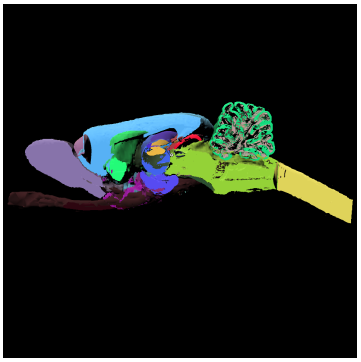
NTNU – Trondheim
Norwegian University of
Science and Technology



Towards a 3D model of the rat brain in AR as an educational tool

Ole V. Ravna

Autumn 2020

PROJECT REPORT
TDT4501 - Specialization Project
Department of Computer Science
Norwegian University of Science and Technology

Supervisor 1: Gabriel Kiss

Supervisor 2: Ekaterina Prasolova-Førland

Abstract

This study aims to explore whether Augmented Reality can be used as a tool for medical students learning neuroanatomy. A application, *Nevrolens*, was created with features simulating a conventional rat brain dissection. While the results from this study are limited, the application has shown promise in its use as a virtual simulation of a rat brain dissection.

Contents

Chapter 1

Introduction

1.1 Motivation

Augmented reality is a technology which has experienced great leaps in recent years, and this growth has inspired many visons of medical potentials for this young technology. Within medical education there are many fields where visual understanding is critical, one of being neuroanatomy. Neuroanatomy is a highly complex domain both visually and spatially, the ability to use the human senses in a real-world setting could result in greater intuition and understanding. With that in mind the use of augmented reality could be a natural way to virtualize the experience of a brain dissection, and further the unique capabilities of AR could enable innovative ways of learning. (Moro et al., 2017) shows the possibility of greater immersion and engagement while using augmented reality in teaching anatomy to medical students. This has also recently been shown with promising result by (Wish-Baratz et al., 2020), where COVID-lockdown required from-home teaching, and the use of HoloAnatomy, an anatomy application for the HoloLens, performed significantly better than even conventional in-class lectures.

1.2 Problem Formulation

The main problem with most academic implementations, like (Wish-Baratz et al., 2020), of AR in medical education is the use of head-mounted display (HMD) devices like the HoloLens 2 and Magic Leap, which in the near to mid-term future will have limited practical use in education, as a result of the high price-tag, combined with the still inadequate general use-case for these types of devices. This project will try to mend these challenges by having the lecturer using an HMD and having student view and interact with the lecture

in an AR-based application running on their smartphone. This is possible because of the great leap in AR-performance seen in recent models of Android and especially iPhones, in combination with development platforms like Unity, Mixed Reality Toolkit and **??** which enables multiplatform development and real-time collaboration between devices. The aim of the project will be to create a seamless educational experience in augmented reality which can be valuable both on an HMD device and a modern smartphone. The focus will be on investigating its feasibility as an educational tool both in a lecture-type setting and for students to explore the brain anatomy independently.

1.3 Research Questions

What follow are the research questions which motivates this project:

Main-RQ: How can AR support teaching of neuroanatomy and dissection for medical students?

Sub-RQ1: How should interaction in be implemented in AR to accommodate medical students and educators?

Sub-RQ2: How will a collaborative experience shared between an HMD and a smartphone compare to accommodate medical users?

Sub-RQ3: Can understanding be increased by integrating microscopical data into a macroscopical model?

1.4 Approach

The research questions were derived through discussing the needs of the intended users with neuroscientists at the Kavli Institute. It was then narrowed down by a literature review, finding a lack of satisfactory substitutions for real brain dissections and especially finding no attempt at a practical multiplatform application for a more scalable use for students. The

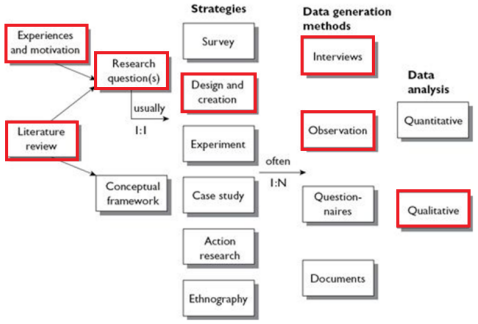


Figure 1.1: Model of the research process as illustrated in Oates (2006)

projects research question falls under the strategy of Design and Creation as the main goal is to develop a useful application for medical education. The focus on a smartphone solution was further motivated by the COVID-pandemic making from-home learning quite essential and making the passing around of HMD devices an unwanted scenario. As part of an agile software development model the gathering of qualitative data from observations and interviews within the scope of user testing will be essential.

1.5 Contributions

The research product resulting from this project will be a new computer-based software application using augmented reality and running on multiple platforms like HoloLens 1 and 2, Android and more. The aim will be to develop an application that can bridge the gap between expensive head mounted displays and everyday smartphones which you will find in the pocket of any student, and to use this as a collaborative tool for learning neuroanatomy. Throughout the development period we will consult with medical professionals and gather feedback from students on the usability of the application.

1.6 Outline

Chapter 2

Background

2.1 Augmented Reality

Augmented Reality (AR) describes the use of computer technology to generate an audio-visual experience combining real-world impressions with computer generated graphics, and, *essentially*, the ability to interact seamlessly within both domains. Ever since its infancy medical usage of AR technology has been envisioned as a great potential. The idea of x-ray vision is seen both in science fiction and in genuine research dating all the way back to the 1930s when H. Steinhaus explored ways to visualize metal pieces inside the body (Sielhorst et al., 2008). There is now substantial interest in the use of AR within a wide array of medical fields as well as in industry and education. As an emerging technology there is still much research needed, and great leaps in hardware, software and sensor capabilities are bound to happen in the near future. Already AR shows promising results in both surgical settings and in education (Singh and Kharb, 2013).

Disambiguation of some acronyms

As a new field, this field suffers from naming disagreements. This is a confusing reality which needs to be addressed. There are differing view of what each acronym refers to, and even what they stand for. I will overlook most of this discussion, and simply explain what is meant by each acronym in the scope of this project.

VR Virtual Reality, is enclosed experiences which completely surrounds the user within a computer generated world. This is a generally uncontroversial term and will be used for applications running on devices like the Oculus Rift and the HTC Vive.

AR Augmented Reality, exercises which implement a see-through effect to display 3D visuals on top of the real-world. The idea of holograms is a good stand-in for the effect of AR. This is the term which will be most used in this project.

MR Mixed Reality, anything within the spectrum between reality and pure visual 3D graphics, which blends computer generated visuals and reality. While the term has been in use since it was coined by Milgram and Kishino (1994), it has in recent years been strongly associated with Microsoft, and in this project the term MR will generally only be used as a reference to Microsoft's products or concepts. The term is also used by some as a subset of AR, so in conclusion it is a somewhat controversial term.

XR Extended Reality, much like MR this includes the whole spectrum of experiences blurring the line between the real and the virtual. However it does not have the Microsoft taint, nor the confusion of that term. And thus it is a more acceptable term, and it is what will be used here to describe the spectrum.

2.2 Graphics and Rendering

Models

Three dimensional data can be stored and visualized in a number of ways, and the way a graphical application like Nevrolens does it is very different from the ways of medical applications. Within medicine volumetric data is common, as it is just as important what's inside the model as what's outside. In conventional graphics 3D models are built up of 2D polygons which added up forms a 3D structure, this reduces rendering time while keeping the outside structure of the model intact. Figure 2.1 show a model with about 15 thousand polygons.
The process of generating polygonal models from volumetric models is quite complex, Elden (2017) writes about this process in some detail, which can be found in Appendix B. This is the model used in this research project though the model had to be simplified further to about 300,000 triangles, to run decently on the HoloLens 2.
This research also makes use of the medical data to visualize the brain segmentations inside the volume. The is three dimensional data captured from MRI with a resolution of 512x1024x512 voxels and a voxel size of 39 μm. This results in a 0.5GiB texture asset in the application memory, and rendering of 1024x1024 slices of the volume.

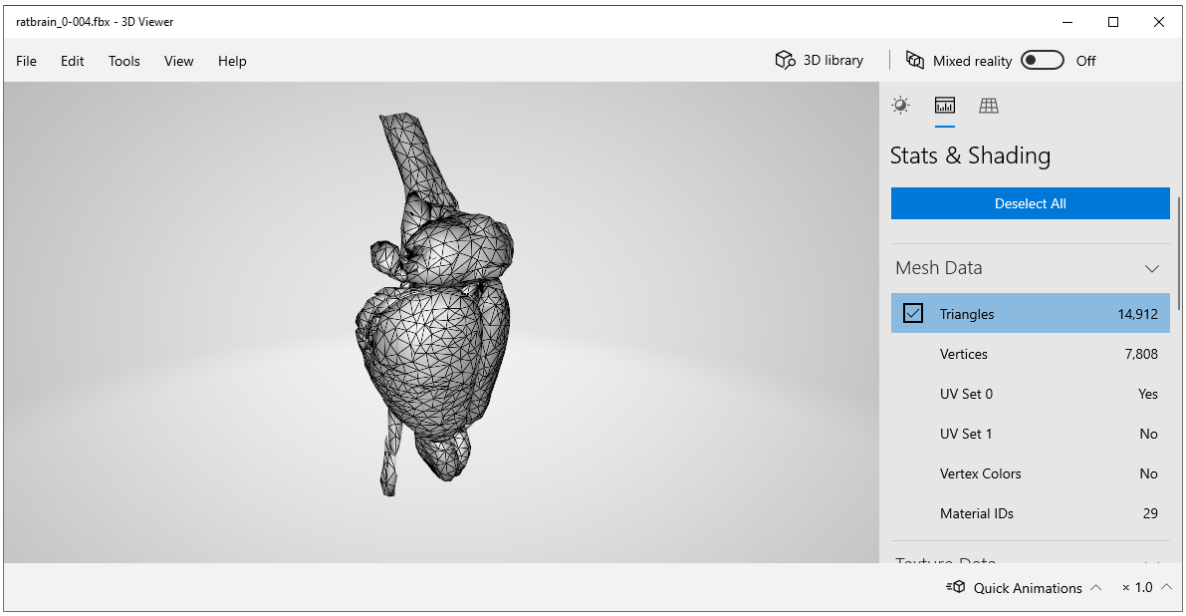


Figure 2.1: Wireframe view of the low resolution rat brain in Microsoft 3D Viewer. The triangles are clearly distinguishable, in total there is 14,912 triangles in this model.

Colors

Within computer graphics colors are generally encoded by their component primary color values in separate channels, this is called RGB for red, green and blue. This is the basis for most color models used on computers. The RGB color model is naturally used widely in this project, and will not be explained further. There is however another less common color model used in this project which has some useful properties worth exploring. This is the HSV color model. The different channels are hue, saturation, and value. The hue is simply the color based on a traditional color wheel, this means that the color will periodically repeat starting with red on zero, green on one third and green on two thirds. The saturation is how "colorful" the color is, there 0 is a gray-scale and 1 is completely colored. The value is sometimes also called light or brightness, where 0 is black and 1 is again completely colored. This periodic properties of the color model is useful for general various color

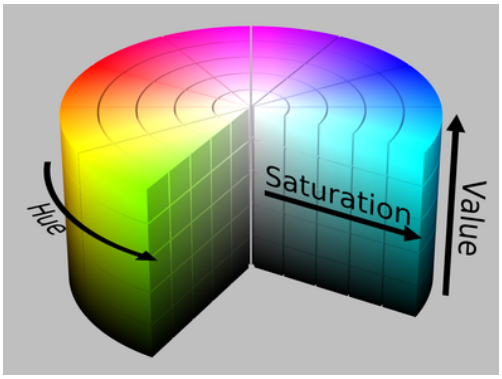


Figure 2.2: HSV color model represented cylindrically.

schemes as will be explored in Implementation.

2.3 Neuroanatomy

The study of neuroanatomy is concerned with the structural organization of the nervous system. This primarily means the brain and its structures, and is what this project will focus exclusively on. Within the study of neuroanatomy, the use of macroscopical brain dissections have long been the conventual practice for teaching the organization of the structures in the brain. Requiring cadavers and the single use of their brain, this method is highly resource intensive and has limited scalability. In addition, there are concerning ethical challenges with the use of animals in research.

The Waxholm Space Atlas of the Sprague Dawley Rat Brain

This project makes use of high-resolution 3D-models of a rat brain. This brain model has been captured and manually delineated[1] by a collaboration between research groups at the University of Oslo and NTNU, and is in fact a highly accurate volumetric representations of the rat brain. This model is an open access community resource, intended as a free tool for education and research[2]. Within the convectional rasterization rendering pipe-line of Nevrolens, a geometric asset derived from this volumetric model is naturally used.
The model is referred to as *The Waxholm Space Atlas of the Sprague Dawley Rat Brain*. That means a atlas of the *Sprague Dawley* rat breed defined in Waxholm Space. I will briefly explain what a brain atlas is and what Waxholm space is.

Brain Atlas

A brain atlas is a composite representation based on one or more datasets of a given brain. An atlas generally has the function of highlighting some specified aspects and relations in the brain, and is a convectional tool in neuroscientific research (Toga and Thompson, 2000). The convectional atlas is based on micrometer scale sliced sections in the brain, effectively creating two-dimensional layers through the brain. While functional, this "turns the brain into a book". Three-dimensional digital atlas are however relative newcomer on the neuro-imagery scene, by employing magnetic resonance imaging (MRI) and diffusion

[1] Delineation refers to the process of clearly defining different structures in the brain into separate namable parts.
[2] https://www.nitrc.org/projects/whs-sd-atlas

tensor imaging (DTI) the resulting atlases are complete volumetric representation of the subject brain (Papp et al., 2014).
This volumetric model is the basis for the delineated 3D-model used in Nevrolens.

