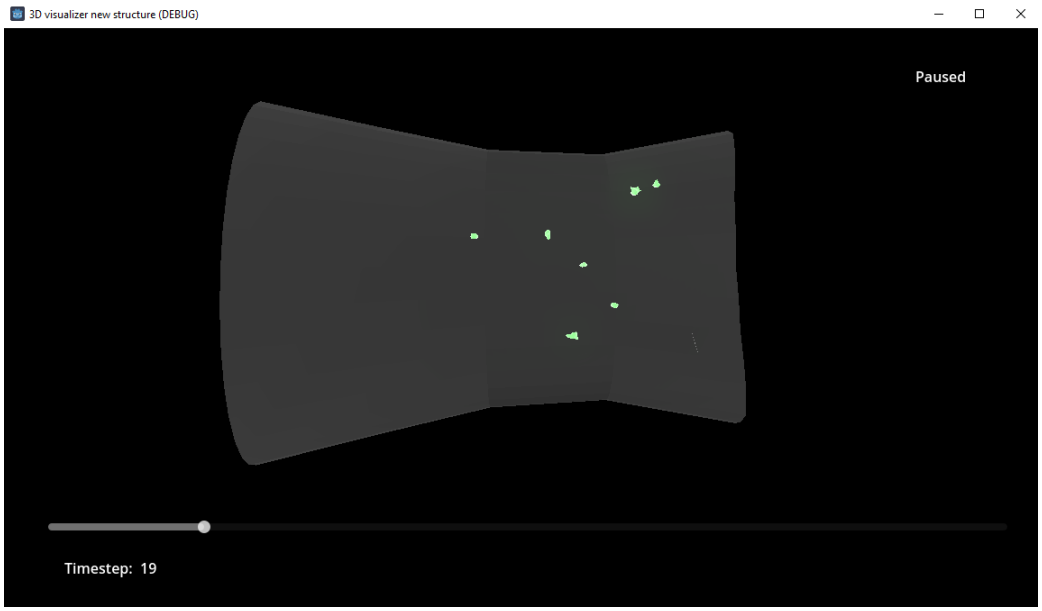

Figure 6: The tail of the zebra fish visualised within Godot. The green cells are the neutrophils marked using Transgenic reporter lines. The opaque gray object is the zebra fish tail visualized.

