Nordic Neuro Lab

Unity

VisualSystem HD

Unity setup with the VisualSystem HD



0 Table of Contents

1	Introduction	3
1.1	VisualSystem HD, ResponseGrips and SyncBox	3
1.2	Virtual Reality	5
2	Project design	6
2.1	Game type	6
2.2	Game design	6
	Core	6
	Features	6
	Game mechanics	7
3	Detailed design	8
3.1	Architecture	8
3.2	Creating the environment in Unity	10
	Settings scene	10
	Game scene	11
	Achieving the 3D effect using dual cameras	12
3.3	Game Functionality	13
3.4	Input controller	14
3.5	Preview screen	14
3.6	Optical distortion	16
	Solving pincushion distortion	16
	Barrel distortion shader	16
	Determining the distortion amount	17
4	References	18

Abbreviations:

NNL	NordicNeuroLab
VSHD	VisualSystem HD
VR	Virtual Reality
HID	Human Interface Device
UI	User Interface

Preface

Over the spring 2020, NordicNeuroLab (NNL) collaborated with two students, Arja Sivapiragasam and Enah Lanto, from the Western Norway University of Applied Sciences on the completion of their bachelor's degree in computer science (1). Their thesis work focused on creating a sample project on how to use Unity with the NordicNeuroLab VisualSystem HD (VSHD). The project created a 3D city environment with a memory and search task, which has support for barrel distortion correction, as well as the capability to adjust the viewing aspect to the interpupillary distance.

The full project and an executable file for testing the application can be accessed on GitHub via the following link. Note that the project has been updated with new assets in Unity 2022.3.

https://github.com/nordicneurolab/vshd-unity-example

NordicNeuroLab advises that one should have programming experience when planning to build an experiment in Unity. NNL is not in a position to provide any form of support for such software development.

The following describes a summary of the thesis to provide a better understanding of the capabilities of the VSHD when used with graphics engines such as Unity. Additionally, it serves as documentation for the sample code provided on GitHub.

1 Introduction

This work creates a 3D/Virtual Reality (VR) environment in Unity which could be integrated with the NordicNeuroLab VSHD for providing visual stimulation when performing fMRI scans on a subject or patient. By creating a 3D/VR environment which mimics real-life interactions with objects and their surroundings, the subject may become more immersed in tasks, which will provide more realistic and accurate brain activations (2).

1.1 VisualSystem HD, ResponseGrips and SyncBox

The VisualSystem HD is a solution for presenting 2D and 3D visual stimuli during a functional MRI procedure. The hardware consists of goggles (Figure 1), which display visual stimuli, as well as featuring the functionality to record eye movements during fMRI scans. During the fMRI scan, the technician can present the visual tasks to the subject who can then respond through the NNL user feedback system, i.e., the ResponseGrips (Figure 2). The ResponseGrips have two response buttons for each hand (thumb and index finger) and are connected to the ResponseGrip Interface which should be then connected to the stimulation PC via a USB port on the side of the NNL SyncBox (Figure 3). For this project, the SyncBox is used in the Human Interface Device (HID) mode, which can be set using its on-screen menu. Note that SyncBox v3.2.1 and v3.2.2 do not support the HID mode.



Figure 1: NNL goggles as a part of the VSHD system.



Figure 2: ResponseGrips with two buttons (thumb and index finger) on each of the left and right handles.



Figure 3: NNL SyncBox and its on-screen menu.

In this project the VSHD serves as the screen output of the VR environment and the ResponseGrips are used for navigation in the VR environment.

1.2 Virtual Reality

Virtual reality can be described as a digital environment that is generated using a computer to create a realistic environment that can be experienced and interacted with by the user (3) via incorporating technologies such as:

- Stereoscopic displays
- Motion tracking hardware
- Input devices

Here the VSHD is used to provide stereoscopic display (4). The ResponseGrips serve as the input device and Unity game as the platform.

Figure 4 is a use case diagram illustrating the interactions between the users and the VR system. A user within a use case diagram is a person who interacts with the system. Here, the users are the spectator/technician and the test subject/patient. The user journey starts with the technician executing the software and configuring the settings. The technician starts the game and can monitor what is seen in the VSHD goggles using a preview screen. The test subject begins to play the game as soon as the technician starts the game. The technician is then able to monitor the game as a spectator throughout the duration of the game (2).