Waxholm Space

Waxholm Space (WHS) is a vector space defined as a standard reference space for the mouse brain and the rat brain (Papp et al., 2013). Its use as a coordinate system simplifies interoperability across atlases. It was developed by International Neuroinformatics Coordinating Facility (INCF) for the mouse brain, and has further been implemented in the rat brain by Papp et al. (2014). The following is the formal definition of WHS:

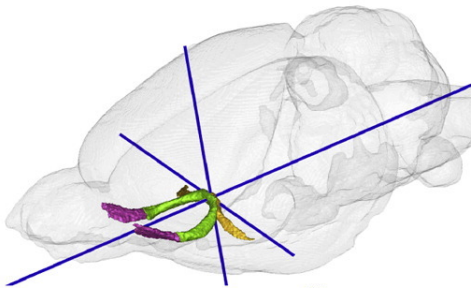


Figure 2.3: Waxholm Space (Papp et al., 2014)

> *The coordinate system for WHS is defined as a continuous Cartesian system with the origin in the brain determined by*
>
> - *the anterior commissure (AC) at the intersection between the mid sagittal plane,*
> - *a coronal plane passing midway (rostro-caudal) through the anterior and posterior branches of AC, and*
> - *a horizontal plane passing midway through the most dorsal and ventral aspect of the AC.*
>
> Hawrylycz et al. (2011)

Figure 2.3 visualizes the axes through origin of WHS in the brain of a rat. Within the scope of this project WHS will be the local space of the rat brain model implemented in Nevrolens.

Teaching of Neuroanatomy

The way students are educated in brain anatomy today in large part consists of two main approaches. The first is text books with two dimensional illustrations with accompanying

textual descriptions as quoted from Hawrylycz et al. (2011) in the previous section. Secondly, dissection of cadavers is used to demonstrate real brain anatomy. Optimally this would be done with human brains, does are however naturally difficult to attain and thus the most used brain it that of the rodent. This approach has the problem that it is not very scalable and the size of the rat brain make it quite trick do demonstrate structures of anatomy.

2.4 Equipment

HoloLens 2

HoloLens 2 is the second iteration of Microsoft immersive headset line. It uses an ARM-based computing unit, running a custom holographic version of Windows 10 for ARM. This enables the HoloLens 2 to produce high quality graphics while being very power efficient. It was announced in early 2019, with a limited release on November 7, 2019. It is however jet, as of writing, not publicly available and could be considered a limited industrial product. As the technology stands today, HoloLens 2 is the most complete augmented reality device on the marked, with interaction features like hand tracking and eye tracking in combination with the most immersive display technology in any AR HMD. This makes it a natural device choice for developing AR applications with today. Very helpfully the HoloLens 2 has on-board screen capturing tools and the option to live preview the video feed from the Windows Device Portal. These features have helped greatly both in user testing and in demonstration the application.

Display	Optics: See-through holographic lenses (waveguides) Resolution: 2k 3:2 light engines Holographic density: >2.5k radiants (light points per radian) Eye-based rendering: Display optimization for 3D eye position
Sensors	Head tracking: 4 visible light cameras Eye tracking: 2 IR cameras Depth: 1-MP Time-of-Flight (ToF) depth sensor IMU: Accelerometer, gyroscope, magnetometer Camera: 8-MP stills, 1080p30 video
Audio and speech	Microphone array: 5 channels Speakers: Built-in spatial sound
Human understanding	Hand tracking: Two-handed fully articulated model, direct manipulation Eye tracking: Real-time tracking Voice: Command and control on-device; natural language with internet connectivity Windows Hello: Enterprise-grade security with iris recognition
Environment understanding	6DoF tracking: World-scale positional tracking Spatial Mapping: Real-time environment mesh Mixed Reality Capture: Mixed hologram and physical environment photos and videos
Compute and connectivity	SoC: Qualcomm Snapdragon 850 Compute Platform HPU: Second-generation custom-built holographic processing unit Memory: 4-GB LPDDR4x system DRAM Storage: 64-GB UFS 2.1 WiFi: Wi-Fi: Wi-Fi 5 (802.11ac 2x2) Bluetooth: 5 USB: USB Type-C

Figure 2.4: Specifications for the HoloLens 2

Android

Android is a operating system developed by Google. The OS runs on many different device types, but most commonly and most relevantly for this project it runs on smartphones. Nearly all modern smartphones not made by Apple run on the Android operating system and they thus have a wide marked penetration, which is one of the reasons for using this OS in this research. With support for development in combination with HoloLens 2 and good support for AR this was a natural choice. Apples iPhones would also be a good choice however Apple is restrictive on how one can development for their platform which makes supporting it difficult. The Android device mostly used in this research is the Samsung Galaxy 8, both the research's personal device and the devices at the VR Lab are of this model. It was release in 2017, and has internals which in theory should make it a bit less performant than the HoloLens 2, it is however modern enough for supporting AR applications and throughout this research it has not been found lacking.

2.5 Software Tools

Unity

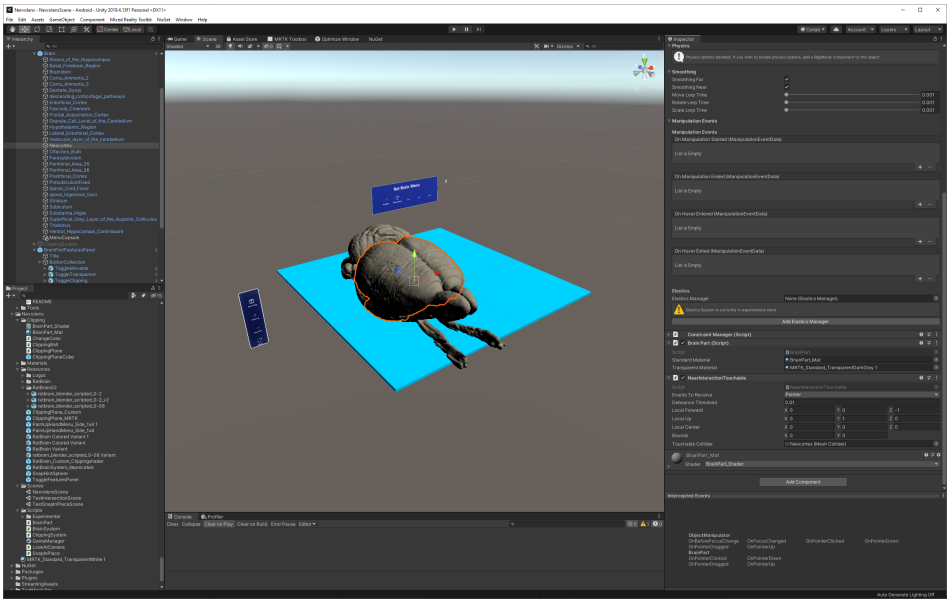


Figure 2.5: Unity 2019.4.13f1 running Nevrolens

Unity is a game engine for developing 2D and 3D games. It has grown to become the most popular game engine used by single developers and small development teams because of its ease of use and simple licensing terms for independent developers. Because of its popularity and ease of use Unity has become a platform for 3D development within more widespread fields than video gaming, such as engineering, moviemaking and architecture. Within this project the critical reason for choosing Unity for our 3D development is the exceptional support for the HoloLens product line. As seen in the section 2.5, Microsoft has poured resources into developing a "relatively" robust open framework for using Unity to develop for HoloLens. Alternatives to using Unity are slim, but one could be to use Unreal Engine, an 3D game engine with great support for VR and AR in general, however the support for HoloLens specific tools like Mixed Reality Toolkit is limited, Microsoft has a version of MRTK for Unreal, called MRTK-Unreal. It seems to be stale however, not having any updates in the last six months in the time of writing.

Mixed Reality Toolkit

Mixed Reality Toolkit (MRTK) is a open source, Microsoft-driven framework for Mixed Reality (MR) development. In practice it is Microsoft's SDK for HoloLens development, greatly simplifying development related to interaction, user interface and device sensors. As it is a framework for MR in general, it supports other platforms like Android, iOS and VR devices such as HTC Vive and Oculus Rift with OpenVR. An alternative to using MRTK would be to us XRTK which is a community-driven fork of MRKT. Thought such a choice would be an exercise of free software principles, it also lends it self to better support for some devices, like the MagicLeap.

Blender

Blender is a 3D modeling application, it is free open source software and is has a wide set of functionalities for 3D modeling, animation, rendering and optimization. This was chosen because of its free and open availability.

Photon Unity Networking 2

Photon Unity Networking 2 (PUN) is the state of the art networking library for Unity. It can manage everything from voice chat to interaction over network.

Windows Device Portal

Windows Device Portal is an web-based application for managing devices running Window, like HoloLens. The HoloLens 2 hosts this application if it's connected to a network, so you can easily log in on the device and manage files, profile application or stream video from it.

Git

Git is a distributed version control system. Together with hosting on NTNUs self-managed GitLab it enables version control and cloud back-up of the project. While this is the most conventional version control system for any software project, using Git with Unity can be frustrating. Git is design for projects with only (mostly) small, human-readable text files, like a code base. A Unity project often as huge files, which Git does not support, and relevant changes can happen in binary files, which makes merging impossible. To mitigate this problems the use of Git LFS was needed. It is an extension for Git which enables

storages of larger files. Together with enabling only human readable settings in Unity and a long gitignore file, Git with Unity was manageable. A good alternative to this use of Git is *Unity Collab*, which is Unity's answer to version control, it lack many features found in Git, but would probably be just fine for a project of this scope with a single programmer. However, I like Git very much and find the feature-set of Git to be very helpful.

GitKraken

GitKraken is a graphical Git management tool. What's more relevant here is its Kanban feature, or GitKraken Boards as their called. This enables synchronization of Kanban tickets and feature branches in Git, and generally makes my life easier.

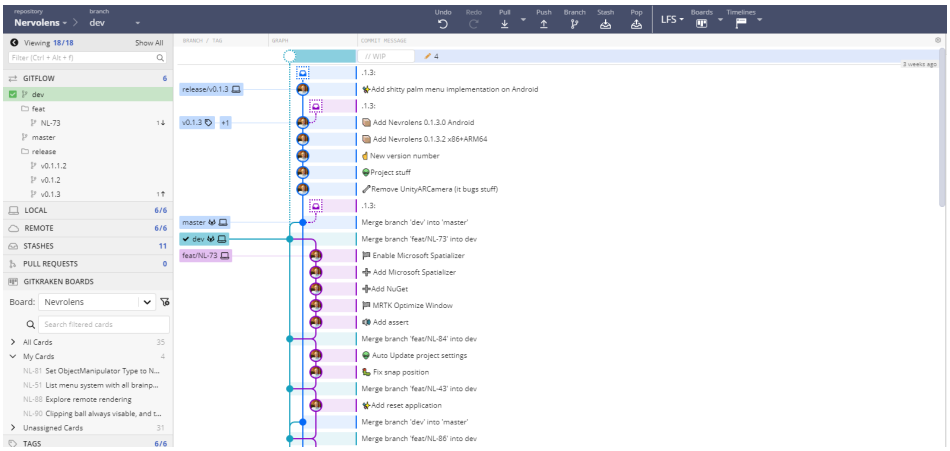


Figure 2.6: Git log in GitKraken, on the left you can see synchronized Kanban tickets

2.6 Related work

VRVisualizer

VRVisualizer is the name of the research product from the master thesis Elden (2017). It is a VR application running on the HTC Vive. While its features are similar to the goals of this project, the focus of Eldens project was to develop universal guidelines for scientific visualization in VR. That is far from this projects goals and thus its feature set, relating to interaction and exploration, is quite superficial.

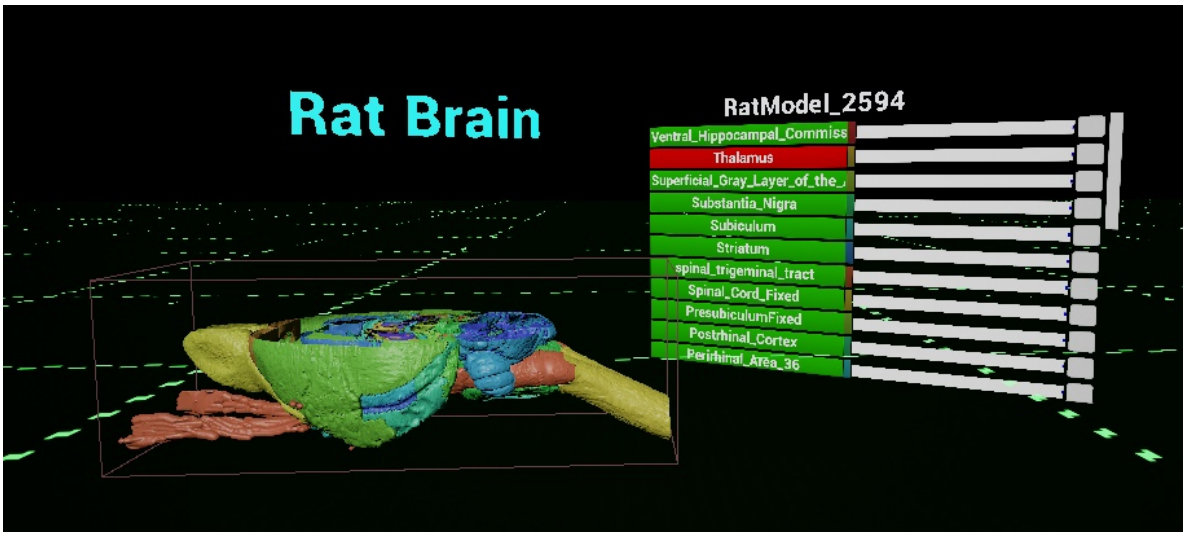


Figure 2.7: Cutting operation in VRVisualizer on HTC Vive

ITK-Snap

ITK-Snap is an open-source interactive computer application for navigating medical imagery. It is a convectional tool used to display, segment and label 3D neuroanatomical data. It supports a wide array of visual tools which helps in learning about the brain. However, the fact that it is purely displayed on a two dimensional monitor, means it limits the opportunity of spatial understanding.