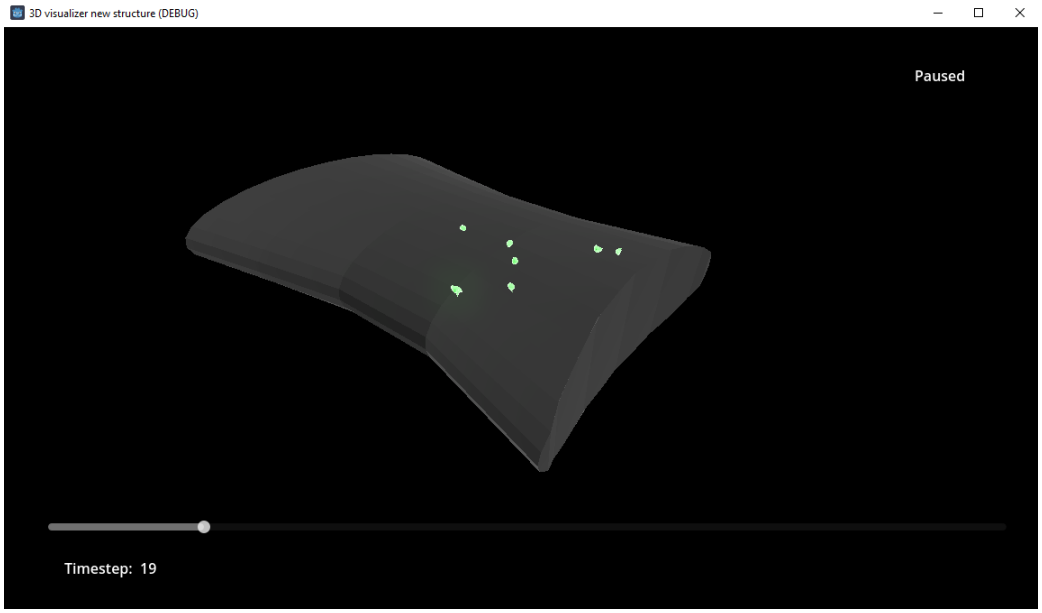

Figure 7: The tail of the zebra fish visualised within Godot. The green cells are the neutrophils marked using Transgenic reporter lines. The opaque gray object is the zebra fish tail visualized. The right side of this figure shows the closed end of the fish tail which indicates this side is the end of the tail in which had been cut.

3.4 Availability of code

The code used to run this thesis can be found on a GitHub page. This Github page can be found on this page made specifically for this project.

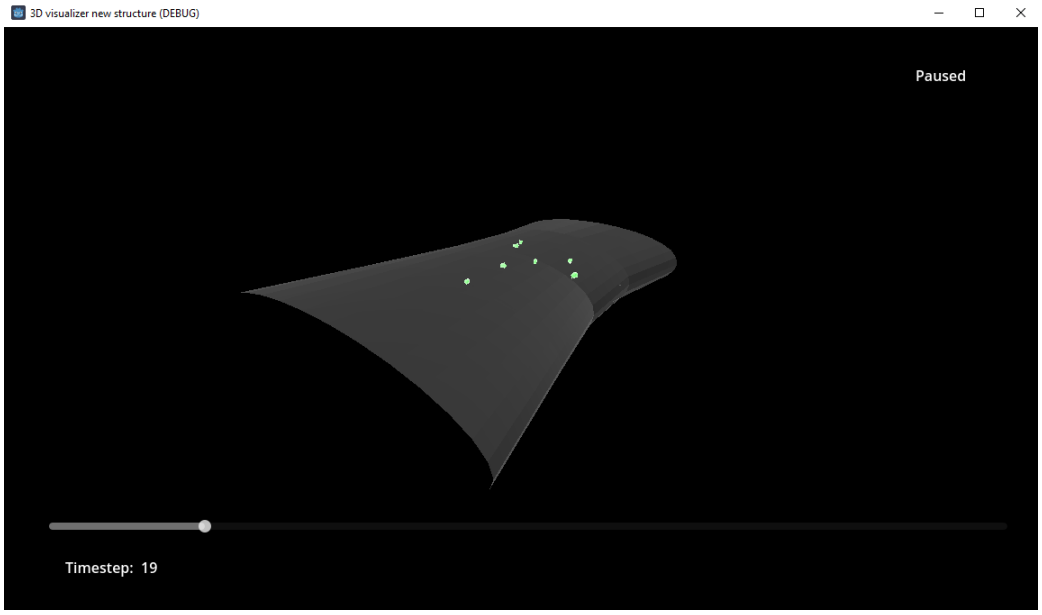

Figure 8: The tail of the zebra fish visualised within Godot. The green cells are the neutrophils marked using Transgenic reporter lines. The opaque gray object is the zebra fish tail visualized. The left side of this figure shows the open end of the fish tail which indicates this side has the rest of the fish attached to it.

4 Results

The visualization of the migration behavior of neutrophils through the tail of a zebra fish was successful. The data was used to make an animation, this animation can be viewed fully from all angles using Godot. The animation will play on screen while the camera can be moved freely in every direction by the user. While the user has free movement with the camera the animation plays in the environment. The user is able to pause the animation in order to look more closely at certain cells at a specific frame within the animation. At a fixed point at the bottom of the screen there is a slider indication on which frame the animation is at that time. In addition to the slider there is also a number increasing to give the precise frame which the animation is. Within a menu the user can restart the simulation or exit the program. Restarting the simulation is useful for when the user lost the orientation of the cells.