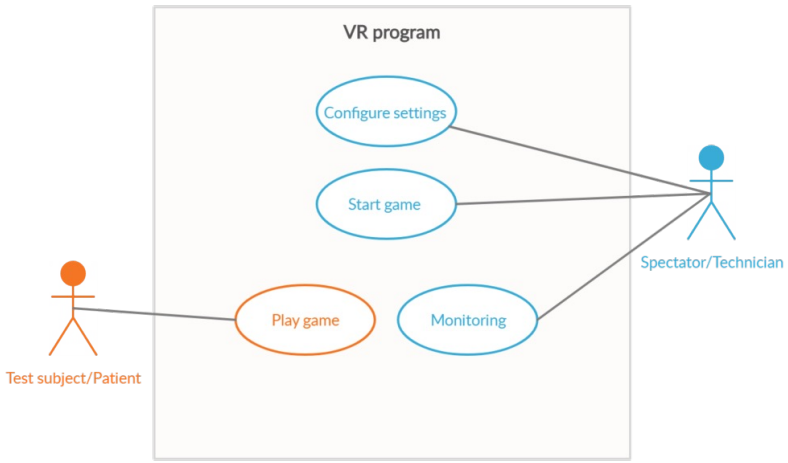

Figure 4: A use case diagram of the VS system.

The virtual world could simulate a natural environment such as a garden or park. Here, the designed game in Unity contains a gamification element where the player must click and collect objects in a virtual city in order to progress in the game.

When running the Unity game through the VSHD goggles, which are formed from two independent OLED displays, there should be a possibility to display the game on a third monitor as a preview screen. To this end, two cameras should be placed within the scene. This implementation is explained in more detail in Section 3.

2 Project design

The specifications of the NNL hardware provide the possibility of developing a stereoscopic 3D game in Unity. The setup of the NNL hardware is in a way that the VSHD goggles are two separate monitors and the ResponseGrips work as a keyboard with A, B, C, D as the key inputs (see Section 3.4).

2.1 Game type

There are three concepts that can make an observer feel like they are in a specific environment: immersion, presence, and reality trade-off (3). Among these factors immersion is implemented in the current project, via a "find the object" type of game where a picture of an object is shown on the screen and the goal of the player is to find this object. In an fMRI environment, however, one should compensate for the movement restrictions in the MR, and this is the reason that the NNL ResponseGrips are used here for navigation. Potentially one could also use eye movements as input for navigation using the eye-tracking cameras of the VSHD. This is, however, not the focus of the current guide.

Note that the drawback of implementing such a game in fMRI is that the player is exposed to motion sickness. This is due to the player having the ability to virtually navigate around the environment which could cause a mismatch between what is seen and what is physically felt (3) (5).

2.2 Game design

Here the game design is based on the basic theories presented by Menard & Wagstaff (6 pp. 31-43). In the following sections, the terminologies "core", "features", and "game mechanics" are described and then an explanation of the game flow will be presented.

Core

As defined by Menard & Wagstaff (6 p. 32), "The core is basically one defining and unifying statement that describes your game. You can think of the core as being like a thesis statement for a research paper". The core of this game is immersive memorization intended to trigger processes in the brain which regulate context-dependent perception, cognition, and movements (5) (7). The sole character of the game is the player. A city serves as the environment of the game while the player navigates through it to accomplish an objective.

Features

"If the core is like the thesis statement of a research paper, the feature set is the collection of main ideas from each supporting paragraph. Each element in the feature set describes a main part of the play in your game, and each should definitely relate back to the core idea," as explained by Menard & Wagstaff (6 p. 33). The game consists of the following features:

- Virtual city environment (see Figure 5).
- First person perspective.
- Memorization activated by showing an image of the objective for only a few seconds.

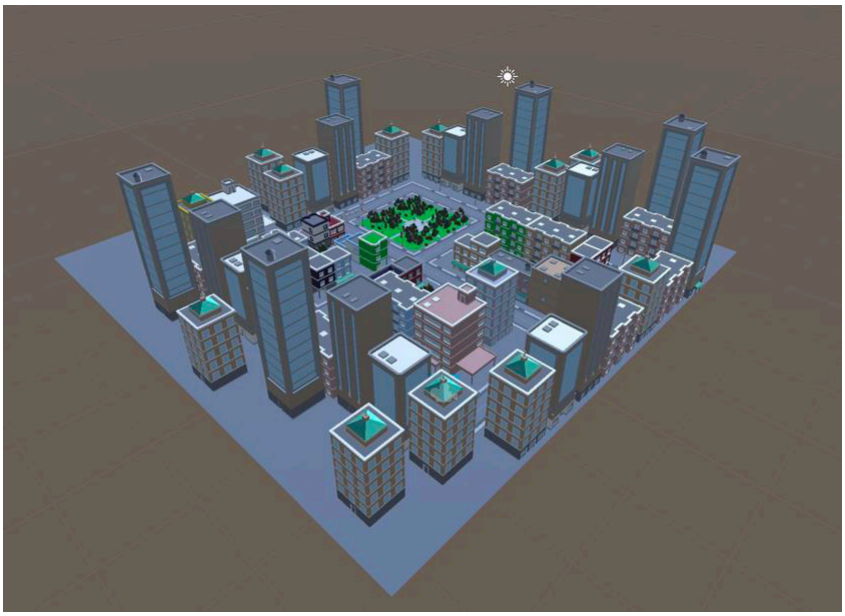

Figure 5: An aerial view of the virtual city environment.

