Bachelor's Thesis Computing Science

Radboud University Nijmegen

3D Fishualizer

Continuing a VR model of the brain recordings of zebrafish

Author:
Boudewijn van Gils
s1045276

First supervisor/assessor:
Dr. Bernhard Englitz

Second assessor:
Dr. Pieter Koopman

June 9, 2023

Abstract

The 3D Fishualizer is a project designed to visualise the brain of a zebrafish in Virtual Reality (VR). With the VR application, researchers can get increased insight in the brain data, which opens the way for exploratory and non hypothesis-driven research. The position, activity and corresponding regions of the roughly 80 000 neurons of the zebrafish are loaded from an hdf5 file and then shown as particles in the program Unity. From September to January 2023, the 3D Fishualizer has been extended with several features. The main changes are being able to select one of 294 regions, to show the activity of a neuron in a graph and to see if that activity is strongly related to the activity of other neurons by using correlation coefficients.

Contents

Chapter 1

Introduction

1.1 What is the 3D Fishualizer?

Let us start by untangling the name. It is called 'Fishualizer' because it 'visualizes' the brain activity of zebra'fish'. And '3D', because it is an application made for Virtual Reality. The 3D Fishualizer is made using the program Unity, which uses c# scripts. The goal of the application is to give researchers an immersive, insightful and engaging way to interact with zebrafish brain recordings. It should be easy and fun to use the model, and hopefully, doing so could lead to new hypotheses or improved understanding.

1.2 The data

If you are unfamiliar with zebrafish brain research and would like to know how the data is obtained, I would highly recommend reading "biology background knowledge" in the appendix (A.1).

The brain data for the 3D Fishualizer is stored in an hdf5 file. This is a commonly used format to store large amounts of scientific data. The hdf5 file includes the activity and positions of the neurons, as well as labels that indicate to which brain region a neuron belongs. The zebrafish larvae have roughly 80 000 till 100 000 neurons. The activity of these neurons is recorded over 3520 timesteps (at least for subject_02, which is the default file used in this project). The neurons can be grouped into one of 6 main regions, which can be further subdivided into 294 subregions.

This is a lot of data to work with in a simulation. Especially calculations with regards to ≈80 000 neurons can be quite computationally heavy and cause lag, which is unpleasant for the user. Therefore, it is important to limit the workload for the CPU/GPU of the computer on which the simulation runs, to make it run smoothly.

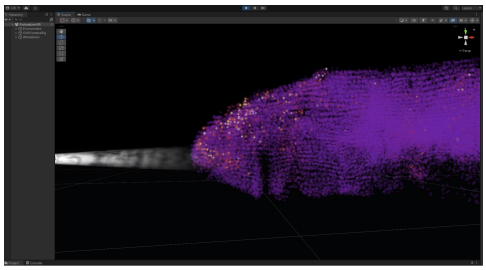


Figure 1.1: The 3D Fishualizer in Unity

1.3 The model

The work in this paper is a continuation of the work that Jannika Koschnitzke has already done in 2021-2022. As a master student of psychology, she developed the core of the 3D Fishualizer. In figure 1, you see that the activity of the brain is shown clearly and quite beautifully in Unity. The basic model of showing the brain activity is excellent and has therefore been left mostly untouched in the new implementation.

This paper covers the changes to the 3D Fishualizer from September to January 2022-2023. The first goal is to make the 3D Fishualizer run on the headset. After that, virtual hands should be added. Next, by using these new hands, the user should be able to select a region of the brain which is made clearly visible, whereas the rest of the brain becomes more transparent. After that, we will look into other functionalities and ways to interact with the brain. The functionalities do not need to be completely finished, as the 3D Fishualizer is still a work in progress. A fellow student, Utku, will help in continuing the project in the spring semester.

1.4 The new features of the 3D Fishualizer

These are the changes that I have made to the 3D Fishualizer in the autumn semester of 2022, described briefly (and shown in figure 1.2)

1. The "region mode". Let's say the user is interested in taking a close look at a part of the brain and nothing else. This is where they can activate the region mode. With virtual hands, the user can select a region of the brain. That selected region becomes very clear, whereas the rest of the brain turns nearly invisible.

   In the normal region mode the user can select from 6 large brain regions. In the 'subregion mode' the user can be even more specific, and select from 294 small regions.

2. A graph with neuron activity. If the user is interested in seeing the activity of a neuron over time, he or she can select an individual neuron.

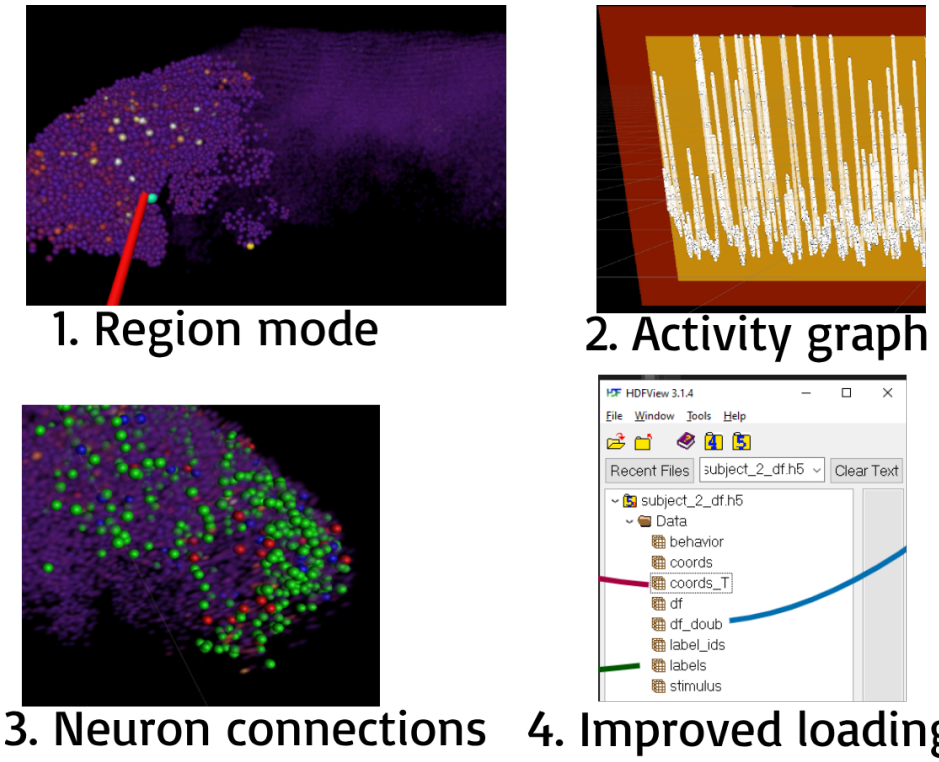


Figure 1.2: The four main changes covered in this paper

The complete activity of that neuron will then show up in a graph next to the brain.

3. Additionally, if the user is interested in how the activity of that neuron is related to the activity of other neurons, they can enter the 'neuron connections' mode. Other neurons that have a similar activity are colored red. Neurons that have a somewhat opposite activity are colored blue. With this function, the user can recognize clusters of neurons that are related to each other.