The final result of the visualization program is not perfect. Some features are missing while others do work but not optimally. Improving on these features will elevate the final product. One improvement that the user cannot change the speed of the animation. It will always move at the same speed which can be inconvenient if a user wants to see the final few seconds of the animation. If the user presses the escape button they can either quit the simulation or reset the simulation. This would again be inconvenient if the user wants to see the final few seconds of the animation. Image 9 shows the menu which does not contain a "continue"option.

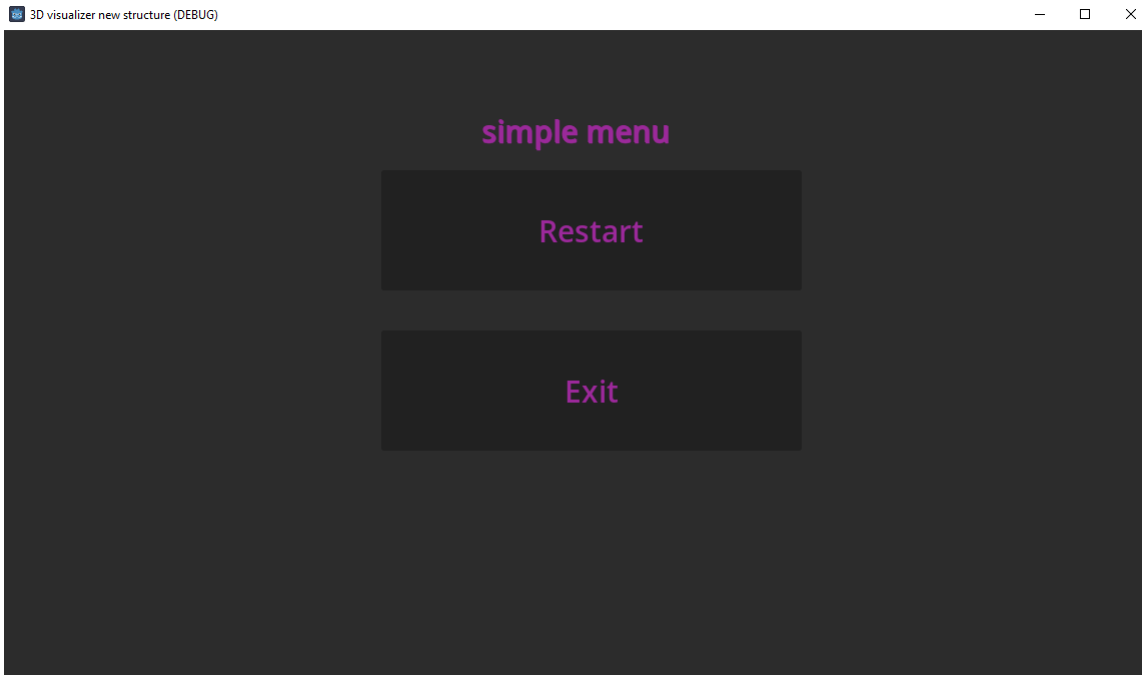

Figure 9: The menu of the Godot visualization project.

When the pause button is pressed the animation pauses as expected. However when the pause button is pressed again in order to un-pause the simulation the time step which is shown at that moment disappears. The next time step appears without issue, however with this issue the animation seems to flicker every time the user un-pauses.