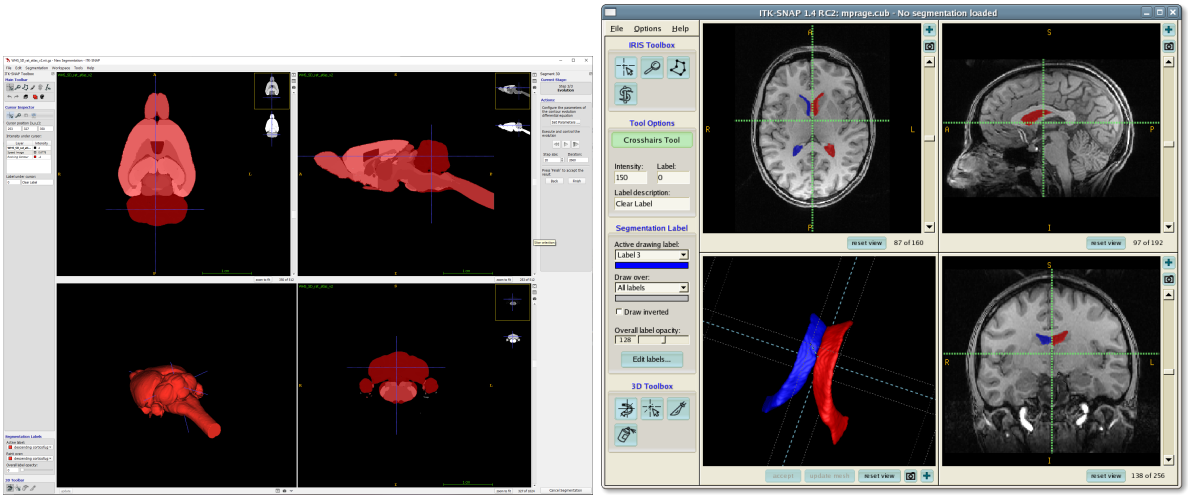


Figure 2.8: ITK-Snap, Waxholm Space Rat Brain (left), Human brain (right)

HoloAnatomy

HoloAnatomy is a application running on HoloLens, which aims to be a educational tool for anatomy including basic neuroanatomy. Wish-Baratz et al. (2020) has shown great promise in use of this application for education. While this application is really promising, it lacks some features we would like to focus on. There is no use of mobile devices for collaborative learning. And HoloAnatomy focuses on the macroscopic image of anatomy, and does not go into microscopic detail, which is something this project aims to do. Further, at NTNU and the Kavli Institute research and education using rat brains is normal an thus visualizing it it AR is also a focus.

Figure 2.9: HoloAnatomy

Chapter 3

Requirements

The first meeting initializing the project took place at VRLab Dragvoll in early September, here I was introduced to the general background and the problem description of how Witter and others envisioned the use of AR for neuroanatomical education. It was explained how cadavers for education are difficult to acquire and therefore used quite sparingly. Another problem we discussed was related to the difference in medium between VR and AR. While the application VRVisualizer did have many of the features envisioned, and could have been a basis for further development. The fact that is was implemented in VR was problematic for the envisioned use cases. Being completely enclosed visually limits its use case in lectures and in any use case with collaboration in a physical space. Generally the loss of spatial awareness and eye contact as a result of using VR headsets was though of as an impediment for using VR for such an application. Thus, we had an outline of a neuroanatomical education tool in AR using the HoloLens 2 and concluded with some questions and requirements for the project:

1. Can the current VR dataset[1] be used in the HoloLens 2 AR environment?
2. If not, which steps need to be taken to use the segmented WHS rat brain to develop a suitable 3D model that can be used in AR?
3. Develop an optimal user interface for a single person to explore the rat brain as if the user is doing a dissection of a real brain.
4. Develop/test ways to make this a multiuser/shareable tool adequate in a teaching environment.
5. Explore ways to integrate microscopical data into the AR representation.

6. Describe/explore the feasibility to implement the system for Human neuroanatomy education.

Here items 1-4 were deemed critical for the project, while 5 and 6 were dependent on the progress made.
This meeting together with the list formed a clear problem description and can be seen as the initial discovery process of the project. Though the following period of exploring the newly arrived HoloLens 2 and its capabilities, we formed a set of *system requirements*. System requirements are descriptions of how a system should operate, what it should be able to do and the constraints of its operation. The requirements must reflect the stakeholders needs for the system (Sommerville, 2011). System requirements are generally split into functional requirements, which describe specifics of what the system (and its subsystems) should do, and non-functional requirements, which generally are descriptions of the user experience of the system as a whole. What follows are the system requirements decided on for the application:

Functional Requirements

1. **Implement a brain dissection tool in AR.**
   The app should render a brain at sufficient quality for educational use, and have the tools for creating a dissection experience in AR.

2. **The application must run in HoloLens 2 and at least one mobile platform**
   The ability to run a version for the app on multiple platforms is essential for the purpose of this project. While the main platforms are HoloLens and mobile, others may also be implemented in the future.

3. **Implement cross-platform collaboration over network**
   For the application to have value above a single user it is important that it can be used with a HoloLens and a more accessible platforms in a collaborative manner.

Non-Functional Requirements

1. **Medical students should find educational use for the app.**
   It is critical that there is educational value in the application.

[1] Referring to VRVisualizer by (Elden, 2017).

2. **The application should be usable without outside guidance.**
   The app should have a clear and understandable design, such that a new user should be able to navigate the app by them self, even with minimal experience with AR.

3. **All relevant usability criteria for a mixed reality app should be met.**
   We should work to not fall under the 'meets' criteria on any relevant metric in the App quality criteria[2]. This includes criteria on; FPS, spatial anchoring and view comfort.

[2]https://docs.microsoft.com/en-us/windows/mixed-reality/develop/platform-capabilities-and-apis/app-quality-criteria

Chapter 4

Technical Design

This chapter will give a overview of the structure of the application as well as some choices taken when developing the research product, Nevrolens. The chapter will exclusively focus on the application as it is at the end of this research project, which is functionally identical to version 0.3.3 of Nevrolens. However, some refactoring i.e. name changes and restructuring may have occurred.

4.1 Game Structure

Unity Scene Graph

Within Unity a *Scene* consists of a *scene graph* which is a tree structure of `GameObjects`. By default a scene consist of a `Directional Light` lighting up the scene at its default light source and a `Main Camera` which is the view point of the running game. In addition, the MRTK library will add two objects to the scene graph, one called `MixedRealityToolkit` which contains configuration of the Mixed Reality features and systems. This is where input systems are defined and where control of spatial awareness and boundary detection is handled, in short all features and sensors of the HoloLens system or other AR system are defined and controlled here. The other object added by MRTK is the `MixedRealityPlayspace`, this encapsulates the `Main Camera`, but is lacking any useful documentation on what its purpose is. The name could be hinting at it being the parent of the *Playspace*, meaning all `GameObjects` in the game. However, even MRTK demos seem to ignore this object and thus it has not been used in this project either.
The functionality of the scene graph, other than organizing GameObjects, is that child objects inherit the position, rotation and scale of their parents, thus simplifying transfor-

mation of complex object. This naturally structures many systems, however in a AR application there can be many independent 3D objects floating in space. In addition, some objects are dependent not on their parent, but on a defined object `Transform`. Therefore, some organization is needed and some objects are placed by choice and convenience rather than any practical reason. Another practical use for child objects are the use of the `GameObject.GetComponentsInChildren()` and `GameObject.GetComponentInParent()` methods which allows for simple access to `Components` in child and parent `GameObjects`, this is however of limited use as such dependencies in code has a tendency to result in tedious bugs.

The top most application specific object of the project is the `BrainSystem`, this acts as the parent GameObject for all objects defined by the application. The right side of Figure 4.1 gives an overview of all 3D object in the `BrainSystem`. The `InfoBoard` on the right, the button group, or `HandMenu` in the center and the complete brain model with axes etc. named `GameWorldSpace`, are spatially independent systems all having `BrainSystem` as a parent, this can also be seen in Figure 4.1 in the scene hierarchy on the left side. The reasoning for having the parent object `BrainSystem` is purely to to tidy up the top layer of the scene graph and having a clear distinction of project specific custom objects.

The main attraction within the `BrainSystem` is the `GameWorldSpace`, it is the parent of the brain model and all objects with are spatially dependent on the brain. This allows for movement and scaling of the whole model worldspace. This is also the local space of the synchronized multiplayer world.

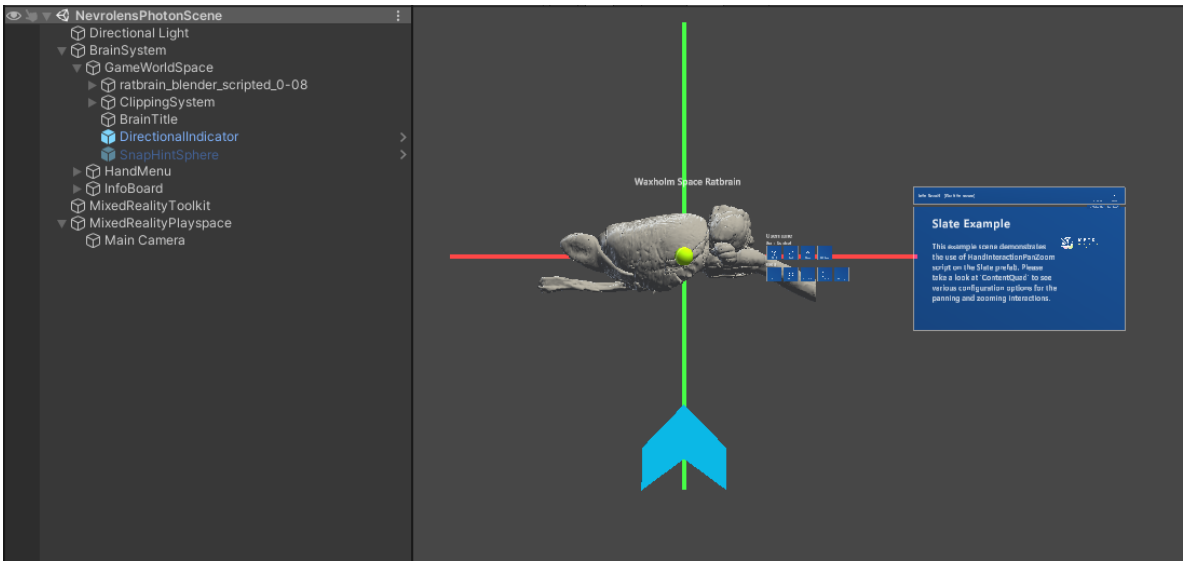


Figure 4.1: Every 3D object in the `BrainSystem`

4.2 Networking

Networking Solutions

Multiplayer games in Unity can be created in numerous ways, in the development phase of this project three solutions were explored; UNET, LiteNetLib and Photon PUN2 . Common for all are that they are mature, reliable and are well documented, they all support multiple device types including all devices within the scope of this project, there are however some very clear differences making the choice for this project quite simple. *UNET* is Unity's own default networking solution, it provides high level functionality and is generally easy to use. It is however deprecated and will be discontinued by the end of 2021, an open source fork of the networking API, called *Mirror* has seen continuing development and improvement, but because of the state of the original project, both were deemed nonideal. Unity is working on a new networking solution called *MLAPI*, it is in alpha stages but shows great promise.
LiteNetLib is a open source, and more low level framework. It is intended for use cases where in-depth control of the networking processer are wanted or needed, if high performance and low latency is important this would be a good choice. It supports peer-to-peer and self-managed servers. Because this project can be thought of a small scale proof of concept, it is of limited concern whether the networking is highly performant and seeing as a low level API is more complicated to implement it is neither a optimal use of a single developers time.
Lastly, *Photon PUN2* is the an high level networking library with managed hosting and a free basic plan for up till 20 concurrent users. This makes it ideal for small projects and single developers. It is also the general first choice for networking solution in Unity and its surge in popularity is partly the reason for Unity abandoning their own solution. PUN2 was a natural choice simply because is of its low barrier for entry and it having a free hosting options making development as easy as possible. It being the most popular solution also has the added benefit of having well made tutorials and forums for troubleshooting.
While developing this application, Microsoft announced a new solution for networking specifically targeting MR applications called *Microsoft Mesh*, it promises to solve networking, and many MR specific problems like spatial anchoring and face-to-face interaction. This could be a promising step for this application in the projects continuation, and should be kept an eye on by future researchers.

Connection

In this project, networking has as stated been implemented as simply as possible for a proof of concept and because of time constraints on a single developer. If concerns like scalability and reliability was of higher importance different choices would have been made, and steps to fulfill those concerns should probably be taken in future development. When using the networking, which is how the built versions of the application are set up, the application initially launches in a empty `Scene` named `NevrolensStartPhotonScene`, its only purpose is connecting the user to the server and creating or joining a *room*. A room is a Photon abstraction for connecting users to the same game state. Listing 4.1 shows a striped down version of the script running in this scene, it is a complete and functional script to emphasize the ease of implementation. The script is all that is needed to initiate a connation in Photon PUN2, and is all connection handling in the application. The implementation has, because of its simplicity, some flaws, Figure 4.2 shows that if connection to the photon server fails the application will give no response and the user will just stay in an empty scene, this could easily be fixed by either giving some error message feedback with a retry button or even loading the game scene in offline mode, neither has been implement mainly because connation issues have seldom raised and thus development time has not been invested in fixing this problem.

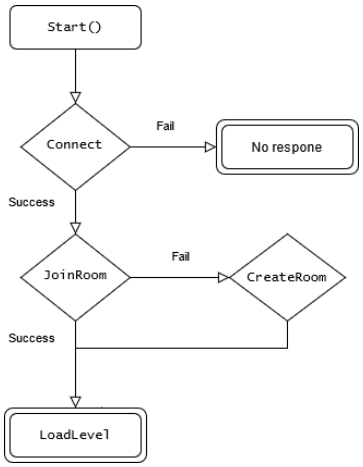


Figure 4.2: State diagram of the implemented connection process

```
1   void Start() {
2       PhotonNetwork.ConnectUsingSettings();
3   }
4   override void OnConnectedToMaster() {
5       PhotonNetwork.JoinRandomRoom();
6   }
7   override void OnJoinRandomFailed(short returnCode, string msg) {
8       PhotonNetwork.CreateRoom(roomName: "room1");
9   }
10  override void OnJoinedRoom() {
11      if (PhotonNetwork.IsMasterClient)
12          PhotonNetwork.LoadLevel("NevrolensPhotonScene");
13  }
```

Listing 4.1: The connect process in a Unity `MonoBehaviour` written in C#.

Another implication of this design is that every user will necessarily connect to the same room. This happens because the first user will find no room and thus create a new one, while all others will find the one room and connect to it. The user has no control over who they play with, and can not start a session by them self, both could easily be implemented, but would also result in overhead for the user as they will have to make a choice which now is simply made for them.
All in all this solution work well enough for the current state of the research project. In fact, by abstracting away the connection and room selection process it has simplified the user testing process because there are fewer steps to get to a running application.