4. We would like the 3D Fishualizer to work on any zebrafish recording. That allows the user to pick a recording in which they are interested. A recording contains data for the position of the neurons, the activity of the neurons, and the 'region labels' of the neurons. Before, the 3D Fishualizer only worked on a single recording, which had altered datasets. The altered datasets did not exist for any other recordings, so the 3D Fishualizer could not work on them. Besides that, there was no way to load the 'region labels' yet. Both of these limitations have been overcome. The 3D Fishualizer can now load data from any recording, including the 'region labels'.

5. Besides those big functionalities, quite a few optimizations have been made to improve the 3D Fishualizer. The texture of the neurons has been improved, to make them appear less blurry. The brain can be rotated, such that you can view it from all angles. Finally, scaling the brain to a different size no longer moves the brain, which was a bit of a nuisance.

1.5 Overview of sections

If c# scripts are related to a part of this paper, they are placed in brackets behind the name of the section.

Chapter 2 gives a tiny introduction to Unity.

Chapter 3 looks into research about VR brain models, as well as the predecessor to the 3D Fishualizer.

Chapter 4 goes in depth on how the model works, what changes have been made and the process behind making those changes. In this chapter: Section 4.1 describes the model before any changes. This is the 3D Fishualizer as implemented by Jannika Koschnitze. Section 4.2 gives a preview to the controls of the new Fishualizer. Sections 4.3-4.8 describe how the model has been improved and extended in several ways:

3. Rotation and improved scaling of the brain are added. [RescaleByHand.cs]

4. Virtual hands have been implemented[PointerFollowsHand.cs]. Those are used to fully implement the Region Mode, in which brain regions are highlighted. [RegionMode.cs]

5. The activity of a single neuron can be shown in a graph. [ActivityGraph.cs]

6. The data is loaded in a more general way. [VisualiseNeurons.cs & RegionMode.cs]

7. Correlation coefficients between neurons can be shown in a basic way. [NeuronConnections.cs]

8. Unity uses a different version and the particles have a new texture.

Chapter 5 concludes all the changes made and discusses what can be improved in future versions of the 3D Fishualizer.

The Appendix provides extra information about certain topics.

Chapter 2

Preliminaries

2.1 Small introduction to Unity

The 3D Fishualizer is made in Unity. Unity is a developer tool, which is mainly used for creating video games but can also be used for an interactive Virtual Reality (VR) simulation. The main screen for developing an app in Unity is the Unity Editor. Here you can make GameObjects, which form the foundation of what is visible in the simulation. e.g. The brain of the zebrafish is a single GameObject in Unity called "WholeBrain". By using c# scripts these GameObjects can be given instructions such that they change on certain conditions, or when the user interacts with them. The c# scripts normally use a start() function and an update() function. The start() functions runs when you first load the scene. The update() function runs every frame.

Chapter 3

Related Work

3.1 Brain visualization

Presumably, the 3D Fishualizer is the first attempt to create a Virtual Reality (VR) application to show the Zebrafish brain. But the human brain has been modeled in VR before. A prominent example of such a VR model for the human brain is described by Pester in a recent paper[6]. This paper covers a VR application that aims to provide a better insight into the connectivity in the human brain. The 3D Fishualizer has the exact same goal, but for the zebrafish brain. The paper is very positive about being able to achieve this goal. To quote:

"*For our user study, we included experts from the field of EEG data analysis as well as computer scientists. The results yield positive feedback for exploratory data analysis especially in the use case of non hypothesis-driven research. Participants liked the overall experience and experts with background in brain activity analysis think it would be helpful using this application in a professional context. This indicates that an immersive 3D view of anatomically arranged brain offers a support for a data-driven,intuitive exploration of temporally varying, multi-dimensional brain networks.*"[6]

Other studies also point out the challenges of designing VR brain interaction [5]. It is not always clear which features are useful and effective and which are not. This makes the design difficult and requires the use of trial and error, to see whether the design works in practice. Furthermore, there needs to be a consideration whether specific data sets are computed before running the simulation, or during the simulation. This is especially challenging when using Unity, as it is not optimized for performing large computations.

The zebrafish brain itself has also been mapped out several times. One of these maps called mapZebrain can be found online[10]. This map gives outlines of the regions and subregions of the zebrafish brain, which light up when selected. This clear visualization is a good example of what a region

mode could look like for the 3D Fishualizer.

However, the most important precursor to the 3D Fishualizer is the original Fishualizer which was developed for desktop use.

3.2 The original Fishualizer

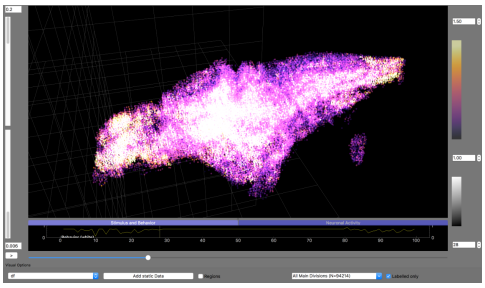


Figure 3.1: The original Fishualizer for desktop

This Fishualizer was made in Python by Thijs van der Plas, Rémi Proville and Bernhard Englitz. The model also takes an hdf5 file as an input and shows the neuron activity using colours (see figure 3.1). There are several ways to analyze the data, like showing the regions and deconvoluting spikes in the activity. Because it uses the same recordings as input, this original Fishualizer can be used as a benchmark to check whether new features in the 3D Fishualizer are working correctly. For now, we will focus on the 3D Fishualizer.

Chapter 4

Research

4.1 The 3D Fishualizer by Jannika Koschnitzke [VisualiseNeurons.cs & RescaleByHand.cs]

Neurons as Particles in Unity

The most important part of the project is the GameObject "wholebrain". This GameObject is a particle system. Choosing a particle system was an excellent choice for representing the brain, because it allows you to show ≈80 000 neurons, without having to create individual GameObjects for every neuron. Unity is not optimized for creating scenes with such large numbers of GameObjects, so creating and running a project with ≈80 000 GameObjects causes severe lag. The downside of using particles instead of GameObjects is that the resolution of the individual particles is not very high. This makes them unpleasant to look at from up close. Unity renders the particles using a billboard. A billboard makes a 2D sprite – in this case a circle – appear in 3D. Because of the low resolution of the sprites, billboards allow for a good performance at the cost of resolution. As far as we know, this is the best way to represent the brain in Unity.

Showing the activity

There is no clear hierarchy in the scripts. Everything could be written in a single script, but for clarity and accessibility of the code, separate files are used.

Jannika's project had two important c# scripts:'VisualiseNeurons.cs' and 'RescaleByHand.cs'. Those scripts still exist and contain most of the old code albeit now integrated into a larger whole. [1]

[1] There were a few other scripts (luminosity.cs, shader.cs) as well as a flashlight object in the Unity environment. I spoke to Dr. Englitz and Jannika and we concluded that those are not necessary and were related to some other ideas which we will not use anymore.