The first two features support the core of the game by offering a virtual platform that contributes to the player feeling more immersed. The last feature contributes to the core of the game by serving as the visual stimuli that triggers the relevant brain activations.

Game mechanics

Menard & Wagstaff (6 p. 32) further explain, “The most basic building block of the game is a mechanic, a rule or description that governs a single, specific aspect of play”. The following explains the flow of the game as well as the interaction of the player with the game.

This game consists of four targets. The general objective is to find a traffic sign that is shown to the player. When this sign is found, the player receives a new traffic sign to find. The player is able to navigate within the game environment using the ResponseGrips and can move forward, backward as well as to the left and right.

An example of a game session is as follows:

The player is placed in a city as the game starts. Figure 6 shows an example where the objective is to find a No Left Turn sign. After the image of the target sign is shown for five seconds, the player can start looking around the city to find it. The player must move close to the target traffic sign in order to hit it.



Figure 6: A screenshot of the compressed left and right displays when an image of the target traffic sign is shown.

A green check mark is shown when the player hits the No Left Turn sign (Figure 7). If the player hits a wrong sign, a red cross is shown instead (Figure 8). Hitting the correct traffic sign increases the score of the player (see Figure 9) and the objective gets completed.

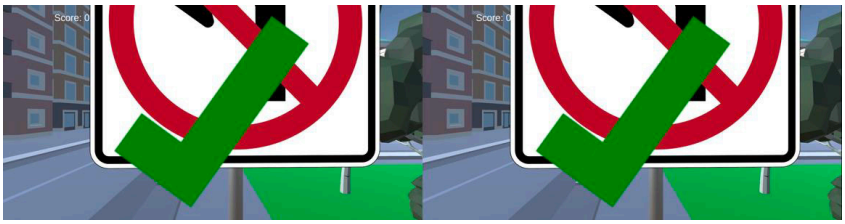

Figure 7: A screenshot of the compressed left and right displays when the player hits the correct traffic sign.

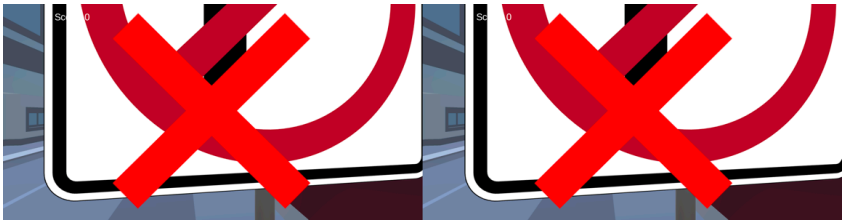

Figure 8: A screenshot of the compressed left and right displays when the player hits a wrong traffic sign.

After the green check mark is shown, a new traffic sign, e.g., Exit Right, is shown on the screen as the new target sign (Figure 9). Figure 9 also shows that the score of the player increases when the No Left Turn sign is found. The game continues until all four targets are found.

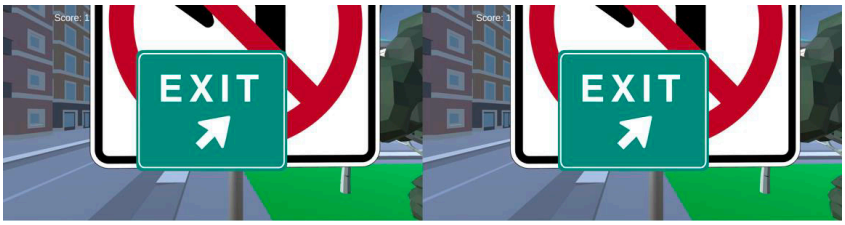

Figure 9: A screenshot of the compressed left and right displays when the player finds the No Left Turn sign.

3 Detailed design

3.1 Architecture

In the early stages of game development, the main components are mapped in a global system view, as shown in Figure 10.