Multiplayer world space

When collaborating with other players there is a need for having a synchronized world space, by default Photon PUN2 will synchronize all objects spatially by their own model space, or `local space`. This works good for basic tasks like synchronize the movement of a brain structure relative to the others, but meets problems when separating the shared multiplayer world space from from the local user world space. This is needed so that a user can move the AR objects to fit in their field of view, physical surroundings and at the scale appropriate for their comfort and device type. For this reason all GameObjects which are to be synchronized in the shared multiplayer world space are placed as children of an empty GameObject called `GameWorldSpace` and the local model space of this object is used as the multiplayer world space. With this implementation, and manipulation of the `GameWorldSpace` is local and not shared over network, while any manipulation of its child object will be reflected for all users over the network.

Chapter 5

Development Process

5.1 Software Process

Even though I have developed the Nevrolens application by myself, I have strived to use best practices for a software development workflow. These practices have generally grown out of the the needs of a multi-developer setting, enabling simpler use of collaboration and version control. Though their value possibly increases exponentially by the number of team members, I have found value in the structure and clarity I find in the workflow.

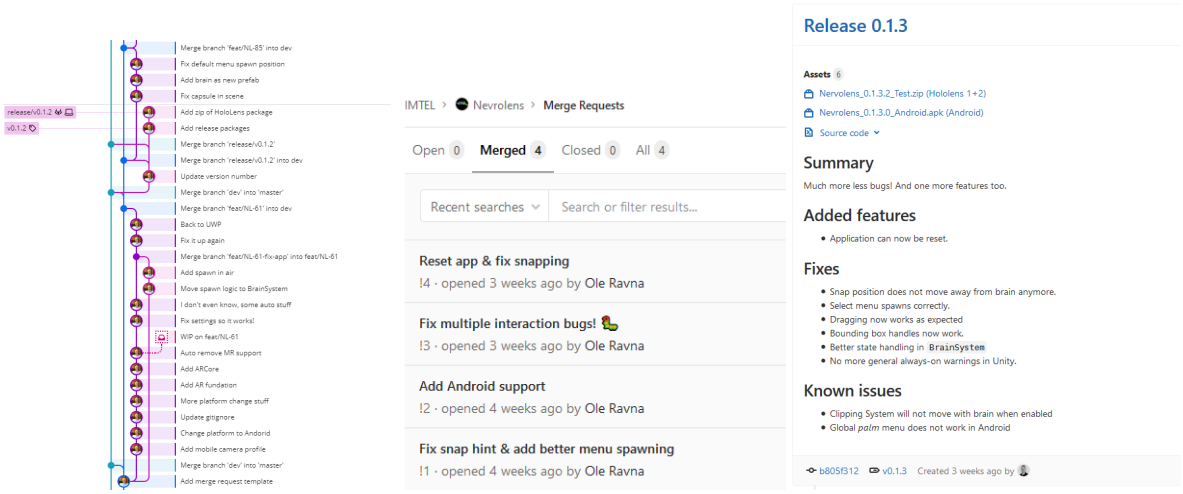


Figure 5.1: Feature branches, merge requests and releases.

My workflow is based on *Gitflow*, a workflow framework optimized for continuous software development. In short, this is just a very basic rule set for branch-naming and the sanctity of the master-branch (requiring merge requests of only product ready code), within the version control system Git. It does however act as a fundament which enables prac-

tices like rapid release cycles, because of the clearly define production ready state, and the integration with lean development technics like Kanban. This stems from the parallels between feature-branches in Gitflow and the *ticket* in Kanban. In practice, this means that tickets, with issues or new features for the app, are created on in the *Backlog* column of the Kanban board and are then moved to *Doing* column simultaneously as a feature-branch is created with the ID of the ticket, e.g. `feat/NL-42`. All of this is automated in the Git management tool GitKraken, which manages both the git-repo and the Kanban board.

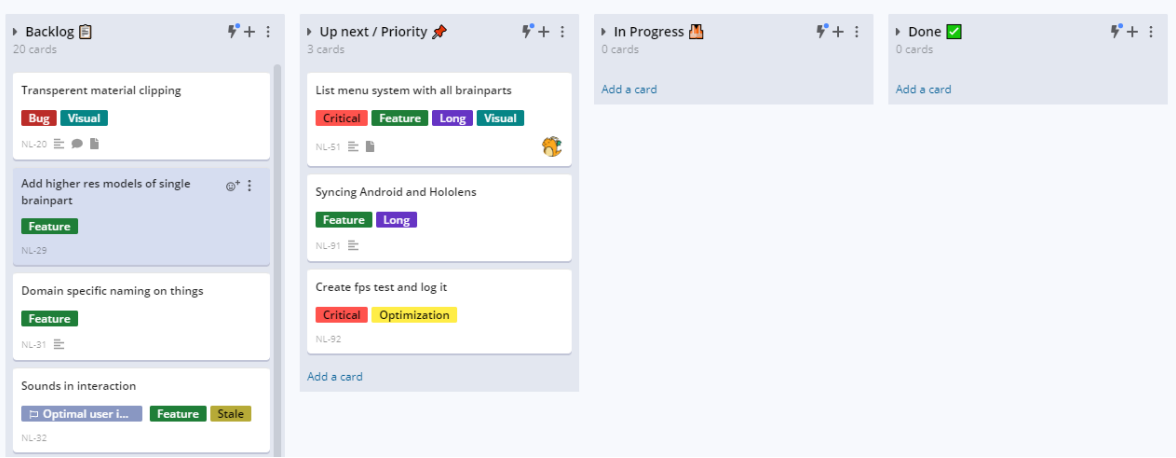


Figure 5.2: A snapshot of the Kanban board in GitKraken, after a development sprint, when completed tickets are archived (closed).
Note: *Priority* acts as pined tickets on *Backlog*, as the backlog tends to sizeable.

This workflow, by design, supports an agile development process. Agile approaches to software development are generally human-centered, valuing individuals and interactions over processes and tools[1], and focused on iterating rather than upfront planning. This is ideas which are beneficial for single-developer or small teams especially when developing for new platforms like the HoloLens 2. While the project aims for an agile approach, the sprint cycle core to the agile development, where stakeholders are involved for regular feedback, has, due to a number of factors like COVID-19, only really been done for one cycle. However, the steps taken for an agile workflow should enable more agile development for the master project.

[1]The Agile Manifesto https://agilemanifesto.org/

Chapter 6

Implementation

This research project has been in active progression for two semesters and development has effectively been done in two stages. First in October and November 2020 and then from February till May in 2021. During the fall and first stage of development the focus was on the HoloLens 2 and creating a usable act of dissection in augmented reality. The second stage at spring, had a wider scope, with focus on implementing networking, volumetric data and more. During this chapter it will thus be clearly discrete improvements during the first iterations of the product while later iterations will be grouped and focus will be put on specific features or pain points. The final iteration will focus on improvements done from user feedback and the feature set at the end of development.

6.1 Iteration 0: First steps, importing model

The first phase of development started by acquiring the surface model of the WHS rat brain created by Elden (2017). This was done by simply moving the FBX files from the VRVisualizer application and to a new Unity project. After initializing MRTK by following their Getting Started documentation, the application was ready to deploy on the HoloLens 2. This resulted in a barely running application as the polygon count of the brain model was orders of magnitude larger than what is recommended to maintain adequate performance on

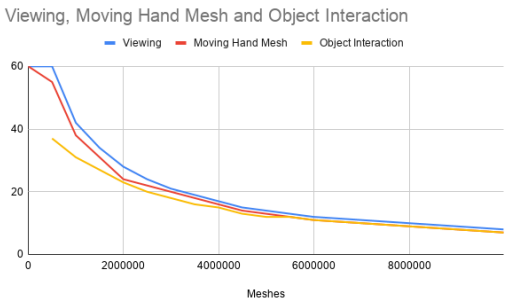


Figure 6.1: Figure showing frame rate as a function of polygon count on HoloLens 2. Credit: Fologram

the HoloLens 2, which is in the order of 100,000 polygons shown in Figure 6.1. The model used by Elden was scaled to run on workstation computer outputting to an HTC Vive, and thus his model was reduced from a original 16 million polygons to around 3 million. The HoloLens 2 runs all calculations on-device on a mobile ARM-based processing unit and naturally the brain model created for rendering on a dedicated workstation graphics card had to be further scaled down. This downscaling was first experimented with doing at run-time dynamically on-device using the library *UnityMeshSimplifier*. It was quickly determined that this was not a viable solution both because of untenable processing time, but also because the simplified result had a huge impact on quality of model, hinting at the performance optimization that had to be done on the simplifier algorithm to be able to execute at run-time. The next and final solution for downscaling was to use the *decimate* modifier in Blender. *Incremental decimation* is a mesh simplification algorithm which trades some speed for higher mesh quality, in contrast to the *vertex clustering* presumably used in UnityMeshSimplifier which prioritizes speed in such a way that topology is not preserved. Within Blender functionality for simple application of the modifier to all objects in a tree-structure was not found, or understood to exist, so a script applying the decimate modifier with a given ratio was written, see Listing 6.1. The `ratio` parameter is a value between 0 and 1, representing the scaling of the resulting mesh' polygon count.

```
import bpy # importing the blender python library

def decimate(ratio, replace = True):
    # Finds all objects and filters irrelevant objects from the FBX
    brainparts = [n for n in bpy.data.objects \
        if n.name not in ("Camera", "Light")]

    for part in brainparts:
        mod = part.modifiers.new(type='DECIMATE', name='Decimate')
        mod.decimate_type = 'COLLAPSE'
        # Sets the specifies strength to the decimate operation.
        mod.ratio = ratio
# Calls function with given decimate strength.
decimate(0.08)
```

Listing 6.1: Blender script applying a decimate modifier to all relevant objects in a scene.

The resulting decimated model, even at the ratio of 0.08, was visibly nearly indistinguishable from the original model when looking at them through the HoloLens 2 display which, as described in section 2.4, is somewhat blurry. Figure 6.2 shows the difference as seen in the Unity editor. Ultimately, a decimation ratio of 0.08 was chosen as a compromise

between detail and performance being about 300,000 polygons, this compromise will be discussed further in section 10.3. At this stage requirement 1 and 2 in the initial requirements from chapter 3 could conclusively be answer as possible and completed.

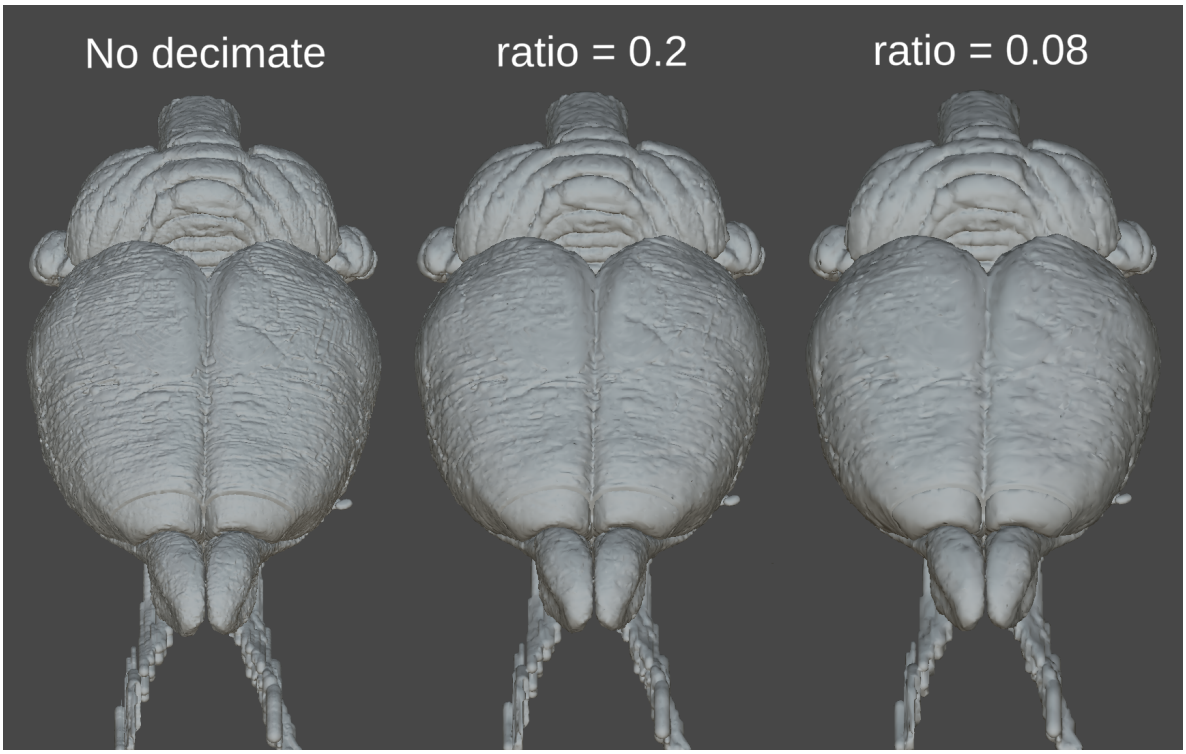


Figure 6.2: WHS brain models with 3M, 744K and 297K polygons respectively.