The script 'VisualiseNeurons.cs' performs the most essential tasks of the project. First it loads the coordinates and activity of the neurons from the hdf5 file. Then:

1. The coordinates are used to place the particles/neurons at the right positions to form the brain.

2. The activity is used to change the color of those particles, such that a bright and white color corresponds to a highly active neuron, and a dim and lavender particle corresponds to a lowly active neuron. This is implemented by using a colormap and a clever function that makes the color change smoothly.

In 'RescaleByHand.cs', the brain GameObject is transformed by controller inputs. There are two transformations possible, if the right buttons are pressed:

1. Moving: Changing the location of the brain by moving the right controller in the direction you want the brain to go. This transformation feels quite natural to perform.

2. Scaling: Increasing/decreasing the size of the brain, by moving the controllers outwards or inwards.

Because all the neurons are part of the brain GameObject, they will move/scale when the GameObject is moved/scaled.

4.1.1 Region mode code

Jannika also made the code for highlighting certain neurons in the brain by making all the other neurons a lot more transparent. However, because she had not found a way to create virtual hands, with which to select a region of the brain, this part of the code was never used. But the code was later revised into 'Regionmode.cs".

A major issue at the start of working on the project was that nothing showed up on the Oculus Rift. The application ran perfectly fine on the computer, but when putting on the headset, nothing showed up. Resolving this issue took a lot of time and effort, as well as some help from outside.

This is detailed in the appendix under "Making the Oculus Rift show the brain again" (A.2).

4.2 Controls of the new 3D Fishualizer

The controls for all functionalities of the 3D Fishualizer are shown in figure 4.1. The boxes show which scripts direct which actions. This image is meant to give an early overview to be used for reference. We will now take a closer look at the features.

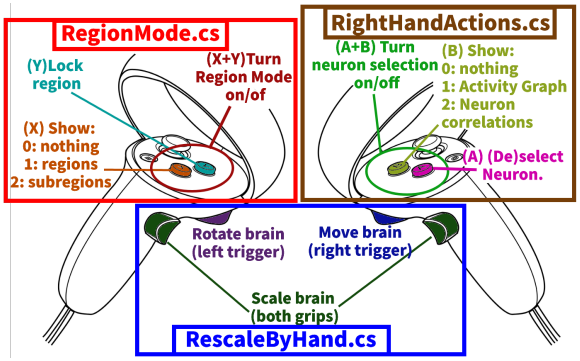


Figure 4.1: The controls for the 3D Fishualizer at the end of January [8]

4.3 Hands and brain rotation [RescaleByHand.cs & PointerFollowsHand.cs]

Adding virtual hands is a good starting point for extending the 3D Fishualizer because they enable the user to select regions or neurons of the brain. This is useful for implementing other functionalities. There are several options when making hands in Unity:

1. Humanlike hands: As an advantage, these look natural for the user. You can also use these hands for animations like grabbing or pointing with the index finger. But hand gestures are not very important for the 3D Fishualizer. Because the virtual brain is quite large, something longer than a humanlike hand is desirable.

2. Raycast hands: These are similar to a laser pointer. They are useful for grabbing from far away or pointing at objects. Pointing is important in the 3D Fishualizer. However, objects need to be solid for the raycast laser to interact with them. Therefore, you can't use the raycast hands to point to the middle of the brain.

3. Object hands: These hands are made from Unity GameObjects, like a cylinder, for example. With such basic shapes you can create a pointer with which the user can easily point to the brain, including right in the middle of it. At the end of the pointer is a little sphere. The position of this sphere can be used to find the position of the nearest neuron/region. The disadvantage of using the object hands is that they don't look very appealing. Perhaps, in a later stage the basic shapes which make up the hands can be changed for a more complex and beautiful pointer. For now the basic shapes will do.

4.3.1 Implementing the object hands

To create the object hands, you can start by implementing basic raycast hands by following this Youtube tutorial [12]. When you finish the tutorial, you will have two raycast hands. To turn them into object hands, you create an object in Unity from basic shapes (2 spheres and a cylinder). Then by applying the script PointerFollowingHands.cs on these hands, they start following the VR controllers. In the script, the hands are set at a 90 degrees rotation on the x axis, such that they naturally extend from the user's arm instead of going upward.

4.3.2 Implementing rotation

The left trigger button is used for rotating the brain. This is implemented such that if you rotate the left controller, the brain rotates in the same way as the left controller. Candidly, this is not the most intuitive way of doing a rotation. There are two ways of rotation which are probably more intuitive:

1. A rotation in which you use both hands to rotate the brain, using a motion much like how you would rotate a plate of food in real life.

2. A rotation in which you spin the brain with one hand. This is similar to how you can spin the earth in Google Earth.

However, because rotating is not nearly as important as moving and scaling, I haven't taken the time to implement one of the better rotations. The simple rotation is sufficient for now.

4.3.3 No centre point

At this point, the rotating and scaling worked, but they both had a defect. The defect is most easily shown in a picture (see figure 4.2).

Because the neurons of the activated brain did not appear at the centre of the brain GameObject, the brain would move a lot when scaling or rotating. In short, the neurons need a centre point around which they rotate or scale. At first I solved this issue by creating an artificial centre for the brain. The artificial centre, for which I used a green sphere was the parent object of the brain object. Therefore, when the green sphere moves/rotates/scales, the brain does exactly the same. This solution is illustrated in the appendix under "Green sphere solution for rotating/scaling" (A.3).

Eventually, this solution became obsolete because of a better solution: moving the active neurons such that the center of them is the exact center of the brain object. The details for this solution are also in the appendix under "Moving the neurons solution [VisualiseNeurons.cs]" (A.4).