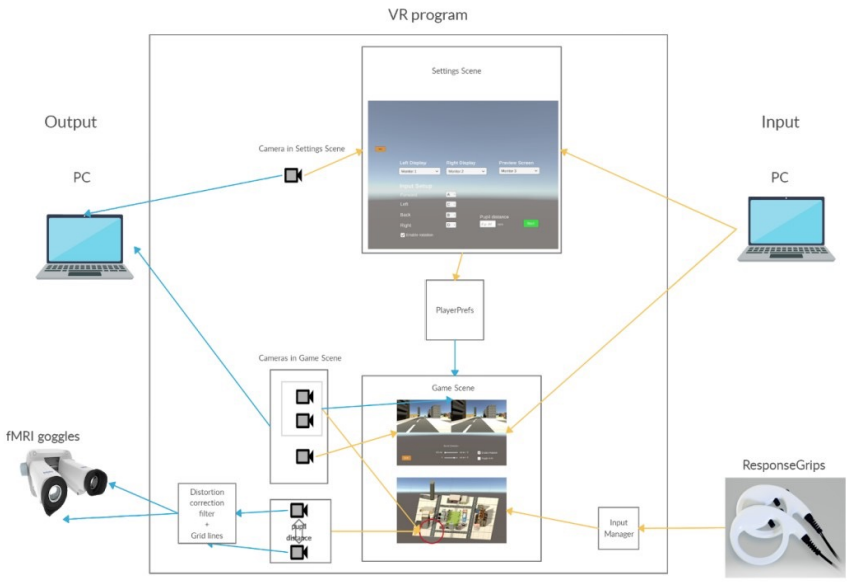

Figure 10: Architecture of the entire system, illustrating the interactions between the different components of the system.

Figure 10 shows a full overview of the entire system consisting of both software and hardware. The box in the middle represents the entire game or VR program created in Unity. To the left of this box is the computer running the program, which also serves as the output monitor for the Settings scene that is displayed when the program is first executed (see Section 3.2).

The Game scene is displayed for the subject in the left and right displays of the VSHD goggles. Details about the Game scene can be found in Section 3.2. The two cameras at the bottom of the diagram point towards the area of the game environment which are shown on the VSHD. The left and right cameras for the left and right eyes have a slight offset compared to the pupil distance between the eyes. This will create a stereoscopic 3D effect as explained in the “Achieving the 3D effect using dual camera” section below. A distortion filter is additionally applied to these two cameras accounting for the pincushion effect on the VSHD displays (see section 3.6). The NNL ResponseGrips, shown on the right, is used to navigate within the game environment.

On the diagram, there are also three cameras pointing towards the preview screen. Two of these are copies of the left and right eye cameras mentioned above so that the technician can monitor what the subject is viewing. The final camera points at a user interface canvas for the technician to be able to control in-game settings such as the amount of barrel distortion or quitting the game.

The UML diagram shown in Figure 11 illustrates the basic ideas behind the game structure which is the back end of the program code. There are two options for executing the game, either through the command line or by running an executable file, as shown at the top of the diagram. Here we concentrate on the second option, i.e., running the .exe file. Each box is titled with the name of a class. The main classes are the Settings Scene and Game Scene. The parameters, needed to initiate the game, are set in the Settings Scene (see section 3.2). The Game Scene includes two subclasses, titled Player, and ScoreKeeper, which give the player the ability to move, find the objectives and increase the score. The third subclass gives the technician the ability to control the game settings. The text inside the boxes represents the operations that are to be implemented within the various classes.

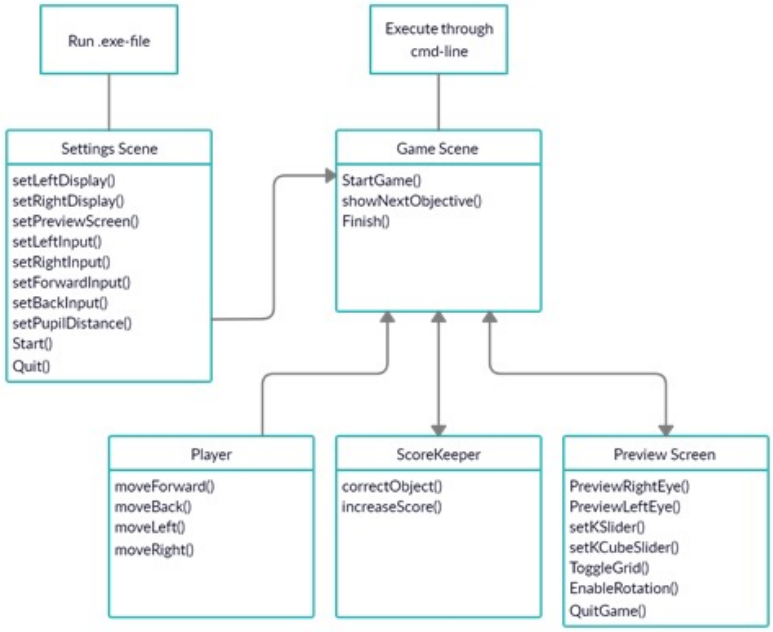

Figure 11: UML diagram.