6.2 First iteration: Minimum Viable Product

Having a surface model of the brain running reasonably well on the HoloLens 2, the next step in developing the application was to implement basic AR-based interact features. The brain model consist of an empty parent object with 29 children each containing the mesh of a delineated brain structure, see Figure 6.3. Adding the `Object Manipulator` component from MRTK and a standard Unity `Mesh Collider` component to each child in the brain model allows for picking apart the brain. This is done by grabbing and moving each separate structure with a MRTK defined *pointer*, this is the logical abstraction for the simplest interact handling with HoloLens 2 giving the user a virtual laser pointer from their finger. The resulting action can be seen in Figure 6.4.
An apparent problem at this stage was that thought the brain structures are separate objects, they were difficult to visually distinguish from each other. A script which took all

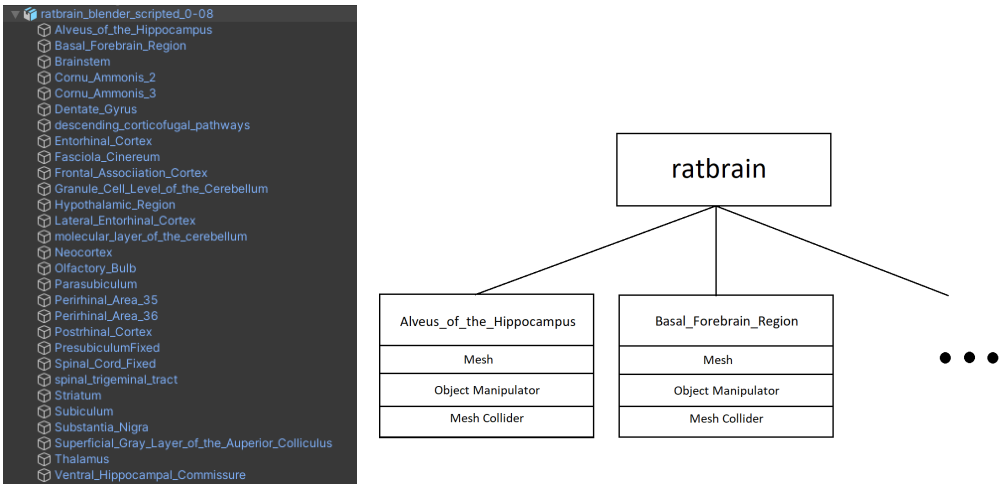


Figure 6.3: The tree structure of the Unity `GameObject` of the brain model.

child objects and applied a random color to each was written and placed on the parent object, thus quickly giving some visual separation of the structures. While implementing this feature, the *material* of each child was changed from Unity's default material to a *MRTK Standard* material. Materials are the way Unity handles rendering details for each object, this is where shader, texture and general rendering options are configured. The MRTK Standard materials is a set of materials using the the `MixedRealityStandard.shader` shader, this shader is optimized for MR use, and superficially for HoloLens, and is meant for fulfill all shader-needs when developing for these platforms.

With a some basic visibility and manipulation features for the brain model, the next natural step was tackling the system requirements, specifically the first functional requirement, implementing brain dissection. A *clipping* shader was written and implement to work with the brain, giving more control over the feature than using MRTKs prebuild clipping feature, but seeing as it was not possible to combine a custom shader with MRTK optimizes feature set for AR rendering, the custom clipping implementation was abandoned in favor of MRTK, using the aforementioned MixedRealityStandard shader. Clipping has the effect of removing vertices by some defined function, and by using a prebuilt clipping plane prefab and declare on which meshes is should act, a dissection affect was created. A handle for manipulating the plane was added for ease of use, by dragged a ball the plane would move such that it was a fixed distance from the ball and perpendicular to the line between the ball and the center of the brain.

Further, a hovering menu displaying the name of the last grabbed brain structure and buttons for the actions moving, transparency and dissection was implemented. This was created by modifying a MRTK prefab and updating its name based on the name of the

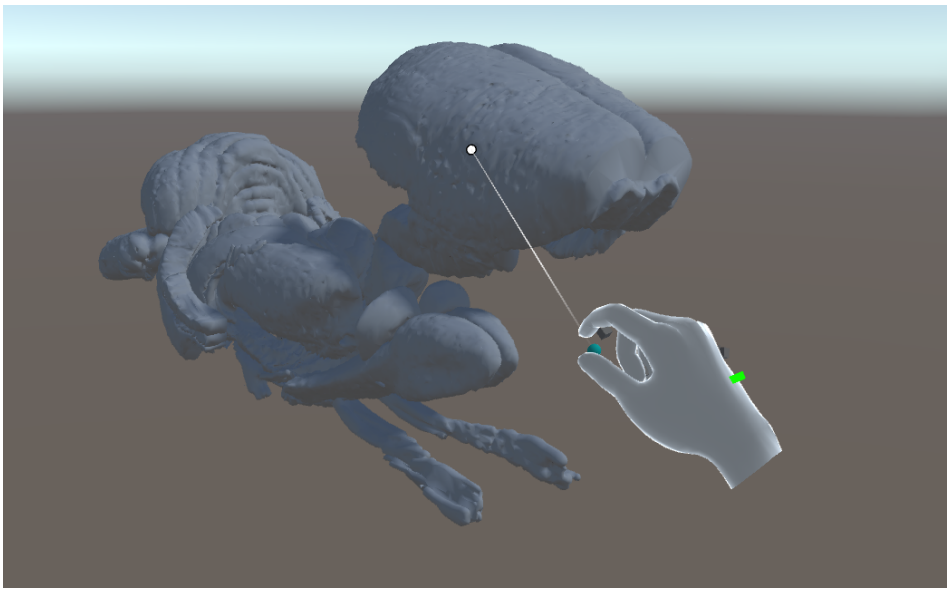


Figure 6.4: Grabbing the neocortex brain structure with a MRTK pointer in the Unity editor.

`GameObject` the `pointer` targeted while dragging, at the same time a selection lighting effect as applied by simply enabling `Border Lighting` in the MixedRealityStandard shader. Unity's layer functionality was used to ensure that it was a brain structurer being dragged. One last feature implemented at this phase was a tap-to-spawn feature, this entailed using the `pointer` to tap on the physical space, and using spatial awareness to place the brain at the locations the the user tapped. In MRTK spatial awareness is enabled by default and its mesh can be identified by a predefined Unity layer, thus Listing 6.2 shows a simplified implementation of the `EventHandler` method, `OnPointerDown` which spawns the brain if the pointer is hitting the spatial awareness mesh and enables border lighting and menu text if it hits a brain structure.

```
1  void OnPointerDown(MixedRealityPointerEventData eventData)
2  {
3      if (!HasTarget(eventData.Pointer))
4          return;
5      Vector3 hitPoint = GetHitPoint(eventData.Pointer);
6      GameObject target = GetCurrentTarget(eventData.Pointer);
7
8      switch (target.layer)
9      {
10         case SpatialAwarenessLayer:
11         {
12             if (BrainHasNotBeenSpawned())
```

```
13                  SpawnBrainAt(hitPoint);
14          }
15          case BrainStructureLayer:
16          {
17              if (selectedStructure != null)
18                  DisableBorderLighting(selectedStructure);
19              EnableBorderLighting(target);
20              SetMenuText(target.name);
21              selectedStructure = target;
22          }
23      }
24  }
```

Listing 6.2: A simplified version of the event function called when a `Ponter` is clicked.

The application was deployed for HoloLens 2, and was a first MVP demo of the research project. Figure 6.5 shows spawning of the brain model from image 1 to image 2 in the top row, notice the pointer on the table in image 1. Image 3 illustrates the clipping feature, while image 4 has a user taking out the *cornu ammonis 3* brain structure.

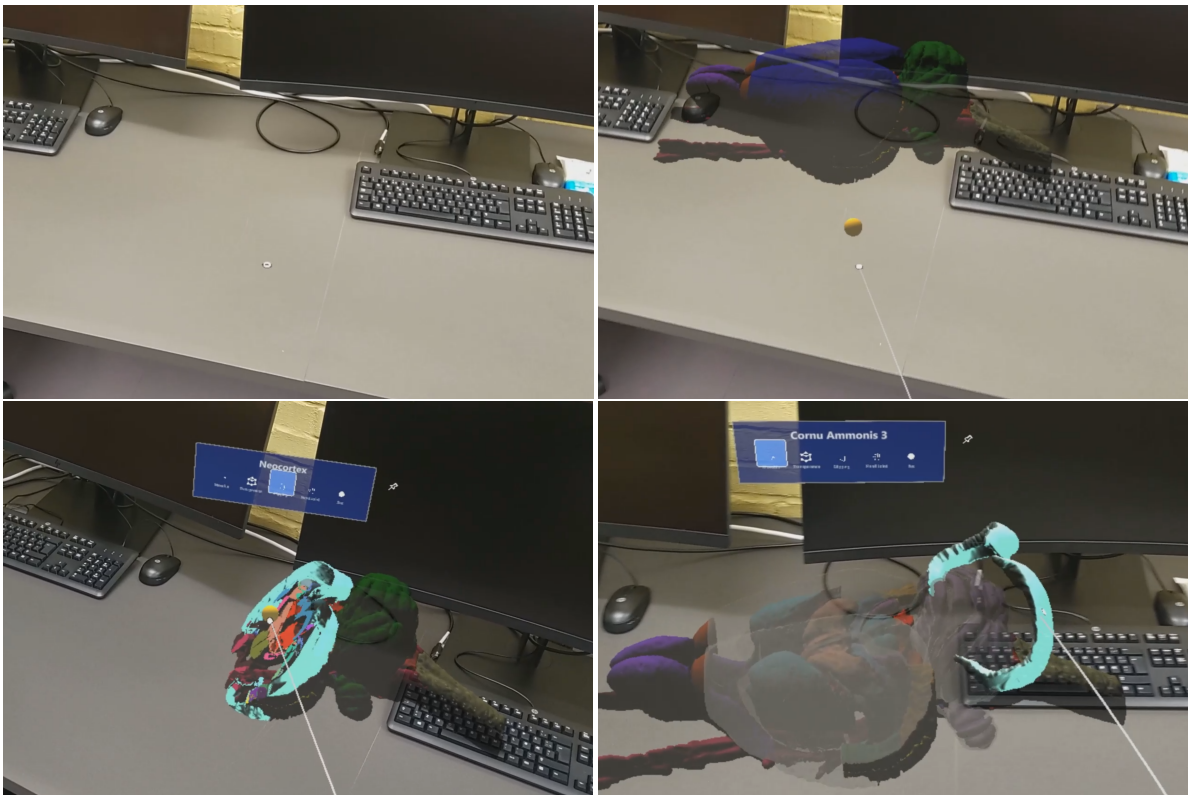


Figure 6.5: The first demo of the application, running on the HoloLens 2.

6.3 Next iterations: Continuing development

The continuation of the project will be explored further, but will focus on implementation of highlighted features and and high-level overview of the process, rather than a chronological log as in previous sections.
Shortly after end of the first iteration a demonstration of the application was done over video conference, with a pre-recorded YouTube video, demonstrating the features of the application. In addition a physical stakeholder meeting at St. Olavs Hospital was arranged where only the project researcher used the application with the HMD, but through live streaming the video feed from the headset and operational guidance from Dr. Menno Witter testing of dissection features were held. The application was at this stage nearly identical to what was seen at the end of the Iteration 1, with minor tweaks and bug fixes.
After this initial demonstrations stakeholders from the Kavli Institute were intrigued to see further development and had very positive sentiments toward the research project. Feedback gathered from this meeting included mainly two features, an ability to place brain structures back into the brain after deconstruction, basically to tidy up the brain after manipulation. Second, a list view for choosing which brain structures should be visible, this feature request was inspired by Eldens VRVisualizer which some of the stakeholders had previous experience with.

6.3.1 Snapping

The first of the described feature request to be able to put brain structures back into place. Snapping structures as magnets was suggested as a metaphor for the action.
This snapping effect was implement as a `MonoBehavior` called `SnapInPlace`, by storing the `initialPosition` of each snappable object and comparing the distance to this position with a given `threshold` distance at the end of each manipulation:

```
1 void OnManipulationEnded() {
2     if (Distance(initialPosition, structure.position) < threshold)
3         // set brain structure to initial position
4         structure.position = initialPosition;
5 }
```

This worked as expected, however no indication of the snapping behavior was given to the user and so it could be interpreted as an unexpected behavior when the brain structure just disappears when the user releases it. This issue was solved by having a semi-transparent shadow of the brain structure at its initial position colored green when snap-

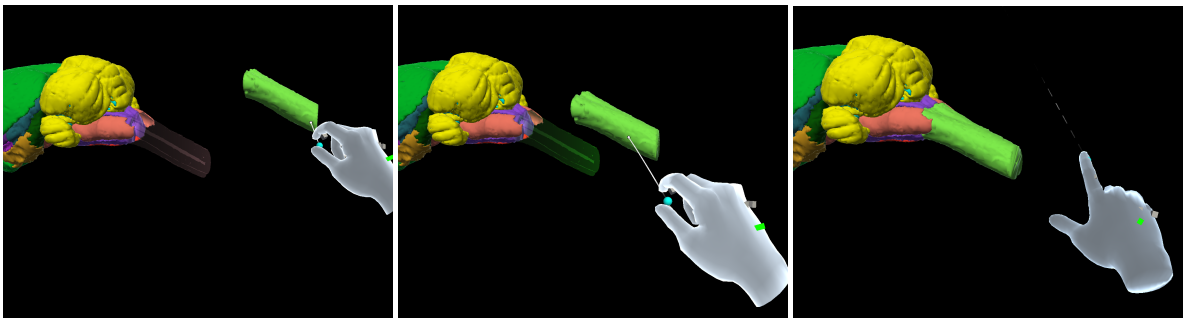
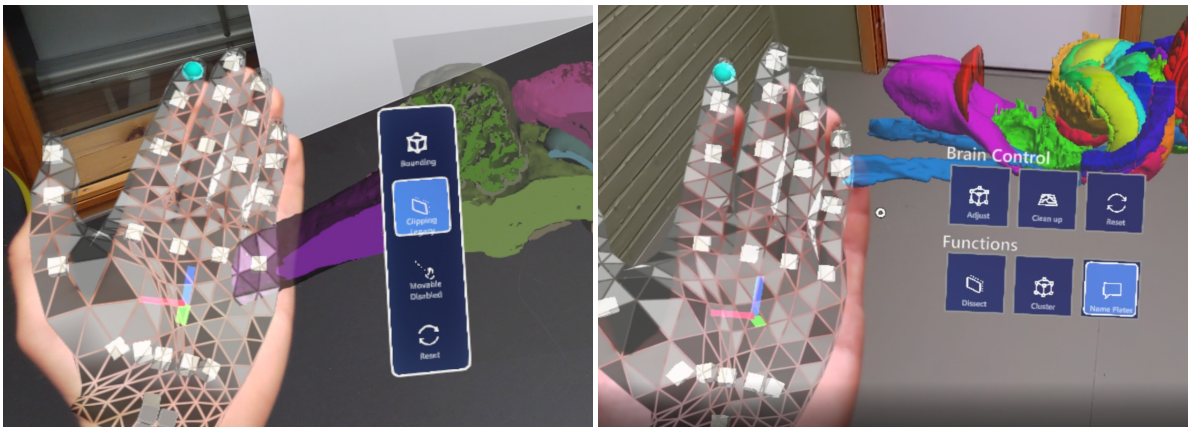


Figure 6.6: The complete snapping process. Right most image shows the brain structure snapping in place at release.

ping would occur at release and gray otherwise. Additionally, a audio effect was implemented such that a clicking sound as the structure snapping in place. As the snap implementation code above suggests, structures are snapped in place the same frame as the manipulating has ended. It was experienced with using *lerping* for smoother movement, however it was found to not give the same feeling of "snap" as just setting the position and playing a sound effect.