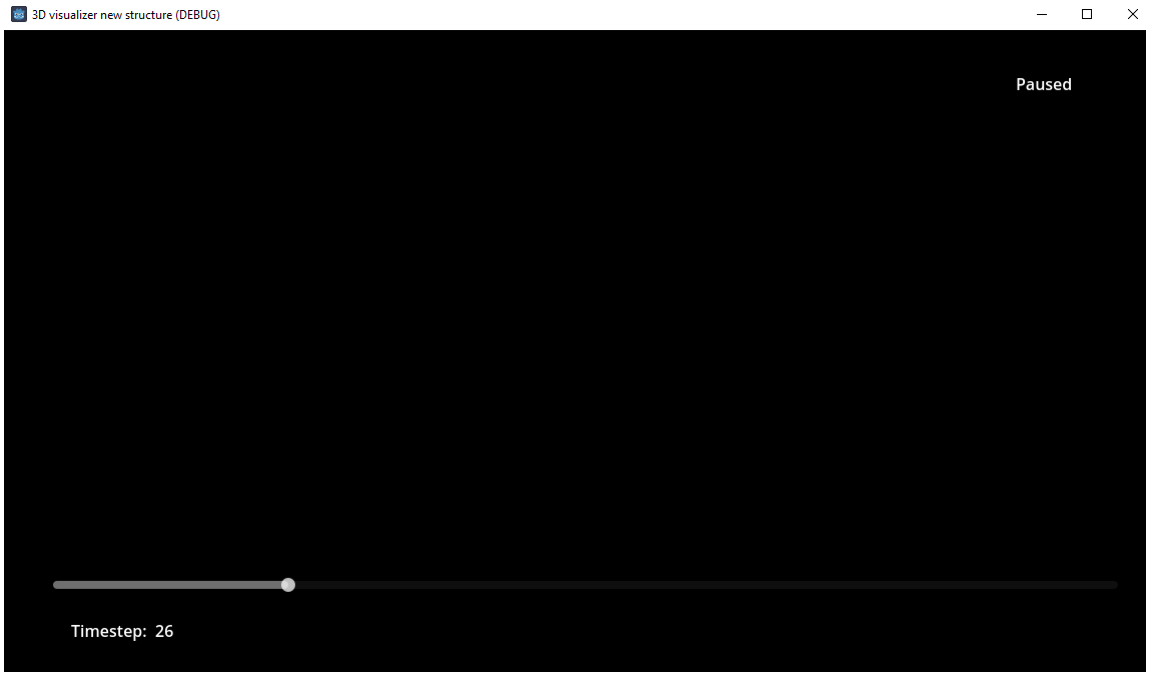

Figure 10: The example of the animation not being shown right after un-pausing

When entering full screen on the animation something goes wrong, the slider at the bottom of the screen does not resize along the window as seen in figure 11. The slider is not the only thing which does not re-scale with the window. The menu also does not re-scale as seen in figure 12. Both the menu and the slider still work as intended the only problem is that they do not appear in locations where the user would like to seen a menu and a slider.