3.2 Creating the environment in Unity

The game was created as a 3D project in Unity 2019.3 and built by choosing Windows as the target platform from the PC, Mac & Linux Standalone build settings. It was later updated with new assets in Unity 2022.3. The game created in Unity was separated into individual files called “scenes”. Each scene contains the corresponding objects of the game such as user interface components, cameras, 3D assets. All the components are placed in the scenes through the drag-and-drop functionality or by creating them through the Unity user interface. The project consists of two main scenes: the Settings scene and the Game scene.

Settings scene

The user experience of this game starts with the Settings scene (Figure 12). This is the main user interface of the game where the technician can select which monitor displays which image between the left eye, right eye, and the preview screen. The technician is also able to set the keyboard or controller inputs as well as the pupil distance of the test subject. More information about setting the pupil distance can be found in the rest of this section.

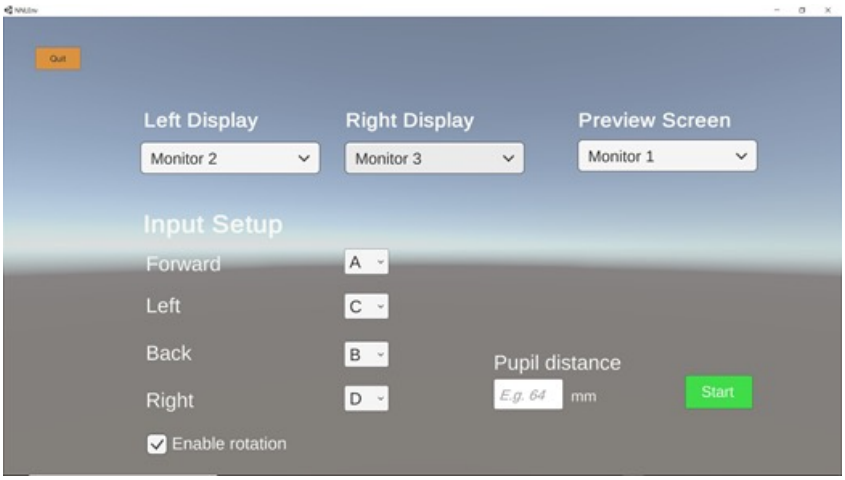

Figure 12: The Settings scene for specifying player’s preferences before starting the game.

The technician provides input through Unity's default user interface components with drop-down menus consisting of strings for the displays and controller inputs and by entering the pupil distance in millimeters as an integer in an input box. These user inputs are stored as "player's preferences" in the game using the PlayerPrefs functionality in the provided script.

If the user leaves the input fields empty, the default values are set to A for forward movement, B for backward movement, C for left- and D for right-side movements. The "Enable Rotation" checkbox changes the left and right movement from simply moving in the left/right directions to rotating the player in the left/right directions. The default pupil distance is 64 mm based on the average distance between the left and right pupil on the human body (8). In Figure 12, the preview screen is set to be shown on Monitor 1 by default while the left and right eye displays are set to the second and third monitors. This monitor allocation, however, might be different on your system.

Game scene

Clicking on the "Start" button in Figure 12 loads the Game scene. For the Game scene a city environment was created as shown in Figure 13. Prefabs, materials, and textures, available from the Unity Asset Store, were utilized to shape the environment. All assets were scaled intuitively based on system testing to ensure a realistic experience. A 3D plane was placed as the foundation and road prefabs were placed on top of it. A 3D capsule was used as the body of the player to mimic the stance of a person.



Figure 13: An aerial view of the city environment.

The landscape of the city was divided into blocks separated by a road network. Each block in the city was populated with several 3D components such as apartment complexes, lamp posts, and nature assets. In one of the blocks a park was created with trees, bushes and stones with a walkway from all four sides meeting in the middle at a square (Figure 14). The grass was created using a green plane with groups of premade grass objects. All the buildings placed around the city were attached with a collider, so the player cannot walk through them (see Section 3.3).



Figure 14: An in-game view of the park, showcasing the 3D assets.