6.3.2 User Interface

User interfaces in augmented reality is still a rapidly developing field with few optimal solution. This project uses quite minimal UI for simplicity of use and development, but some UI is required i.a. for enabling features and informing the user. During this project, several iterations on menu design has tried and found lacking. The first iteration is seen in Figure 6.5 and is a combination of text content naming the selected brain structure and action buttons. This worked as for simple MVP purposes, but was far from optimal. It was tedious for users to click the buttons as they were hovering above the brain model.
This was solved by adding the action buttons to a menu which follows the hand of the user. This can be seen in Figure 6.7a which uses a prefab hand menu found in MRTK. The prefab was however deemed too inflexible and a custom hand menu was designed. This new design uses default MRTK `PressableButtons` grouped by context in horizontal `GridObjectCollections`, which again are in a vertical `GridObjectCollection` that uses a `HandBound` and `HandConstraintPalmUp` component to follow the users hand. In short the menu is designed by using simpler MRTK building blocks to create a more scalable hand menu, a early version of this menu design can be seen in Figure 6.7b. The horizontal button groups can be hidden or shown based on context, in the final product there are five button groups; *Brian Control, Tools, Admin Tools, Clustering Control, Dissection Tools.*



(a) Second iteration (b) Final iteration

Figure 6.7

The first two will always be visible, while the rest are context aware. Hand menus have one big issue, they are dependent on hand recognition. So devices without this feature can not fully utilize this type of UI, the solution used in this project is to simply disable the `HandConstraintPalmUp` for other device types, as in Listing 6.3, and let the menu hover in air. This is of course wildly suboptimal, and a better cross platform solution should be implemented in further research.

6.3.3 Info Board

Textual information about the brain structures was requested by stakeholders as a educational tool for student to use by them self or in groups. The idea was that lecturers could add text as they see fit and change it to an appropriate level for the intended user group. This feature was implemented based on the MRTK `Slate` with is an AR based floating text box, perfect to simulate a black board. In fact, if wished for in the future it would be trivial to make it look more as a black board by changing the color and the font to something more handwritten. The slate was customized by changing the content text reference from `InfoBoard` MonoBehavior when the a new brain structure was selected, this was done with a `UnityEvent`, a simple callback function, implemented to trigger when a new brain section is selected (this implementation is found in `NetworkBrainManager.cs`). The text descriptions for each structure is saved as a text file and is parsed in `InfoBoard.cs`, the text file uses a custom structure where "@" at the begin of a line indicates a new brain structure and "+" indicates a added images and and all other text is seen as descriptions for the last indicated brain structure. This works well enough, but an obviously more stan-

dard and readable approach would be to use a common text based serialization format like JSON or XML. In future development it should be looked into using one of these as both are well supported with built-in deserialization features in Unity with C#. The implementation does not support any run-time importing of this text even though the parsing is done at run time, functional it would thus be trivial to add. What has to be implemented for this to work however is either file exploring or some other way of accessing files, e.g. from network, see Future Work for more on this.

6.3.4 Clustering

By clustering brain structures based on the neurological attributes lecturers can visualize how complete compound structures operation and where they are located in the brain. Just as the `InfoBoard` this feature utilizes a custom text file parser to define each clusters name, brain structures and color pallet. Again JSON or XML would probably be a wise transition for these configuration files. When selecting the clustering action, the `Clustering` MonoBehavior iterates through all brain structures and enables or disables according to whether it is a part of the cluster. It also assigns a new color based on the color pallet defined in the configuration file. Color pallets are defined based on HSV color hues, with two numbers indicating start and end hue in degrees, then the brain structures are assigned colors uniformly the given sector of the color wheel. This was found to be a simple and reliable solution to color coding, but could with ease be argue as not being a intuitive solution for unaccustomed or non-technical users. For users not to loose spatial orientation when exploring the clusters a outline of the rat brain was added. This outline is a hollow version of the geometric brain model, which was created by manually selecting all outer vertices of the brain model using Blender, this is far from a perfect approach and gives a inaccurate and rough outline and a high polygon count, however if not for future performance issues the author sees no need to fix this.

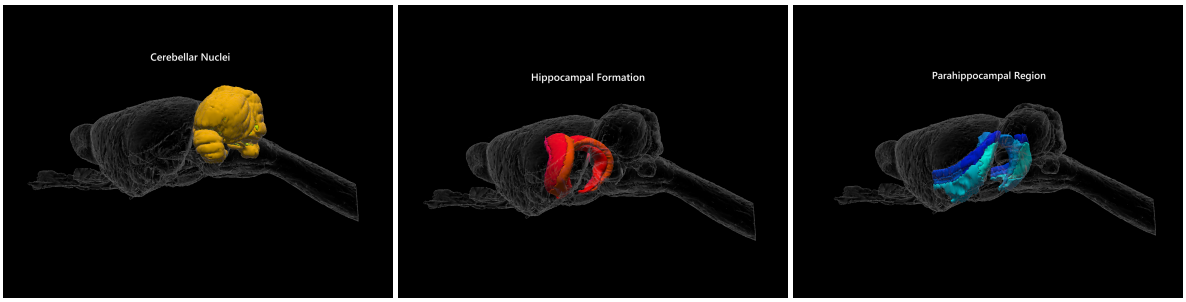


Figure 6.8: Clusters in Nevrolens.

6.3.5 Porting to Android

As MRTK is a framework specifically designed for HoloLens development, the HoloLens 2 naturally became the main focus of development and thus the application running on Android is a porting of the HoloLens version with small changes to somewhat accommodate the smartphone interface. It has however seen too little platform specific development and is in reality an incomplete product. It is however very much usable and can be seen as a great demonstration of the multiplatform capabilities of MRTK. Porting to Android was initially done by simply deploy the same application to Android, this is easily done in Unity with some MRTK tweaks which can be read up on in the documentation. While this works, some expected features are missing. For example, on the HoloLens 2 the user enables actions in a menu which is bound to their hand this is not possible in Android because the user interacts with the application not by moving the hand in space, but by touching the devices screen, therefore the application will simply disable the hand following feature on Android such that the menu buttons are static in space at all time. This is far from a optimal solution, but has worked for basic proof of concept.

```
1 if (runtimePlatform != GlobalSettings.SupportedPlatform.HoloLens2)
2 {
3     GetComponent<HandConstraintPalmUp>().enabled = false;
4 }
```

Listing 6.3: Basic example of separate features per device.

6.3.6 Volumetric dissection plane

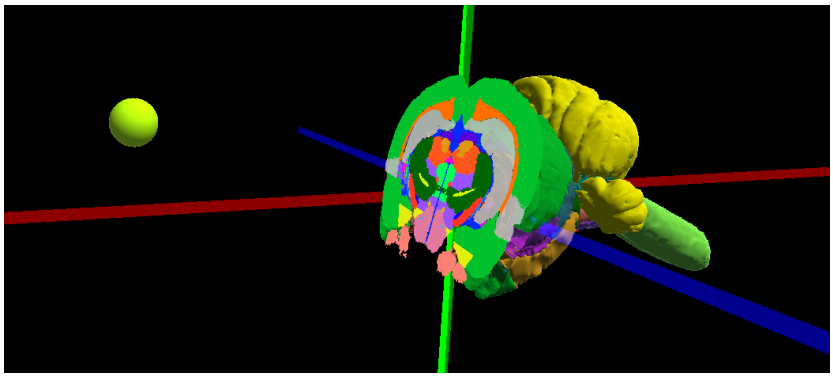


Figure 6.9: Volume dissection in Nevrolens

The basic implementation of clipping used for the dissection functionality leaves much to be desired. Clipping is a basic feature in most graphical libraries and shader languages, is

will simply discard vertices which does not meet some set predicate. Its foremost purpose in computer graphics is to ensure that vertices outside the viewing cone will not be rendered, this to optimize render time. The result of this process on the surface model being cut open, such that the clipped part of the model becomes a empty hole to the inside of the model, this is illustrated on the left image of Figure 6.10. This can be solved by rasterizing the cut plane, and this was experimented with in this project with some success, however this functionality is not afforded by MRTKs standard shader, and thus implementation required the use of a custom shader, which meant that many other useful features implemented in the standard shader also had to be reimplemented. This lead to abandoning the solution and focusing instead on overlaying volumetric data upon the plane of dissection. The Waxholm Space rat brain model is a volumetric data set, thus the goal with this

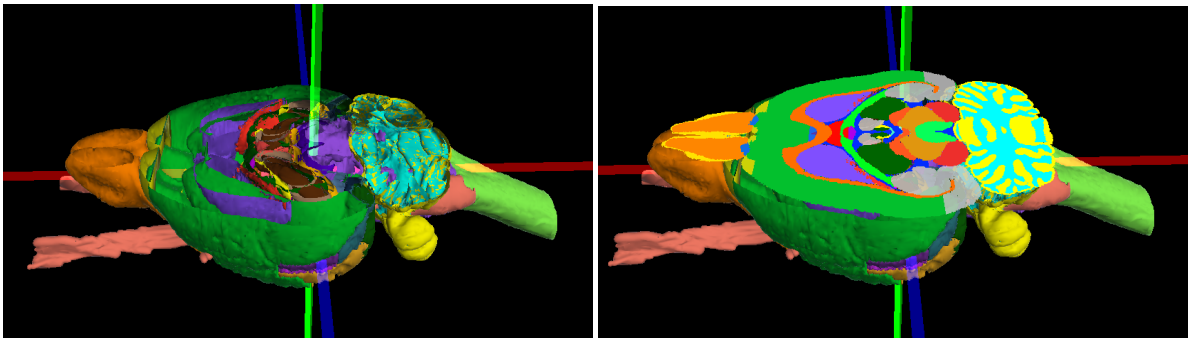


Figure 6.10: Dissection without and with volume plane.

feature was to visualize this data set as slices of the volume. The first step in this process was to format the volume in a manner that is usable in Unity and runnable on the target devices. The format of the WHS data set is the NIFTI format, this is a standardized format for neuroimaging data. With help from the library Nifti.NET accessing the `ushort[]` 3d data array was made possible. An issue arising was that Unity struggled with parsing the data set, and this would result in the application freezing and no data array, thus a basic *dotnet* console application called NevrolensNiftiTool was made to parse the NIFTI-file and output the 3d array as, of all things, a JSON object. JSON was chosen, after experimenting with `BinarySerialization`, but finding it to complex to use for data transfer, JSON was tested with success. The Unity scene `TextureBuilderScene` has a single purpose to generate a `Texture3D` resource from a JSON file. The MonoBehavior `BuildNifti3DTexture` takes the JSON file as input and outputs the data as a `Texture3D` Unity asset. This texture asset can than be used in later stages independent of the JSON file. With the data in the right format the first naïve implementation was using a MonoBehavior to apply a slice of the 3D data onto the `Texture2D` of a plane-object. This gave the desired effect of display-

ing a fixed slice of the volume, and by setting a free variable to one axis a sliding effect was made, however it was with the price of atrocious performance. For a robust implementation the feature would have to be excecated on the GPU as a shader.
For the intended behavior, it was critical not just to generate slices along predefined axes, but to generate slices along arbitrary planes in the volume. This is, in theory, quite easy to implement on the shader by making use *transformation matrices*. The idea is to map the UV coordinates of the default 2D texture onto the 3D volume. A basic implementation of this in the fragment shader can look like this:

```
fixed4 frag (v2f i) {
    // point from the 2D UV and the uniform float VariableSlider
    float3 uvw = float3(i.uv, VariableSlider);
    // tex3D samples the 3D texture NiftiTexture3D at point uvw
    fixed4 color = tex3D(NiftiTexture3D, uvw);
    return color;
}
```

This implementation will, just as the C# MonoBehavior, display slices along the Z-axis. However, by simply multiplying the UV coordinate with a `TransformationMatrix` instead of setting the Z-coordinate, the result is the implementation of the shader used in this application. This one change makes the whole mapping process dependent on the TransformationMatrix, which makes the path forward very flexible.

```
fixed4 frag (v2f i) {
    // point cast from product of TransformMatrix and UV as 4d point
    float3 uvw = mul(TransformationMatrix, float4(i.uv, 0, 1));
    // tex3D samples the 3D texture NiftiTexture3D at point uvw
    fixed4 color = tex3D(NiftiTexture3D, uvw);
    return color;
}
```

The affine transformations with matrices has built in support in Unity with the methods `Matrix4x4.Translate(Vector3)`,`Matrix4x4.Rotate(Quaternion)`,`Matrix4x4.Scale(Vector3)` and a short hand for the product of all of these the `Matrix4x4.TRS(Vector3, Quaternion, Vector3)`. With the described implementation of the shader the aim will be to set the transformation matrix corresponding with a transformation from the clipping plane surface to a slice in the volumetric texture of the brain model, for arbitrary positions of the clipping plane relative to the brain model. This will be done by transforming through several vector spaces. As is custom with matrix multiplication these transformations will be explained in reverse order.

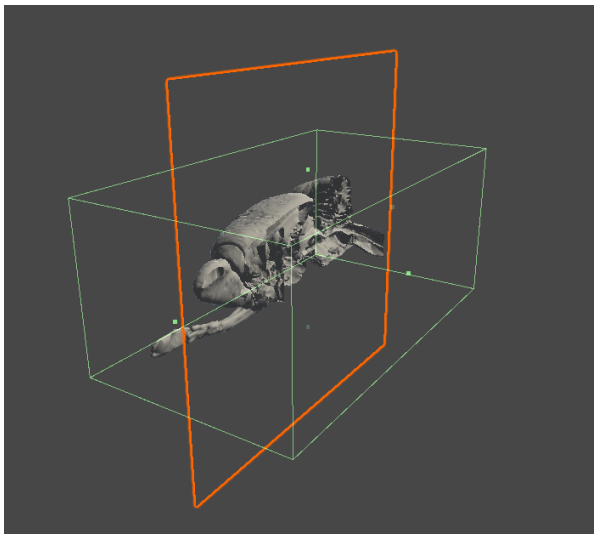


Figure 6.11: The orange dissection plane and the green 1x2x1 cuboid brain model space.