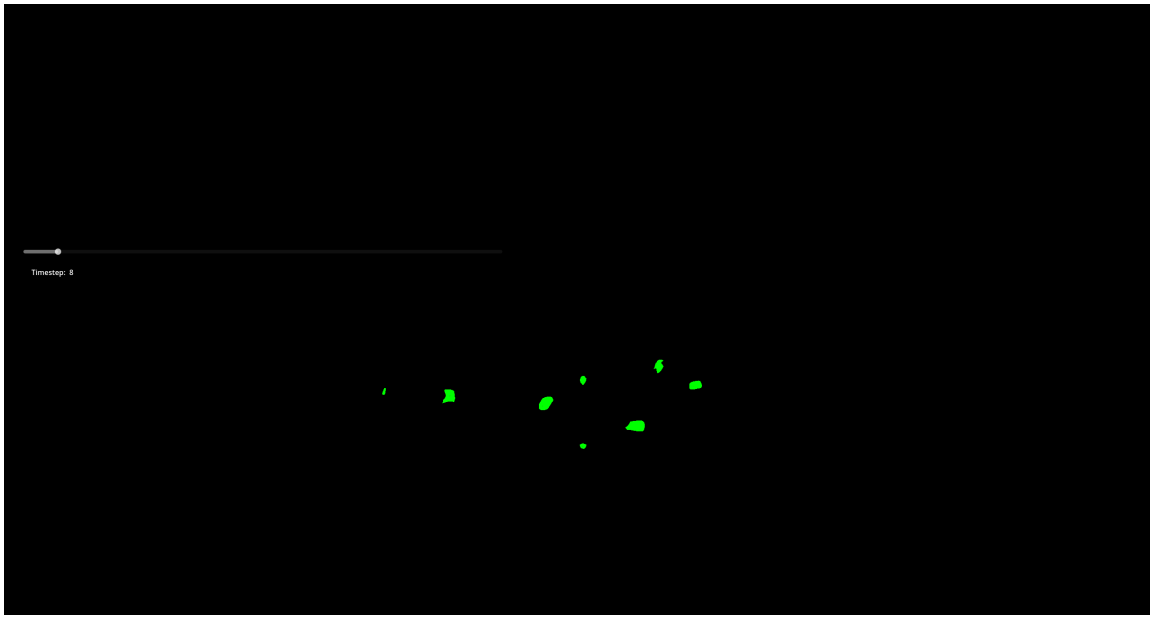

Figure 11: The example of the slider not being at the place in the screen where the user would want it.

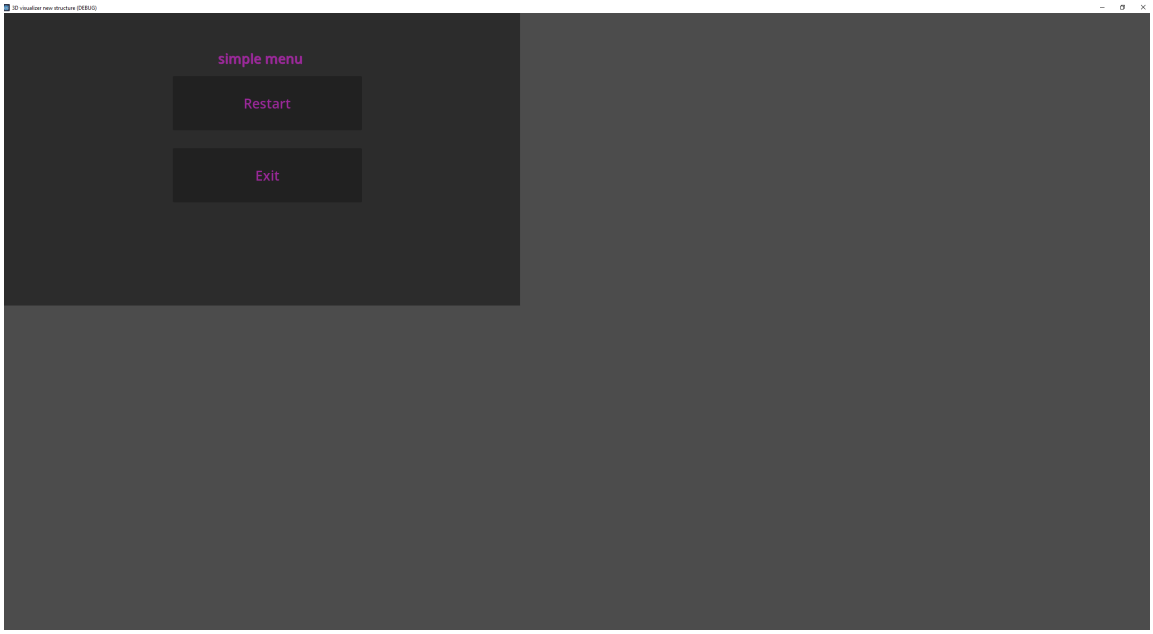

Figure 12: The example of the main menu not being at the place in the screen where the user would want it.