When a camera is placed within a scene, it is automatically rendered inside a skybox creating the appearance of a sky around the city environment. Placing directional light in the sky adds to the realism of the game environment, causing the 3D objects of the game to cast shadows (Figure 15). The skybox updates the direction of the shadows as the player rotates.

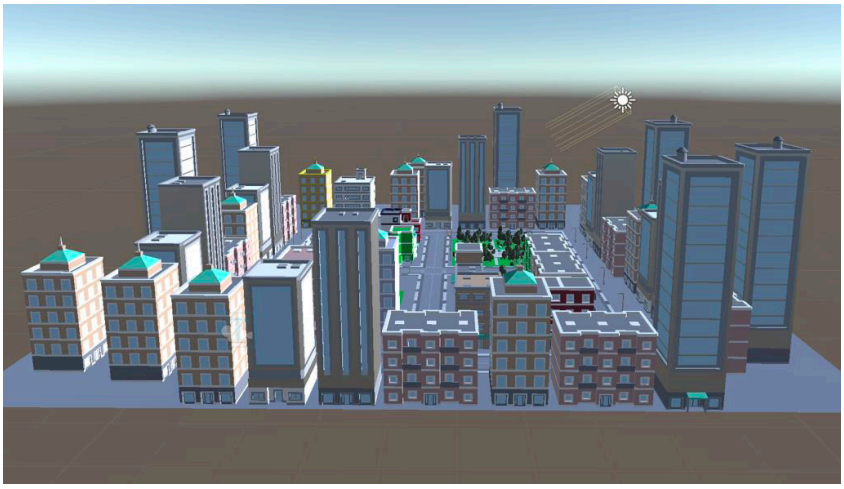

Figure 15: A view of the game environment. The directional light is placed in the sky on the right side of the city, causing the game objects to cast shadows.

Achieving the 3D effect using dual cameras

The dual screen output is created through Unity's multiple camera feature. This is implemented by placing two cameras at the same point in the environment (Figure 16). These cameras are then assigned to each of the displays and activated by the "ActivateAllDisplays.cs" script available in the provided project. The same script is used to assign the target displays through the "player's preferences".

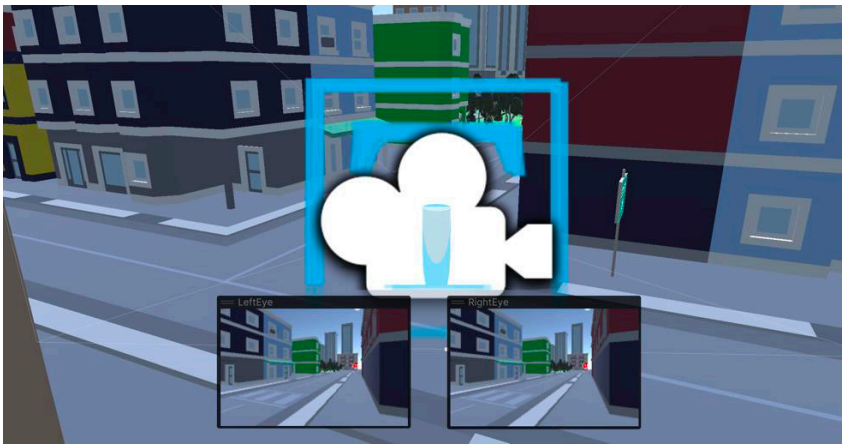

Figure 16: A scene preview in Unity showing both camera objects placed on top of each other and a preview of each camera view.

The stereoscopic (3D) setup, which comes with most commercial VR devices, displays a separate image for each eye, where one image is slightly offset from the other to simulate parallax (9). Here the pupil distance (specified through the "player's preferences") is scaled to the Unity environment, and on loading the scene, the positions of the left and right eye cameras are adjusted to provide the correct offset. Setting the pupil distance manually according to each subject, to match the inter-pupil distance (IPD) on the goggles, gives the most comfortable 3D view for the player. If the IPD is set incorrectly it could lead to breakdown of binocular fusion, reduced sense of depth, and chromatic aberrations, resulting in distorted 3D effect and sharpness (10).

3.3 Game Functionality

Due to the setup of the dual output functionality, images are displayed on the screen using sprites, as opposed to User Interface (UI) elements. The images shown on the goggles, including the images of the traffic sign and images of the green check mark and red cross, have a sprite texture and are stored in a game object. The game object containing the sprite is activated and deactivated in accordance with the state of the program. For example, when the player finds the correct road sign, the game object containing the green check mark is activated and appears on the screen for a set amount of time before it is deactivated again and disappears. To regulate the program's behavior, a finite state machine was created (Figure 17). The flow of the visual cues is also controlled by this state machine. The images are only shown either during the transition from "Show Objective" state, to "Start finding" state, or during the transition from "Selected a traffic sign" state to the next state, which is dependent on the validation done by the program.