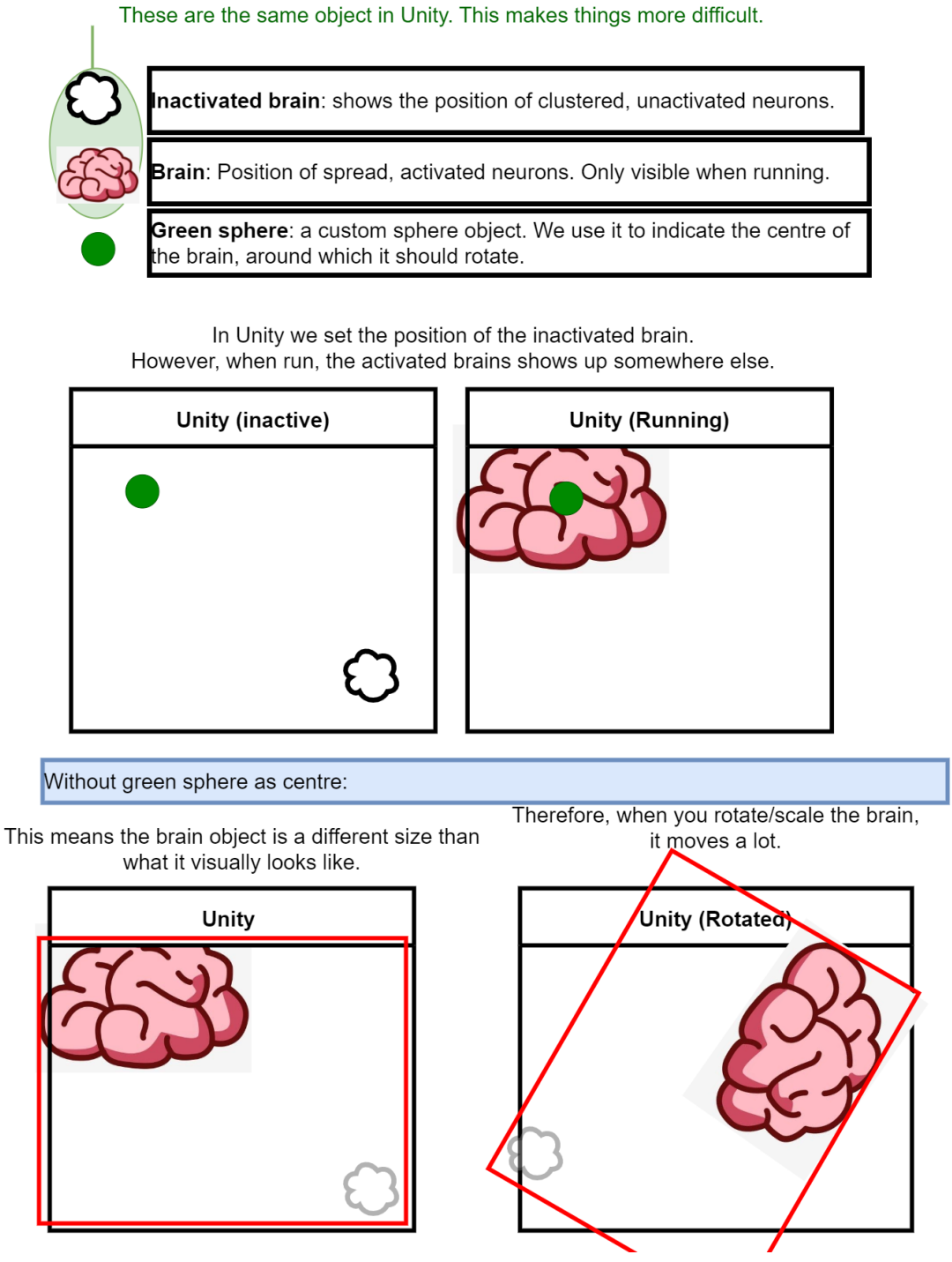


Figure 4.2: Issue with rotating/scaling the activated brain

4.4 Showing brain regions [RegionMode.cs]

4.4.1 How the basic region mode works

The basic idea of the region mode was already worked out by Jannika. Using the controller to aim, the user points somewhere in the brain with the virtual pointer. Then, the neuron whose position is closest to the end of the pointer is selected. The selected neuron is passed into a dictionary as the key. This dictionary links the neuron to the region to which it belongs. Once we have the selected region, the next step is quite simple: All neurons that are not part of the region are made more transparent. This is done by using the same neuron-to-region dictionary, but in the other way around (i.e. loop through all neurons and select those which are not part of the region). In figure 4.3 you can see an example of what the region mode looks like.

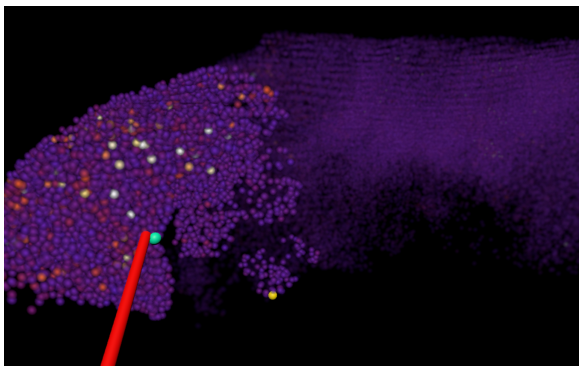


Figure 4.3: The left pointer is red while in region mode. The neuron closest to the end of the pointer is indicated with a blue sphere. The region of that neuron lights up.

4.4.2 Updating the neuron positions

To select a region, we need the position of the closest neuron to the pointer. This means we need to store the positions of all ≈80 000 neurons. The position of the neurons needs to be up to date for the region mode to work. Otherwise, the region mode will still use the old positions, from before the brain was moved/rotated/scaled.

If you wish to keep these positions up to date by calculating them in the update function – which is called every frame – it causes heavy lag. Therefore, RegionMode makes use of a boolean 'brainChanged'. 'brainChanged' is set to true after the brain object has been moved/rotated/scaled. If the boolean is true, the positions of the neurons are recalculated. Otherwise, using the old positions is fine.

Getting the updated positions is done by taking the original position of

the neuron, and then adjusting it for the new position, rotation and scale of the brain object [2]. If the calculation is only done once, the lag it causes is insignificant such that you do not notice it.

4.4.3 Controls: SwitchOnButton

The SwitchOnButton() function is difficult to understand, despite its small size, which is why I will explain it here: We want to turn the region mode on and off by pressing a single button. Implementing this in Unity is not very straightforward. By default, Unity can only detect whether a button is pressed or not. Therefore, if you use a simple implementation, holding the button causes the region mode to be turned on/off every frame, which is very inconvenient. In our implementation we would like to change the region mode only once when the button is pressed continuously. To do this, we use a boolean 'timeToChange'. After the first frame a button is pressed, 'timeToChange' is set to false. This prevents the region mode from changing again when the button is continously pressed. Only after the button has been released, 'timeToChange' becomes true again.

4.4.4 Color changes and locking

Once the region mode is activated, the left pointer becomes red, to give a clear visual indication that it is turned on. The region which is currently selected shows up as text in Unity. The text is displayed on the activity graph, which we will cover later.

Locking regions or subregions is a functionality which is semi-implemented. It is possible to lock one region or subregion, such that it remains visible when you stop hovering it with the left pointer. This is convenient when you want to observe a region without putting in the effort of hovering over it with the pointer. In the future, it might be desirable to extend this such that you can do this for multiple regions/subregions.

4.5 Showing neuron activity in a graph [ActivityGraph.cs]

4.5.1 Creating a graph

Next, we wanted to create a graph which shows the activity of a neuron over time. You can select a neuron with your right hand, and then see in the

[2] A pitfall, which both me and Jannika fell into is thinking that the "world position" of the neurons would refer to their absolute position. In Unity, "world position" stands in contrast to the "local position" which is the position of something in regards to their parent object. However, the world position of the neurons does not give you the correct position where they are, as you still need to (manually) adjust for the position/rotation/scale of the brain object to which they belong.

graph how active it is over all (in this case 3520) timesteps. The creation of a graph was done mainly by following a Youtube tutorial [11]. The graph is made in a basic and low level way: by creating circle- and rectangle objects at the right positions on a canvas. The positions are calculated from the neurons activity. Figure 4.4 shows an example of what the activity graph looks like.