The `tex3D` texture sampler on the shader samples textures as a 1x1x1 cube, with its center in (0.5, 0.5, 0.5), thus the output of the matrix transformation script must reflect this by mapping the UV coordinates to this space, lets call this the *sampler space*.
The model space of the geometric brain model has its center at the local origin and is a 1x2x1 cuboid. This *brain model space* is illustrated as the green box in Figure 6.11. Additionally the geometric model is 180 degrees rotated on the X-axis relatively to the volumetric data, this is probably a result of swapped axis when creating either the volumetric data or the geometric data. Thus, the matrix transformation from brain model space to sampler space can look like this:

```
Matrix4x4 modelToSamplerMatrix = Matrix4x4.TRS(
    new Vector3(0.5f, 0.5f, 0.5f),// translate center from origin
    Quaternion.Euler(180, 0, 0),  // rotational difference in models
    new Vector3(1, 0.5f, 1)       // scale down from 1x2x1 to 1x1x1
);
```

Transforming further from brain model space to the dissection plane space, colored orange in Figure 6.11, is done in to steps; from brain model to game world and from game world to dissection plane space. The matrices also, as oppose to the previous matrix, has to be calculated in the `Update` loop as it changes when the brain model or dissection plane is manipulated. These matrices are both trivial to implement as they use `Matrix4x4.TRS` with the `position`, `rotation` and `lossyScale` from each models `Transform` component. This generates a model to world matrix, therefore the brain model matrix has to be Inversed. Lastly, a transformation is needed to offset the center of the dissection plane from

the graphical origin in the upper left to the geometric origin in the center, this `planeOffsetMatrix` is a constant translation of (0.5, 0.5, 0). The final implementation will look like this:

```
void Update() {
    worldToModelMatrix = Matrix4x4.TRS(
        model.position, model.rotation, model.lossyScale);

    worldToPlaneMatrix = Matrix4x4.TRS(
        plane.position, plane.rotation, plane.lossyScale);

    Matrix4x4 transformation = modelToSamplerMatrix
                * Matrix4x4.Inverse(worldToModelMatrix)
                * worldToPlaneMatrix
                * planeOffsetMatrix;

    // moves the matrix to the shader as TransformationMatrix
    renderer.SetMatrix("TransformationMatrix", transformation);
}
```

The resulting plane will however be in grayscale as the `Texture3D` contains single `ushort` values per voxel, how this was fixed will be explored in Coloring the brain.
A last note is that there is a small, jet very noticeable, discrepancy between the volume model and the geometric model such that they do not completely overlap, Figure 6.10 shows this to some extent. The researcher attributes this to the fact that different versions of the Waxholm Space brain are used in the generation of each volume and the lack of control in how the geometric model was created. Thus, when a single new data set is used for both geometric and volumetric models this issue should disappear.

6.3.7 Coloring the brain

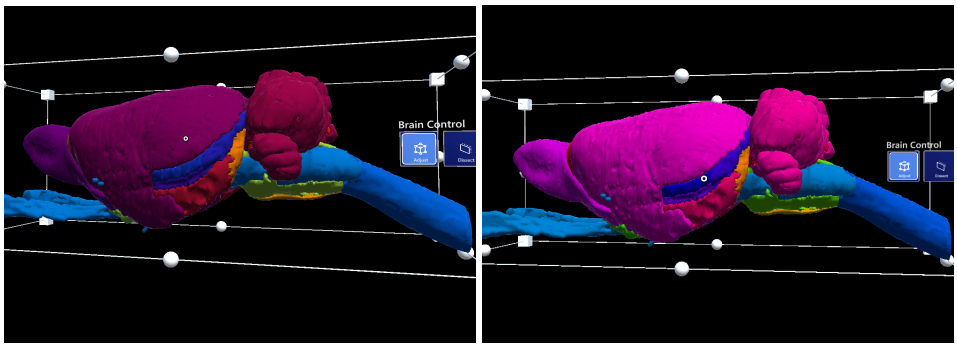
Throughout this thesis the reader will see the brain visualized in a number of different colors. Many images seen are of completely gray brains this are not and have never been part of the application, and are just for demonstration, this is how the model is seen in the Unity editor because color is set at runtime. The first iteration of coloring the brain structures, described in section 6.2, was to use random colors. The goal of this was simply to visually distinguish the separate brain structures. An improvement was added which biased the random generation such that it generated lighter more saturated colors, but otherwise this was the state of brain colors until networking was implemented. With multiple users randomized colors was not appropriate anymore, because users would not be able to describe structures in terms of their color (unless the master client generates col-

ors and distributes them to all, this approach was not taken). The properties needed from the colors was to be visually distinct and to be deterministically generated. This was implemented by sampling colors with uniform distance on the circumference of a circle of radius one inside the RGB-cube, like this.

```
1 float r = (Mathf.Sin(v) + 1) / 2;
2 float g = (Mathf.Cos(v) + 1) / 2;
3 float b = (-Mathf.Sin(v) + 1) / 2;
```

Listing 6.4: With v ranging from 0 to 2π.

After some time the developer realized that smarter people than him already had solved this issue, and that the exact color properties needed was found in the HSV color model. The *hue* channel in the HSV color model operates in the same manner as the code above when the other channels are constant. Thus by setting *saturation* and *value* to their max value and setting hue to the v from above mapped between zero and one, the resulting colors were basically the same only more saturated[1]. The difference can be seen in Figure 6.12.



(a) RGB implementation (b) HSV implementation

Figure 6.12

After demonstrating the application to neurologists, the developer was made aware that the Waxholm Space model has a predefined color model, and thus all the hard work was for naught. The final implementation consists of a parser of the LABLE-files in the Waxholm Space data set which contains color data. Then matching the name of the brain structure to its label and applying this color. The resulting colors can be seen in subsection 6.3.6, e.g. in Figure 6.10. Though the HSV implementation is not used anymore for the brain coloring, the properties of this color model was found useful for other part of the

[1]The RGB implementation had a radius of 1, while $\sqrt{3}$ would (probably) give max saturation.

application and is used to generate distinct colors for each user when in multiplayer and in clustering as seen in Figure 6.8.

Coloring the dissection plane

To color the dissect plane appropriate the same label colors as above as used, however the coloring of the structures in the plane happens on the shader so the color data had to be moved to the GPU. A 1D texture was used as a lookup table for the shader. The values in the volume `Texture3D` explored in subsection 6.3.6 are identifiers for the segmentation or brain structure the voxel is part of. These identifiers are defined in the label file, and thus by setting colors at the index of the identifier in the lookup table, the correct color could be sampled with `tex1D` in the shader.

```
1 lookupTableLength = Labels.Last().identifier;
2 // Unity doesn't support 1D texture so it's 2D with height=1
3 lookupTable = new Texture2D(width: lookupTableLength, height: 1);
4 foreach (var label : Labels)
5     lookupTable.SetPixel(x: label.identifier, y: 0, label.color);
6
7 // moves the values to shader
8 renderer.SetFloat("LookupTableLength", lookupTableLength);
9 renderer.SetTexture("LookupTable", lookupTable);
```

The code above shows how the lookup table can be implemented. The fragment shader explored in subsection 6.3.6 can now be extended to sample these colors:

```
1 fixed4 frag (v2f i) {
2     float3 uvw = mul(TransformationMatrix, float4(i.uv, 0, 1));
3     fixed4 voxel = tex3D(NiftiTexture3D, uvw);
4
5     // maps 0 - 1 alpha channel, to 8-bit numbers
6     float identifier = voxel.a * 256;
7     fixed4 color = tex1D(LookupTable, identifier / LookupTableLength)
8     return color;
9 }
```

The volume texture is of the format `Alpha8` instead of the default `RGBA32`, this means that all its data is in the alpha channel and has a bit depth of eight, this could be done because all the data in the volume is the range 0 to 115, the range of identifiers in the label file. Thus, line 6 in the shader code above uses only the alpha value.

6.3.8 Final Iteration

Networking optimization

Chapter 7

Deployment

The application in this research has been development using Microsoft's Mixed Reality Toolkit and has followed the platforms best practices and recommendations, thus, any issues arising when going through the following section should be answered in Microsoft's MRTK documentation. The deployment process for HoloLens devices are based on sideloading of the application compiled and built with Visual Studio Community 2019, this is the standard way of both building and deploying for HoloLens. For Android the principle of sideloading is also used, but here Unity is responsible for compilation and the build process. The application as

Installation of Nevrolens

This section will be a guide for installing Nevrolens on HoloLens 2 and Android from packages hosted on GitLab. By following the instructions for HoloLens 2, this should also work for HoloLens 1, but has limited testing on that platform as it was not a focus of this research.

Deploy to HoloLens 2

1. **Go to the release page**
   Found here: `https://gitlab.stud.idi.ntnu.no/olevra/nevrolens/-/releases`

2. **Choose a release**
   Preferably the topmost and latest. This research ends on is version 0.3.3.

3. **Download the HoloLens zip package**
   Under *Packages* click on the package for HoloLens (1 and 2) to download it.

4. **Unzip file**
   Open the downloaded ZIP-file and extract it.

5. **Open the Windows Device Portal for HoloLens**
   Guide by Microsoft: Using the Windows Device Portal

6. **Install appxbundle**
   Under *Views* / *Apps* click `Choose File` and locate the APPXBUNDLE-file inside the folder extracted from the ZIP-file. Then click `Install`.

Deploy to Android

1. **Go to the release page on your Android device**
   Found here: `https://gitlab.stud.idi.ntnu.no/olevra/nevrolens/-/releases`

2. **Choose a release**
   Preferably the topmost and latest. This research ends on is version 0.3.3.

3. **Download the Android APK-file**
   Under *Packages* click on the package for Android to download it.

4. **Open the downloaded file.**
   Your device will ask for your permission to install an application from a unknown source. Accepting this, the device will start installing the application.

Getting started with developing the project

This section will briefly explain how to set up the project for development, this differs from deployment in that the goal is to be able to continue development of the project from within Unity and with supporting tools. This is the process the current developer uses when developing from a new computer. Having completed this set up deployment of the application can be done, directly from Unity for Android and through Visual Studio 2019 for HoloLens devices.

Requirements

1. Git and Git LFS

2. Unity 2019.4 LTS
   When installing add: *Android Build support*

3. Visual Studio 2019 (Community or other)
   When installing add: *Universal Windows Platform development, USB Device Connectivity, Game development with Unity*

Type the following commands in to the *Git Bash*, it will clone the repository, and download all files which are to large for git with *Git LFS*.

```
$ git clone "https://gitlab.stud.idi.ntnu.no/olevra/nevrolens.git"
$ cd nevrolens
$ git lfs install
$ git lfs fetch --all
```

When this has completed, open Unity Hub and locate and add the `nevrolens` repository just cloned as a Unity project. Open it with Unity 2019.4, it is critical that this version is used for MRTK to work properly. First time opening this project will take some time, when it is done the project is ready. Deployment for HoloLens 2 can be done by building (ctrl+shift+b) the Unity project with the configurations in Figure 7.1 and opening the outputted `Nevrolens.sln` solution in Visual Studio 2019 and running the solution for `Remote Machine` while the HoloLens 2 device is connected by USB. Refer to the MRTK documentation for OTA deployment through Wi-Fi. Android deployment is done by simply choosing Android in Unity's build configuration view and clicking *Build And Run* while the Android device is connected by USB and is set to developer mode. Make sure to follow the MRTK documentation on which version of ARCore etc. that should be installed in the project, found here.

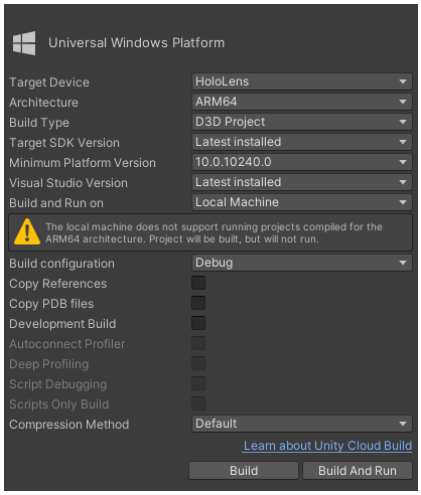


Figure 7.1: Unity build configurations for HoloLens 2.

Chapter 8

Testing

This chapter will explore how testing has been accomplished in this research project, both day-to-day software testing, but also feedback from stakeholders and formal user testing. Because of the current COVID-19 pandemic, physical meeting and user testing been done sparingly, and have at times been impossible carry out, in fact the fall semester did not see any physical testing by users not affiliated with the VRLab at NTNU Dragvoll. What follows are the approach and structure of testing, research results and feedback from the testing sessions will be expanded further in the Results chapter.

8.1 Software testing

During development, software testing has been done unstructured, mostly by way of regular deployment and on-device feature testing. This is admittedly not the most comprehensive testing system and can not guarantee intended behavior as systems are combined and restructured, in the same way as a test suit with *unit testing* and *integration testing* could. But it allows for more rapid development, less overhead for a single developer and can not be said to have meaningfully limited the research project. A case could be made for a testing suit being helpful in the continuation of the research as new developers will have a concrete indication of intended behavior, this has and could not have a high priority as a development goal, because of limitations in development time and the demand for creating research results.

8.2 User Testing Precautions

Consent

Every user tester where handed a consent form at the beginning of each session. This assured the privacy rights of the test persons, and gave approval to use the findings of the test session in this research. The consent form can be found in **??**.

Hygienic Measures

Due to the COVID-19 pandemic, many new precautions have been enacted to prevent the spread of virtual pathogens. The most effective measure has undoubtedly been the limited number of testing sessions, even so what follow are the precautions taken during those physical meetings. Firstly, the general national guidelines of social distancing and clean hands were kept. Between each new person using a head mounted HoloLens 2, the headset was placed in a *Cleanbox*, this is system specially design to disinfect AR and VR devices utilizing UVC radiation to destroy viruses. It claims to kill Covid-19 viruses with 99.999+% efficacy[1]. Further, contact points and buttons on the device are swiped with alcohol-based sanitizer. The reason this step was not enough and the UVC box was needed is because of the fragile nature of the lenses of the AR device, it is generally recommended not to touch the lenses, and alcohol would naturally not be good. On Android devices this was however the main disinfection method as cellphones could be completely wiped. Lastly, the user would ware a hygienic mask under the headset which reduced contact between the user and the headset. All in all these actions in combination with strict adherence to the national guidelines of physical distancing, washing and sanitizing of hands and waring of face masks were done to prevent spread of the virus.