To show the score (see Figure 9), a TextMesh Pro game object is used, as it is presented as normal UI text but behaves in the similar way as a game object. TextMesh Pro does not need a canvas to be rendered, which is the best solution to display text to the goggles in the dual output setup of the application.

Colliders are set to all the buildings in the city. A collider defines the object's shape and the possibility of interaction with the player. To differentiate the interaction between the traffic signs and the player from the interaction with the other objects, triggers are set to the traffic signs. These triggers are set to give feedback to the player. To avoid the player from falling off the plane, colliders are used as boundaries. A class in Unity Engine called Character Controller is set as a component of the player. This class is inherited from the collider class and is in the shape of a capsule. Thus, the 3D capsule that represents the player gains the ability to interact with other colliders.

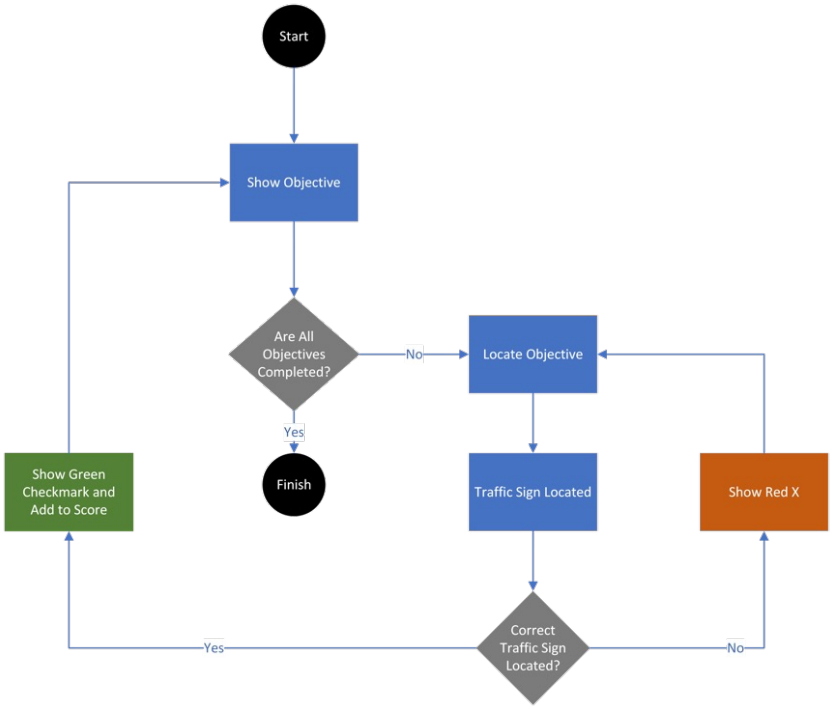

Figure 17: Game system's finite state machine

To reduce the chance of motion sickness, the acceleration speed of the player was set to a constant low value. This minimizes the mismatch between the virtual motion and what the player in reality senses. The player's score is presented on the screen and acts as a rest frame. The rest frame is a part of the scene that a viewer considers stationery, and they perceive other motion relative to it (3).

3.4 Input controller

Since the inputs from the four buttons on the ResponseGrips behave in the same manner as normal keyboard inputs of keys A, B, C and D (provided that the SyncBox is used in HID mode), the inputs are handled through scripting in accordance with the variables that are saved in the player's preferences. This means if a key input is detected and is equal to one of the keys saved for navigation, the player will move in the corresponding direction. By default, the player can rotate using the input buttons when moving sideways. The technician is however able to toggle the rotation option on the Settings screen before the game starts (see Figure 12). If the ResponseGrips are not available, regular keyboards can be used instead.

3.5 Preview screen

Figure 18 shows the preview screen, where the technician or spectator can monitor the player's movement during the game as well as being able to control the settings for barrel distortion (see Section 3.6). This screen is created by placing a third "Preview Screen" camera pointing towards the UI canvas along with the two displays showing the preview of the VSHD goggles.