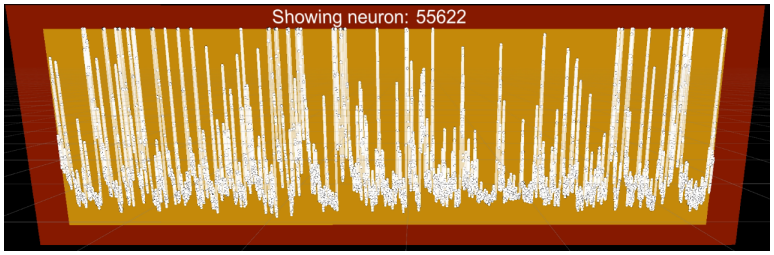


Figure 4.4: The activity of neuron 55622 is shown over 3520 timesteps in the graph.

4.5.2 Visualising the activity of a neuron

The other scripts only need to know a single function from the ActivityGraph.cs script, which is ShowGraph(int selected_neuron).

If this function is called, the rectangles and circles which made up the old graph are deleted. Then the new positions are calculated. Finally, new rectangles and circles are placed at those positions. Creating a large amount of objects causes some lag. This is reduced by the function ChangeChildrenOrder(), which groups the rectangles and circles in the Unity hierarchy. When similar unity objects are next to each other in the hierarchy, they can be rendered in batches. This reduces the lag of loading them all at once.

This process seems to take about half a second of computing in Unity, such that the simulation pauses for a short time. Possibly, the rectangle- and circle objects can be reused when selecting a new neuron, such that less objects need to be created and destroyed. For now, this implementation of the activity graph is simple and quite functional.

4.6 Loading the data

In the project, we use a large database that contains the position, activity and region labels of the zebrafish neurons. This data is stored in an hdf5 file. This filetype is often used for storing large amounts of scientific data. Unity is not build for loading large amounts of data, so it is difficult to do so in c#. This was done in the original implementation by using the library SharpData. This is a custom made library made by Teun van Gils,

who works in Nijmegen in a related field. From the SharpData library, two custom made functions were used to load the positions and activity of the neurons from the hdf5 file. These worked very well, such that the loading the was done in a few miliseconds. However, there were two issues remaining:

1. subject_2_df.h5 –The file which we used for testing the 3D Fishualizer – contains two extra datasets: 'coords_T' and 'df_doub'. These datasets do not occur in any of the other hdf5 files, which only contain the unaltered versions of those datasets: 'coords' and 'df'. (You can see the datasets listed in the white part in figure 4.5). The activity and positions of the neurons are loaded by using the special datasets which only one file had. Therefore the 3D Fishualizer will not work on any of the other files.

2. The region label data is not loaded in. The activity and positions are loaded with custom made functions (as shown in the black parts of figure 4.5), but no such function exists for the region labels.

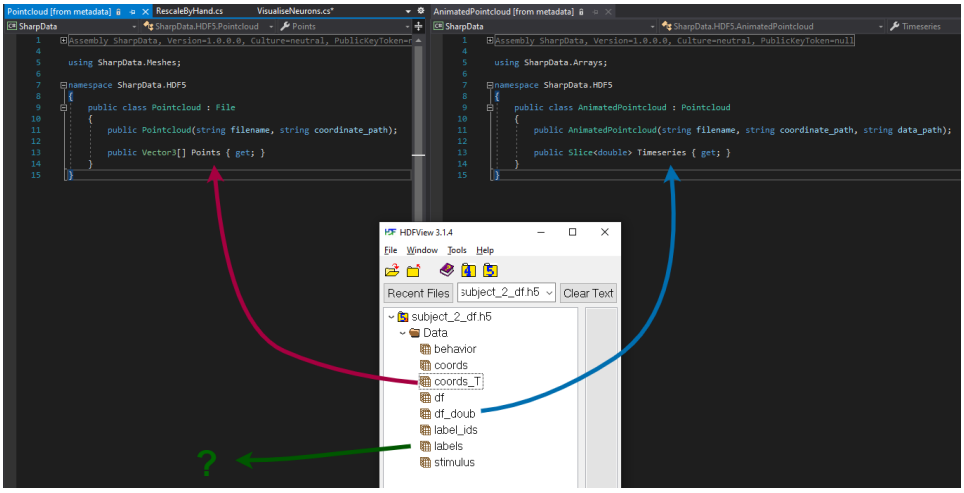


Figure 4.5: Top left and right: The get functions from the Sharpdata library for the positions and activity, respectively. Bottom middle in white: The datasets of subject_2_df.h5.

4.6.1 Removing dependency on the two extra datasets

We want to load the activity and positions of the neurons from the data sets that are in every hdf5 file. Specifically, the activity has to be loaded from df instead of df_doub. df and df_doub are very similar. The only difference between them is that df_doub stores the activity as doubles, whereas df stores it as floats. So to get the data from df, we use the custom made function that

loads df_doub and change Slice<double> to Slice<float>. This causes some minor changes in the rest of the code. Previously, we used a normalizing function from the SharpData library. This function does not work anymore with floats. Therefore, we write our own normalizing function for floats and then everything works as before.

Now for the neuron positions, we want to rewrite the function that gets coords_T such that it works for getting coords instead. coords_T is the same dataset as coords, but transposed. The rewriting is not so easy, mainly because we do not have direct access to the SharpData library. The only access we have is by using DotPeek, which decompiles the library into readable c# code (see figure 4.6). This decompiled code has no comments and many variables have unusual types, which makes them difficult to work with.

Luckily, Teun van Gils dropped by and helped out. With his help, the function is now rewritten to work for coords. Now both the positions and the activity could be loaded from any hdf5 file.

```
public Pointcloud(string filename, string coordinate_path)
  : base(filename)
{
  this.Points = this.ReadData<double, Axis2D>(coordinate_path).AggregateSpace<Vector3, Axis1D, Axis1D>((Func<IData<double, Axis1D>, Vector3>) (data => new Vector3()
  {
    X = (float) ((IMutableList<double>) data)[0],
    Y = (float) ((IMutableList<double>) data)[1],
    Z = (float) ((IMutableList<double>) data)[2]
  }), new SortedSet<Axis2D>() { Axis2D.Y }).ToArray();
}
```

Figure 4.6: The decompiled SharpData library in DotPeek. This is the function for acquiring the data from coords_T

4.6.2 Loading the label data

Teun also helped in loading the region label data.

First let us cover how the label data is stored. Most of the data is stored in the hdf5, with exception of the text file RegionLabels.txt. This textfile stores the names of the 294 subregions, all separated by new lines. This makes it such that we can simply refer to the regions using an index number, and then use this text file to get the right name for it.

The neuron-to-region data is stored in a ≈80 000 by 294 sized matrix in the hdf5 file. The 294 columns correspond to the 294 subregions in the zebrafish brain and the ≈80 000 rows correspond to the neurons. Every neuron is assigned exactly one of these regions, which means that column will contain a 1, and all the other columns are 0. (As an exception: There are a few neurons that are assigned no region at all.) With the help of Teun, we created a function to assign each neuron their respective region. The function navigates this matrix in a very efficient way . The basic idea is: For every row, find the first index which is 1 and return that as the region to which the neuron belongs. Nevertheless, even with using this efficient algorithm, loading the labels still takes about 30 seconds. In all likelihood, there is no way to load the data from such a huge matrix into c# in a fast

way. To circumvent this issue, we only load the data from the hdf5 file the first time that the 3D Fishualizer runs. In that first run, we write the regions of the ≈80 000 neurons to a text file. The text file has ≈80 000 lines and each line contains a number which corresponds to the region of the respective neuron. On subsequent runs of the 3D Fishualizer, the data is loaded from this text file, instead of the huge matrix from the hdf5 file. Reading from the text file only takes 0.016 seconds, which is negligible compared to 30 seconds of loading from the hdf5 file. Accepting this small circumvention, that means all loading issues are solved.