5 Conclusions and Further Research

The visualization of the movement of neutrophils through a zebra fish larvae tail in 3D was successful. The original TIFF files where the neutrophils are shown using GFP are made into point clouds with interior points. This is done by taking each layer of the TIFF file and converting it to an array, this was done by taking every pixel of each TIFF layer and converting it to a corresponding coordinates in a 2D array. The 2D arrays consist of zero's and one's, if the original TIFF's pixel had a GFP neurophil the value of the array is a one. If the pixel does not contain a GFP neutrophil the corresponding array will be zero. These arrays were put on top of each other to make a 3D array. This makes it so that all the layers of the TIFF file are stacked perfectly on top of each other without deviating. Each point in the 3D array will be converted into a point in an environment making a point cloud. From this point cloud a mesh is made, this is done using the pyvista Delaunay triangulation. Having done this for every TIFF file yields 120 meshes. At this point every time step is individually visualised, but the process of the movement should be visualised. So using the Godot game engine an animation was made. Viewing this animation can be done with a free moving camera as the animation plays. The user can view the animation from every angle at all time and also pause the animation for precise viewing at certain points during the movement of the neutrophils.

These limitations are inconvenient but not critical, what is critical is that the product can not show the direction or speed at which the neutrophils are moving. The product does not show the scale of the cells which makes it impossible to determine the speed of the cells. The direction can be estimated when looking from frame to frame and noting how they changed. Doing this however is time consuming and can only give an estimation and not a precise answer. Another limitation is that the user will have to download godot in order to view the product. The product is visualized within Godot and can not be viewed outside of the Godot environment. Godot itself is not a big or complicated download and the product can be easily shared, but the user will have to download Godot themselves.

Within Godot where the animation can be viewed the user might experience some small problems as explained in section 4. The animation disappears for a split second when the user un-pauses the animation. Neither can the user change the speed of the animation. This becomes problematic then the user want to see only the end of the animation. The animation runs with a variable delta which is defined by Godot itself. This variable delta is the number of seconds between frames. Having a counter which switches frames after a certain threshold is hit might solve this problem. This was done in this bachelor thesis, what can be done in the future of this project? A big thing is tracking the trajectory of the cells. Now that the visualization is done tracking the cells is a good next step. Doing this will even further improve the understanding of the effects of TLR2 and MyD88. The neutrophils in TLR2 and MyD88 mutants will behave differently, move at different speeds, or in different directions. Knowing how the cells behave differently in these mutants will elevate our understanding of these processes.

References

[Bar14] Jacques Deguine; Gregory M. Barton. Myd88: a central player in innate immune signaling. *National Library of Medicine*, 2014.

[BT13] Susana Santos Lopes Bárbara Taveres. The importance of zebrafish in biomedical research. *Acta Medica Portuguesa*, 2013.

[Bur11] R. L; Faires D. J. Burden. *Numerical Analysis*. Princeton University Press, 2011.

[Cha94] Tu Y; Euskirchen G; Ward W. W; Prasher D. C. Chalfie, M. Green fluorescent protein as a marker for gene expression. *Science*, 1994.

[Con95] I. N. Stevenson D. K; Contag C. H. Contag, P. R; Olomu. Bioluminescent indicators in living mammals. *Nature Medicine*, 1995.

[CV22] Chen Li; Wilson W.C. Yiu; Wanbin Hu; Lu Cao and Fons J. Verbeek. A feature weighted tracking method for 3d neutrophils in time-lapse microscopy. *IEEE Xplore*, 2022.

[Dag14] Takashi Hato; Pierre C. Dagher. How the innate immune system senses trouble and causes trouble. *Clinical Journal of the American Society of Nephrology*, 2014.

[dB01] Carl de Boor. *A Practical Guide to Splines*. Springer New York, NY, 2001.

[dB08] Mark de Berg. *Computational Geometry: Algorithms and Applications*. Springer Verlag, 2008.

[Des00] M. P. Desbrun, M; Cani. Implicit fairing of irregular meshes using diffusion and curvature flow. *SIGGRAPH*, 2000.

[ea07] Rowena Spence; Gabriele Gerlach; et al. The behaviour and ecology of the zebrafish, danio rerio. *Biological Reviews*, 2007.

[Fla07] William H. Press; Saul A. Teukolsky; William T. Vetterling; Brian P. Flannery. *Numerical Recepies*. Cambridge University Press, 2007.