8.3 Stakeholder meetings

Throughout the project, there have been multiple meetings with stakeholders from the Kavli Institute at St.Olavs. These have been a combination of physical meetings and virtual video conferences using Zoom. When meeting virtually the research prepared one or more video demonstrations captured on-device, either HoloLens 2 or Android, which were uploaded to YouTube for convenience. When meeting physically either the researcher or the stakeholder would use the application with live view enabled such that the other could

[1] https://cleanboxtech.com

see and comment on the usage. Afterwards, the progress was discussed and evaluated and feedback was given on usability and features. Primarily, the resulting feedback was in the form of feature request, and general technical in nature.

8.4 User Testing

During the final stages of the research project, two user testing sessions were held. This both gave useful insight into what the application did well and badly. Many of the issues discovered in the first test session were improved upon, this process is explored in subsection 6.3.8. The last test session was at the end of the development phase on this project and was meant mainly to gather data for research. No further development has happened after this point and as such the last test session gives an accurate picture of the resulting software product of this research project, both in its proficiencies and its limitations.

Figure 8.1: Tester using HoloLens 2, being lectured by the neurologist whos watching their actions through live feed from Device Portal.

First testing session

The first user testing had three participants, all being medical student. Two first year and one third year student. Additionally, one neurologist from the Kavli Institute was present in the role both of stakeholder in the research project and also as lecturer to the student

whom all took courses held by the neurologist. This session did not test the Android application at all and only focused use with the HoloLens 2, this created a challenge for collaboration testing as only two HoloLens 2 devices were available.
The session began by having the participants take a multiple choice test to survey their knowledge of the rat brain anatomy. This test was a standard test from course work in medical courses held by the stakeholder neurologist, and is meant to measure the short term learning from a single lecture. Therefore the exact same test was taken after the end of the session when the participants had used the research application.
After taking this test the participants were freely testing the application somewhat unstructured, and familiarizing them self with the usage of AR devices which none of the participants were accustomed with. Throughout the usage a pattern naturally evolved such that the neurologist lectured students with headset on. This started when the neurologist and a single student both used the application through the headset, the neurologist guided the student and showed them the different brain structures and explained there purpose. The one on one nature of these lectures were broken up by having the lecturer view the live stream of two students through a Windows desktop computer, this happened mostly because of time constraints as it lent itself to both of the remaining students having the lecture rather than just one. Both situations were insightful as observations of practical use of the application, and were not designed, nor intended by the researcher.

Second testing session

The second test session marked the end of this research projects development stage, thus it was a test of the final product in this research. Therefore, in addition to this test session being used for data gathering, it can also be seen as a final evaluation of the application.
The participants in this session were five medical students and two computer science student. The session started as the previous one by having the participants filling out a consent from and taking the same initial knowledge test. The session was organized around the principle use case found in the previous test session where the neurologist lectured the participants, while himself seeing the application ran through live feed from the student participants. This session tested with both HoloLens 2 as well as Android devices and thus two students could use the HoloLens 2 devices while another two could use Android smartphones.

Figure 8.2: Neurologist lecturing Android test users.

Chapter 9

Results

The results in this research project are mainly gathered from the final test session, which was held after the last development stage, thus the results reflect the state of the application at the end of the research.
The test sessions had two separate formal data gathering strategies:
A knowledge test taken by the test participants before and after the session, this was meant to demonstrate learning value in short term time frame. Such a simple short term test could support the claim that the artifact can be used as an educational tool,

9.1 Learning value

9.2 Usability

Chapter 10

Discussion

10.1 Limitations

10.2 Results

10.3 Performance

The main limiting factor for the performance of the Nevrolens application is the polygon count for the models used. The count for the models uses are around 300,000 for the complete model at decimation ratio 0.08 and an hollow outline model with decimation ration 0.4 at about 350,000 polygons, used in clustering. The counts are higher than what is generally recommended for HoloLens 2, naturally this results in an application which runs well under the best recommended performance quality criteria which the non-functional requirements of this research project set out to meet. The application will hover around 40 to 50 frames per second while in normal use and drop to around 30 to 35 while rendering the volumetric dissection plane, the recommended quality criteria lists 60 fps as a goal for *Best* performance, but the application follows the *Meets* criteria, which state that:

> *The app has intermittent frame drops not impeding the core experience, or FPS is consistently lower than desired goal but doesn't impede the app experience.*

Thus, the application can be said to meet the quality criteria set by Microsoft. Throughout testing, framerate was genuinely not a detectable issue, no user tester did at any time commented on low performance and even testers experienced with HoloLens and AR technology did not unprompted notice the low framerate while using the application. This could be the result of the somewhat blurry display of the HoloLens 2, or simply that targeting 60

fps is far less critical in AR application, than in on VR devices. On tested Android devices performance has not been of any concern, running at a consistent 60 fps all the time. The teste device is a Samsung Galaxy S8, which was released in 2017 way before the HoloLens 2, and does in fact have a slower processing unit than the HoloLens 2 does. Answering why this performance gap exist will only be speculation, and will not be attempted. In conclusion, the performance of the application is at an acceptable level and does not impede the user experience, if that were to happen, either by increased load from demanding features or porting to less performant platforms, a natural and easy solution would be to scale down the brain model further.

10.4 Hardware limitations

HoloLens 2 display technology

Android interaction

Android spatial locking

10.5 Missing data set

The brain model data set used in this research, which as described in section 6.1 was imported from VRVisualizer, is missing a number of brain structures. The model consists of 29 separate structures while the Waxholm Space Rat Brain model it is based on included as least 75 structures. This is of course a huge discrepancy, but it is not as critical as it could seem as most of the omitted structures are very small and would be impractical to handle within AR. Upon questioning Elden has explained that the reduction was done to decrease computational time as the model was meant for a proof of concept. As this is still the case the reduces number of brain structures is still not a critical issue, however if this research project were to be used as an actual educational tool this would be of most dire need for fixing. Another reason this issue has not seen any need for fixing is that there will be a new version of the Waxholm Space brain model released shortly, and stakeholders at Kavli are interested in the use of this model in further research, thus creation of a new brain model has been deferred to Future Work. A increases in structure count could however result in increased polygon count which would have to be accounted for then scaling the resolution of the surface models.

10.6 Human neuroanatomy

Describe/explore the feasibility to implement the system for Human neuroanatomy education.

Chapter 11

Conclusion

Both in this research and others augmented reality technology has shown great promise in communicating complex three dimensional data, and results from this research shows the potential for using this ability for education of neuroanatomy to medical students. Finding from this research also support the use of AR in remote lecturing, and shows that lecturing through AR devices has educational potential.
Used as an educational device the HoloLens 2, which has seen the bulk of development resources in this project, is limited from a price and availability perspective. Hopefully, this issue will resolve with better technology or expansion into more available platforms.
This research has established that collaborative learning between head mounted display devices and Android smartphone is archivable, and to some extent that this collaboration has a educational value. However, further research into cross platform collaboration is called for, with the possibility for adding platforms which are deemed accessible for wide adoption.
As a whole the software application is still in a proof of concept phase, which a research project of this scope naturally produces. With further development the author is of the opinion that the application, or an AR app of similar caliber, can be of great value to medicine students, if developed for public release.

11.1 Future Work

This section will lay down suggestions for how to further the research project. These are based on the authors experience with the project and on discussions with the neurologists and medical students testing the application.

Import new data sets

As described in section 2.3 the WHS rat brain is under continuous development and the the near future there will be released a forth version of the brain model, with improved delineation. It is a high priority wish from the Kavli Institute to use this brain model in the application.
To import a WHS brain model as a geometric model is a complicated process, which has been explained by Elden in Appendix B. The process of exchanging the geometric model used now with a new one however is trivial, but time consuming. A considerable improvement to the application would be the ability to drop in a new model, preferable at run time such that the application wouldn't need to be build and deploy for each model change. This would enable future WHS versions to easily be added and even other models like human brains could be switched between.
Another essential feature to reduce the need for new builds is to enable configuration in-app. The application uses three different text files which saves the configuration of clustering, infoboard and labels, all of these should either be possible to upload at run time or configuration with settings UI in-app. A logical step could be to use the JSON format for these files instead of the custom parsing done now.

Improve networking

A critical improvement to the application would be to have the initial scene of the application give the user an option of creating a room, joining an existing room or playing offline. The application as of now will automatically behind the scene, join the existing room or create one if there is none. This gives users less control but more frictionless when testing collaborative features. In a full scale application the user control would probably be preferable.
The networking solution in the application has a considerable amount of bugs and unexpected behavior, this is probably something that is difficult to completely circumvent, but an effort to restructure or fix this should be done.

Voice chat

When collaborating in-app remotely the ability to talk to each other would be a great feature, which for a full scale application would be a high priority. This can be done with by using Photon Voice for PUN 2.

Platform specific input and UI

There are few limitations in which platforms the application can run on. However, each platform comes with it's own needs for specific input types and UI. The application has been always been developed with HoloLens 2 as the highest priority and that can be seen in most choices taken conserving UI and input handling. Bettering the user experience on Android and even Windows or WebGL (the application is buildable for both, but needs input handling to be usable) should be a priority. Feedback from user testers gave indication that the augmented reality in the android version did not provide an improved user experience and thus disabling camera and spatial features in Android could be seen as an improvement, which could be extended to a desktop application. A future version of the application running on the web, with WebGL, would probably be possible and further the goal of accessibility. If use of AR with Android developed further, looking into the possibility of implementing stereoscopic rendering should be a priority. The MRTK framework does not provide this option as of writing, but Android support is still an experimental feature.

Appendix A

Acronyms

AR Augmented Reality

MR Mixed Reality

XR Extended Reality

VR Virtual Reality

FPS Frames per second

HMD Head-mounted display

WHS Waxholm Space

GPU Graphics Processing Unit

SDK Software Development Kit

MRI Magnetic Resonance Imaging

DTI Diffusion Tensor Imaging

NIFTI Neuroimaging Informatics Technology Initiative

Appendix B

A geometric model of the rat brain

This is a section from Elden (2017), the master thesis about VRVisualizer the VR application this project is loosely inspired by. The section explains how Elden extracted a geometric model of the rat brain from the medical models which is high fidelity volumetric data. I have included it in this report because it gives insight into a specific solution to a problem I face and it explains how a resource I use in this project was created.

5.2 Exporting segments of a rat brain atlas as geometry for the Rat Brain model

The geometric meshes used for the rat brain model were extracted from a volumetric and segmented atlas. IKT-SNAP was used to export each segment of the brain as an STL file. These geometric meshes were then opened in Blender3 to be converted to OBJ or FBX files. 3DS Max imported the models and performed all modifications made to the geometry and structure. ITK-SNAP requires three files to segment and label the models; the atlas and a segmentation file, both stored as NII files, and a LABEL file for the labels. When all files are loaded the program lets the user select a segment to export and generate a geometric hull along the boundary of the segment. Due to instability experienced with 3DS Max using all 16 GB of RAM available on the computer used for development, Blender was used to first convert the files to FBX files. These FBX files caused no issues when imported into 3DS Max. Since these meshes were too detailed, they needed to be reduced and transformed in 3DS Max. The meshes were reduced such that the entire model consisted of 4.5 million triangles. Most of the meshes had to be transformed such that each segment was where it should be inside the model. For some reason the exported meshes were of sev-

eral relative scales and heights and a lot of manual work went into moving and scaling the meshes to match the volumetric model seen in ITK-SNAP. Properly processed, the model was exported as an FBX file and sent to UE4.

Bibliography

Elden, M. K. (2017). Implementation and initial assessment of vr for scientific visualisation: Extending unreal engine 4 to visualise scientific data on the htc vive. Master's thesis, University of Oslo.

Hawrylycz, M., Baldock, R. A., Burger, A., Hashikawa, T., Johnson, G. A., Martone, M., Ng, L., Lau, C., Larsen, S. D., Nissanov, J., Puelles, L., Ruffins, S., Verbeek, F., Zaslavsky, I., and Boline, J. (2011). Digital atlasing and standardization in the mouse brain. *PLOS Computational Biology*, 7(2):1–6.

Milgram, P. and Kishino, F. (1994). A taxonomy of mixed reality visual displays. *IEICE Trans. Information Systems*, vol. E77-D, no. 12:1321–1329.

Moro, C., Štromberga, Z., Raikos, A., and Stirling, A. (2017). The effectiveness of virtual and augmented reality in health sciences and medical anatomy. *Anatomical Sciences Education*, 10(6):549–559.

Oates, B. (2006). *Researching Information Systems and Computing*. SAGE Publications.

Papp, E., Kjonigsen, L., Lillehaug, S., Johnson, G., Witter, M., Leergaard, T., and Bjaalie, J. (2013). Volumetric waxholm space atlas of the rat brain for spatial integration of experimental image data. In *Front. Neuroinform. Conference Abstract: Neuroinformatics 2013*.

Papp, E. A., Leergaard, T. B., Calabrese, E., Johnson, G. A., and Bjaalie, J. G. (2014). Waxholm space atlas of the sprague dawley rat brain. *NeuroImage*, 97:374 – 386.

Sielhorst, T., Feuerstein, M., and Navab, N. (2008). Advanced medical displays: A literature review of augmented reality. *Journal of Display Technology*, 4(4):451–467.

Singh, V. and Kharb, P. (2013). A paradigm shift from teaching to learning gross anatomy: meta-analysis of implications for instructional methods. *Journal of the Anatomical Society of India*, 62(1):84 – 89.

Sommerville, I. (2011). *Software Engineering*. Pearson Education, Inc., 9th edition.

Toga, A. W. and Thompson, P. M. (2000). 40 - image registration and the construction of multidimensional brain atlases. In BANKMAN, I. N., editor, *Handbook of Medical Imaging*, Biomedical Engineering, pages 635 – 653. Academic Press, San Diego.

Wish-Baratz, S., Crofton, A. R., Gutierrez, J., Henninger, E., and Griswold, M. A. (2020). Assessment of Mixed-Reality Technology Use in Remote Online Anatomy Education. *JAMA Network Open*, 3(9):e2016271–e2016271.