4.6.3 Why use the sparse matrices?

The previous issue was caused by two factors. One, c# is not optimized for handling large amounts of data. Two, the label data is stored in an enormous matrix. It would only make sense to use such a matrix if some neurons were assigned to multiple regions. But if you analyze the label data for the file which we used – subject_2_df.h5 – you will find that the neurons are never assigned to more than one region. That means having the sparse matrix has no added value and just makes the loading more difficult. If this is the same for the other hdf5 files, a systematic change in storing the label data is desirable.

4.7 Showing Correlation Coefficients between neurons [NeuronConnections.cs]

For the last added functionality of the 3D Fishualizer, we want to user to be able to select a neuron and see which other neurons it is closely related to. To do that, we compare the activity of the neurons. Concretely, the correlation coefficients between the activity of the selected neuron and the activities of the other neurons are calculated. This is then shown by coloring the other neurons such that:

1. Green neurons show no significant correlation with the selected neuron.
2. Red neurons show a strong positive correlation with the selected neuron.
3. Blue neurons show a strong negative correlation with the selected neuron.

To color a neuron, a sphere GameObject with the appropiate color is created on top of the neuron. An example of such a correlation coefficient is shown in figure 4.7. The spheres have the brain GameObject as their parent, such that they move along with it.

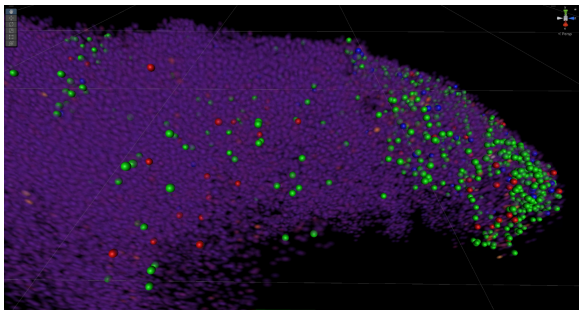


Figure 4.7: Visualization of correlation coefficients of neuron 55622

4.7.1 Calculating/reading Correlation coefficients

If we want to have this functionality for all ≈80 000 neurons that would mean that ≈80 000 correlation coefficients need to be computed every time a new neuron is selected. Alternatively, you could calculate all possible correlation coefficients beforehand, resulting in an ≈80 000 by ≈80 000 matrix but no more calculations during runtime. Unfortunately, calculating correlation coefficents in c# takes a very long time. Even after some small optimizations, calculating correlation coefficients for for only 1000 neurons still takes 56 seconds. This puts some heavy restraints on the possibilities of showing the neuron connectivity using c#. For now, we choose to make this functionality only for the 1000 most active neurons. To reduce loading times, we apply a very similar circumvention as we did for loading the region data. By storing the correlation coefficients in a text file after calculating, subsequents runs don't need to do the computations. This reduces the time to acquire the correlation coefficents from 56 seconds to 0.52 seconds.

4.7.2 Selecting a neuron [RightHandActions.cs]

Similar to ActivityGraph.cs, other scripts only interact with NeuronConnections.cs by calling the function ShowCorCoefs(int selected_neuron). This function calculates the colors for the newly selected neurons, and then replaces the spheres such that they have the right colors. Both showing the activity graph as well as showing the correlation coefficients is controlled in the script RightHandActions.cs. It enables the user to press certain buttons on the right controller with which they can:

- Select a neuron.
- Show the activity for the selected neuron in the graph.
- Show the neuron correlations for the selected neuron.

The controls for doing this work fine, but are not very user friendly. They are now set up more from a developer perspective: To test all functionalities

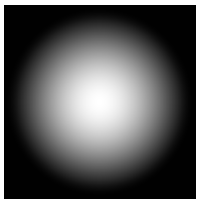
separately. This helps the developer, for example, to see what features cause lag. But in a future version the controls should be simplified for normal users, which should not take too much effort.

4.8 Changes in Unity, non-script related

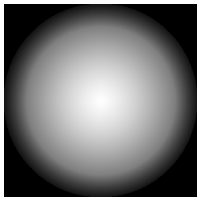
That covers all the major changes to the 3D Fishualizer. I would like to point out some other minor changes:

1. The Unity version is changed from Unity 2021.2.8f1 to Unity 2021.3.11f1. The latter one is a last Long Term Support (LTS) version of Unity. Hopefully, using an LTS version will provide a stable environment for developing the 3D Fishualizer.

2. The brain particles have a more detailed texture. By default the brain particles use the "Default-Particle" texture (figure 4.8a). This material uses a 64 by 64 image of a circle. When looked on from up close, this is quite unpleasant to look at because of the low resolution. Therefore, I made a similar image using Photopea, but with 264 by 264 pixels (see figure 4.8b). This new texture has a higher resolution and a more steep gradient which makes it less blurry. This makes the particles a bit more pleasant to look at from up close, in my opinion. The change did not seem to affect the performance.



(a) Default 64x64 particle texture



(b) New 264x264 particle texture

Figure 4.8

Chapter 5

Conclusions

5.1 Conclusion

Overall, good progress has been made on the 3D Fishualizer. First, The project shows up on the headset again. Second, A diverse range of functionalities have been added: Scaling the brain no longer causes the brain to move and it can also be rotated.

Virtual hands are added which can be used to point to regions and subregions of the brain.

The right pointer can be used to select a single neuron. The activity of that neuron can be shown on a graph. If the selected neuron is one of the 1000 most active neurons, it's relation with the other 1000 most active neuron can be shown by calculating the correlation coefficients between their activities.

5.2 Discussion

The added functionalities provide users with diverse ways to interact with the zebrafish brain. However, the project can still be improved in many ways:

- The rotation of the brain GameObject can be made more intuitive.
- Improving the movement of the brain while in region mode. Right now, when you move the brain in region mode, the region mode is paused. Because of this pausing, the whole brain is visible while moving. This is not very intuitive, as the user probably wants to see only the selected region when moving the brain.
- The inactive neurons can be made more transparent. This would allow the user to see through the brain more easily.

- The controls can be made simpler. Perhaps creating an intuitive arrangement of controls is best done once it is decided which functionalities are going to be in the 3D Fishualizer and which are not. In any case, a framework for the controls can be made, such that it is easy to change them later.
- Showing the neuron connections can be made far more clear and beautiful. A first improvement would be to remove the green spheres, such that only the significant correlations are visible. This would declutter the view a lot. Other improvements, like including more than 1000 neurons, as well as connecting lines between the selected neuron and the other neurons can also be made.