[God] Godot engine website. https://godotengine.org/. Accessed: 2023-12-03.

[Gon18] R. E. Gonzalez, R.C; Woods. *Digital Image Processing*. Pearson, 2018.

[Ham11] H; Kawano H; Ando R; Shimogori T; Noda H; et al. Hama, H; Kurokawa. Scale: a chemical approach for fluorescence imaging and reconstruction of transparent mouse brain. *Nature Neuroscience*, 2011.

[Hop93] T; Duchamp T; McDonald J; Stuetzle W. Hoppe, H; DeRose. Surface reconstruction from unorganized points. *ACM SIGGRAPH Computer Graphics*, 1993.

[KMB06] H Kazhdan M; Bolitho, M; Hoppe. *Poisson Surface Reconstruction*. The Eurographics Association, 2006.

[Lev04] D Levin. Mesh-independent surface interpolation. In *Geometric Modeling for Scientific Visualization*, 2004.

[Lor87] H. E. Lorensen, W. E; Cline. Marching cubes: A high-resolution 3d surface construction algorithm. *ACM SIGGRAPH Computer Graphics*, 1987.

[Mad10] T. A; Sunkin S. M; Oh S. W; Zariwala H. A; Gu H; et al. Madisen, L; Zwingman. A robust and high-throughput cre reporting and characterization system for the whole mouse brain. *Nature Neuroscience*, 2010.

[Mar09] Zhang D. Saya H. Marumoto, T. Aurora-a - a guardian of poles. *Nature Reviews Cancer*, 2009.

[Neu] Nci dictionary of cancer terms. https://www.cancer.gov/publications/dictionaries/cancer-terms/def/neutrophil. Accessed: 2023-12-07.

[Nis21] Aga Syed Sameer; Saniya Nissa. Toll-like receptors (tlrs): Structure, functions, signaling, and role of their polymorphisms in colorectal cancer susceptibility. *National Library of Medicine*, 2021.

[NR21] Miyakwa T Nakajima R, Hagihara H. Similarities of developmental gene expression changes in the brain between human and experimental animals: rhesus monkey, mouse, zebrafish, and drosophila. *Molecular Brain*, 2021.

[Par15] David M Parichy. The natural history of model organisms: Advancing biology through a deeper understanding of zebrafish ecology and evolution, 2015.

[Paw06] James B. Pawley. *Handbook of Biological Confocal Microscopy*. Springer, 2006.

[R02] Janeway C. A. Jr; Medzhitov. R. Innate immune recognition, 2002.

[Tom02] R Tomasi, C; Manduchi. Bilateral filtering for gray and color images. *IEEE Xplore*, 2002.

[vSCLea21] Wanbin Hu; Leonie van Steijn; Chen Li; et al. A novel function of tlr2 and myd88 in the regulation of leukocyte cell migration behavior during wounding in zebrafish larvae, 2021.

[VTK] vtksmoothpolydatafilter class reference. https://vtk.org/doc/nightly/html/classvtkSmoothPolyDataFilter.html. Accessed: 2023-12-18.

[Xie19] S; Oskam J. M; Tonkens T; Meijer A. H; et al. Xie, Y; Tolmeijer. Glucocorticoids inhibit macrophage differentiation towards a pro-inflammatory phenotype upon wounding without affecting their migration. *Disease Models Mechanisms*, 2019.

[Yoo10] Q; Cavnar P. J; Wu Y. I; Hahn K. M; Huttenlocher A. Yoo, S. K; Deng. Differential regulation of protrusion and polarity by pi3k during neutrophil motility in live zebrafish. *Developmental Cell*, 2010.

[Zfi] Zebrafish genome compared to human: How are they similar. https://blog.biobide.com/zebrafish-genome-compared-to-human-how-are-they-similar#:~:text=They%20are%20vertebrates%2C%20however%20and,humans%20is%20actually%20quite%20similar. Accessed: 2023-10-18.