There are many more possible improvements which are not in this list. There is also room for new functionalities. Through improvements and additions, we will hopefully get ever closer to an ideal 3D Fishualizer.

Bibliography

[1] Misha B. Ahrens et al. "Whole-brain functional imaging at cellular resolution using light-sheet microscopy". In: *Nature Methods* 10.5 (May 2013), pp. 413–420. ISSN: 1548-7105. DOI: 10.1038/nmeth.2434. URL: https://doi.org/10.1038/nmeth.2434.

[2] Jillian M Doyle and Roger P Croll. "A Critical Review of Zebrafish Models of Parkinson's Disease". In: *Frontiers in Pharmacology* 13 (2022). DOI: 10.3389/fphar.2022.835827.

[3] Jing Ying Hoo et al. "Zebrafish: a versatile animal model for fertility research". In: *BioMed research international* 2016 (2016). DOI: 10.1155/2016/9732780.

[4] Kerstin Howe, Matthew D Clark, et al. "The zebrafish reference genome sequence and its relationship to the human genome". In: *Nature* 496.7446 (2013), pp. 498–503. DOI: 10.1038/nature12111.

[5] Sabrina Jaeger, Karsten Klein, et al. "Challenges for brain data analysis in VR environments". In: *IEEE Pacific Visualization Symposium* 2019-April (Apr. 2019), pp. 42–46. ISSN: 21658773. DOI: 10.1109/PACIFICVIS.2019.00013.

[6] Britta Pester et al. "Understanding multi-modal brain network data: An immersive 3D visualization approach". In: *Computers & Graphics* 106 (Aug. 2022), pp. 88–97. ISSN: 0097-8493. DOI: 10.1016/J.CAG.2022.05.024.

[7] Tsegay Teame, Zhen Zhang, et al. "The use of zebrafish (Danio rerio) as biomedical models". In: *Animal Frontiers* 9 (3 June 2019), pp. 68–77. ISSN: 21606064. DOI: 10.1093/af/vfz020.

Image/tutorial sources

[8] *Controller image source*. Last accessed 30 March 2023. URL: https://uxdesign.cc/ux-navigation-patterns-in-vr-jumping-between-scenes-3a1fd8df0151.

[9] *Light Sheet Fluorescence Microscopy*. Last accessed 30 March 2023. URL: https://www.youtube.com/watch?v=afIkWHx3duc.

[10] *Max Plank zebrafish brain atlas*. Last accessed 30 March 2023. URL: https://mapzebrain.org/atlas/3d.

[11] *Unity tutorial - create a graph*. Last accessed 30 March 2023. June 2018. URL: https://www.youtube.com/watch?v=CmU5-v-v1Qo.

[12] *Unity VR Game Basics - PART 5 - XR Ray Interactor in 10 Minutes*. Last accessed 30 March 2023. Mar. 2022. URL: https://www.youtube.com/watch?v=iVfa_azjnNI.

[13] *Zebrafish*. Last accessed 30 March 2023. URL: https://en.wikipedia.org/wiki/Zebrafish#/media/File:Zebrafish_Developmental_Stages.tiff.

Appendix A

Appendix

A.1 biology background knowledge

A.1.1 Studying zebrafish

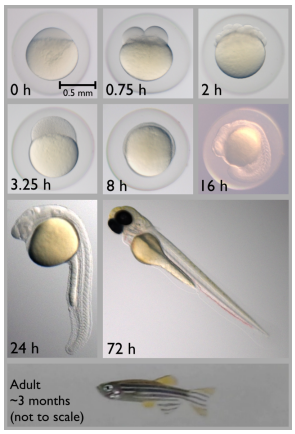
Zebrafish (Danio rerio) have become increasingly popular as a model animal for scientists in recent years[7]. They grow up to 4 centimetres, and have a similar body to humans, as they are both vertebrates. About 70% of the genes of a zebrafish have a human equivalent [4]. Zebrafish can also suffer from the same illnesses, including Parkinson's, Alzheimer's and cancer. [2]. When compared to rats or mice, zebrafish are cheap, easy to maintain and reproduce very quickly. Furthermore, they are oviparous, so the fertilized eggs develop outside of their mothers body, which makes them accessible for experiments from birth [3]. The main aim of most studies is: If the simple zebrafish can be understood better, then so can the more complex human.

A.1.2 Obtaining brain data from zebrafish

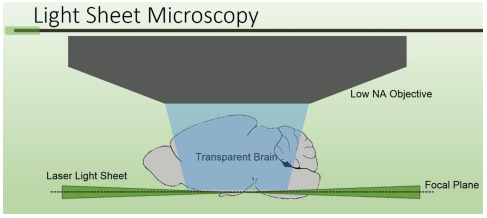
The brain experiments are performed on the zebrafish larvae (see the 72h fish in figure A.1a). The brain of a larva contains 80 000 till 100 000 neurons and is smaller than a cubic millimeter [1]. The larva's tiny body is mostly transparent, which allows for a special brain scanning technique to be used: Light-sheet fluorescence microscopy. This is a hefty term, but thankfully, it can be divided into three pieces:

1. Microscopy: Looking at something which cannot be seen with the naked eye.

2. Light-sheet: To see the neurons of the zebrafish, a laser sheet is shone horizontally through the brain (see figure A.1b). This sheet is very thin, which makes it precise enough to capture all neurons. By combining many 2D images, a 3D model of the brain can be made. The complete scan of the zebrafish brain takes about 1.3 seconds [1].



(a) Developmental stages of the zebrafish [13]



(b) Overview Light-Sheet microscopy [9]

Figure A.1

3. Fluorescence: The studied zebrafish is genetically modified. The modification makes it produce calcium-indicators in its neurons. These indicators become fluorescent when they bind with calcium ions. When a neuron is activated, calcium ions pour into the cell. These ions then bind with the indicators and increase the fluorescence of the cell. The fluorescence of the neuron can then be measured, which shows how active the neuron is.

Using light-sheet microscopy, it is possible to capture more than 80% of the neurons at single-cell resolution [1]. This is a very high resolution when compared to alternative methods such as fMRI, which can only detect clusters of neurons[1]. The data which we use for the 3D Fishualizer is collected in such a way.

A.2 Making the Oculus Rift show the brain again

A.2.1 Nothing shows up on the Oculus Rift

The issue was as follows: When running the Fishualizer 3D in the Unity editor, nothing showed up on the Oculus Rift. At first, we thought this was an issue with the Fishualizer. But the same issue occurred when trying to run a new empty Unity project. There was no error popup or loading screen on the headset. Pressing run didn't affect the headset in any way, even though it should. The headset seemed to be well connected, according to the Oculus app. The headset also seemed to be functioning properly when testing a tutorial on it (unrelated to the PC). Changing the Unity version and changing the packages also didn't resolve the issue. There was

no solution online which helped, so eventually we made a blog post to ask for help. But nobody helped resolving that blog post.

Then we figured it might work on a different computer. Unfortunately, the headset needs to be connected to a GPU with a DisplayPort cable. The other computers in the Computational Neuroscience Lab did not have a GPU with a DisplayPort input.

Then we tried uninstalling everything related to Unity and Oculus, including the Appdata files. After reinstalling the same error still occurred.

A.2.2 Help and Solution

At this point, no solution seemed in sight. I went to seek help at the Max Planck Institute nearby, as I knew they worked with VR there. There I got help from Paul van der Laan, who very generously took the time to help me out (Thank you Paul!). I brought the Oculus Rift, which worked fine on his computer. Therefore, the issue was not with the headset, but with the PC.

Also, I saw him build the program into an executable and then run it. I had tried this before on the 'malfunctioning' PC, but that still did not run on the headset. However, it turns out that if you install the module 'Windows Build Support (IL2CPP)' in Unity hub and then build an executable, it does run on the headset! This breakthrough allowed me to test the implemented hands.

A.2.3 Workaround

This workaround was still not ideal. Every time you wanted to test a change on the headset you had to build an executable and run it from there (which takes about half a minute). Also, running the executable does not give you any debugging information which you might have put in the code. This slowed down working on the project significantly. Furthermore, sometimes even the executable would not work and the headset would show a red light. Unplugging and replugging the headset seemed to resolve this red-light issue most of the time, but not always.

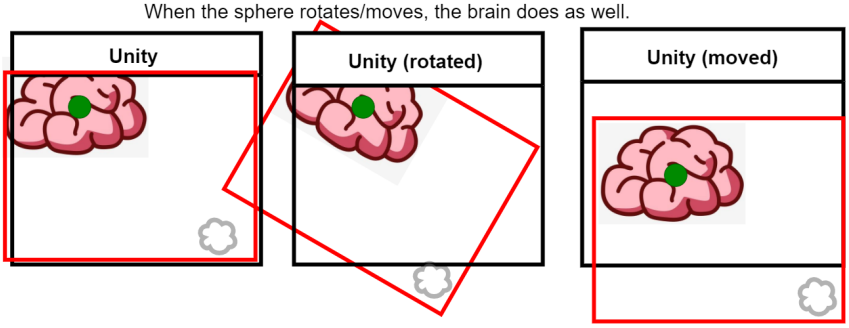
But on the positive side, eventually the program would sometimes run on the headset, even without making an executable. This was very useful for getting debug information and for testing quicker.

All in all, it is probably better to develop the 3D Fishualizer on a different computer, as developing it on this PC has been a headache sometimes.

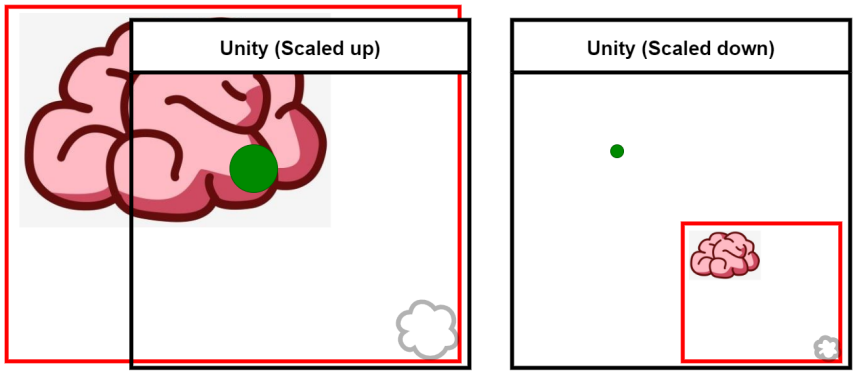
A.3 Green sphere solution for rotating/scaling

With green sphere as centre:

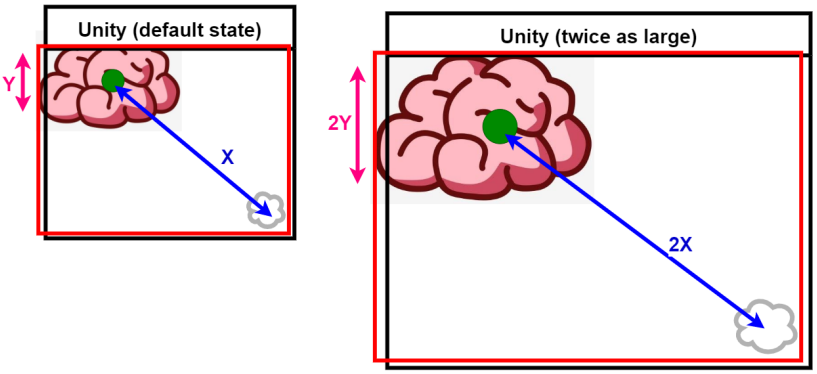
To prevent this, we make the green sphere the parent object of the brain.
When the sphere rotates/moves, the brain does as well.



This works well. However, a new problem arises with the scaling as can be seen below.
When the brain and the sphere scale together, the brain moves away from the sphere.



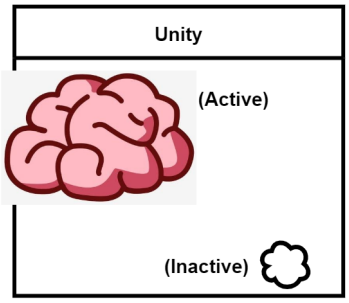
Therefore, while the scaling happens, we compensate for this effect by moving the brain simultaneously.
If the brain gets twice as large, the distance from the green sphere to the inactivated brain position gets twice as large.



Now, moving, rotating and scaling the brain are fully functional!
The green centre is made invisible from now on.

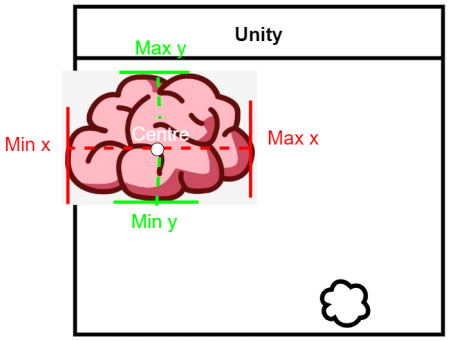
A.4 Moving the neurons solution [VisualiseNeurons.cs]

By default, the brain moves to a different location when it is activated.



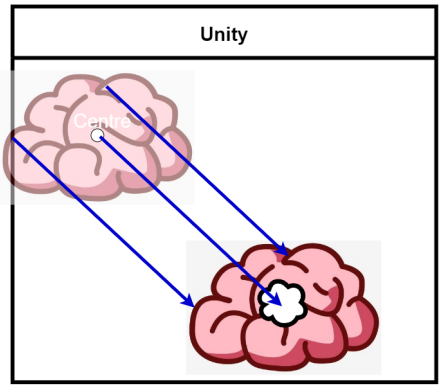
This makes it very difficult to access the position of the individual neurons, as they are not on top of their Unity GameObject, which is the Inactive brain.

Therefore, we calculate the highest and lowest, x, y and z positions for all the neurons.



With these values, you can get the centre position of the brain.

You can then move all neurons such that the centre of the active brain is exactly on top of the inactive brain.



Now there is no difference anymore between the position of the active brain and the inactive brain. This makes rotating/scaling as well as getting the positions of individual neurons much easier.