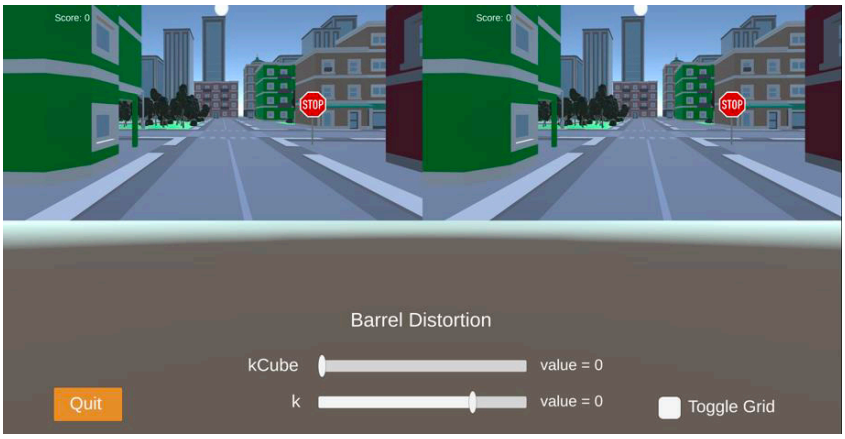

Figure 18: Preview screen

To create the preview of what is shown on the VSHD displays, two additional cameras (Left Preview, Right Preview) are positioned on top of the cameras set for the VSHD displays. The depths for these cameras are then assigned with the numbers 1, 2 and 3 according to the order in which the cameras are to be rendered (Figure 19). This is to make sure that the two game previews are rendered on top of the full preview screen.

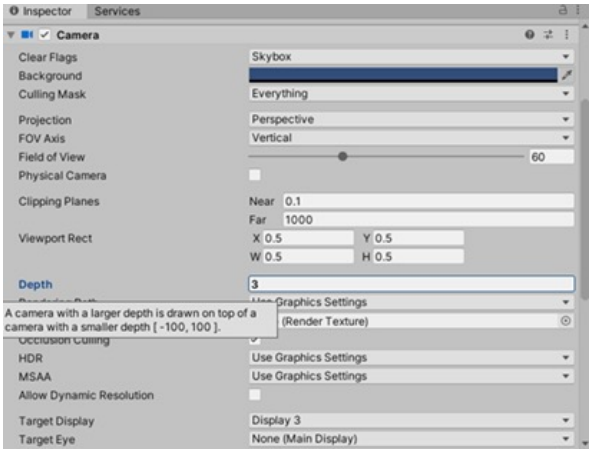

Figure 19: The Camera settings in the Unity Inspector view.

In order to show the left and right preview displays side by side, the Viewport Rect coordinates, which are available in the preview camera´s Inspector (Figure 19), should be changed to fit the screen (Figure 20). The x and y coordinates range from 0 to 1 with 1 representing the full length of the screen. The left camera starts at position x = 0, y = 0.5 and the right camera at x = 0.5, y = 0.5. The width and height are both set to 0.5 to cover 1/4 of the screen.

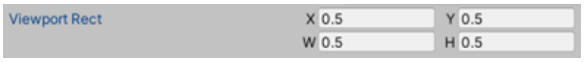

Figure 20: The Viewport Rect settings in the Unity Inspector view for the right preview camera.

3.6 Optical distortion

Solving pincushion distortion

The VSHD has a small but unavoidable pincushion distortion effect due to an increasing magnification away from the optical axis. This results in lines in the image curving outwards from the center (Figure 21a). The inverse effect, known as barrel distortion, (Figure 21b) can therefore be applied to the image before it is sent to the VSHD to compensate for pincushion effect, resulting in an undistorted image (Figure 21c).

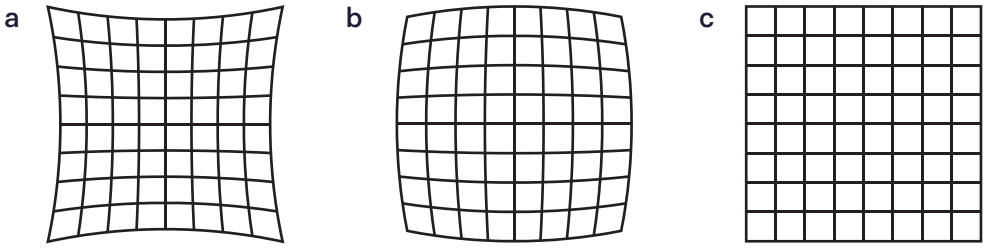

Figure 21: Pincushion distortion (a) and barrel distortion (b) of object (c) (12)

Barrel distortion shader

In Unity, shaders are used to manipulate the objects. A shader is a small script which can be applied to an object within the scene to modify the object by applying mathematical calculations to the graphics data such as pixels, vertices, or textures (Figure 22). Shaders are the most common method for adding different lighting and special effects to an object or scene.

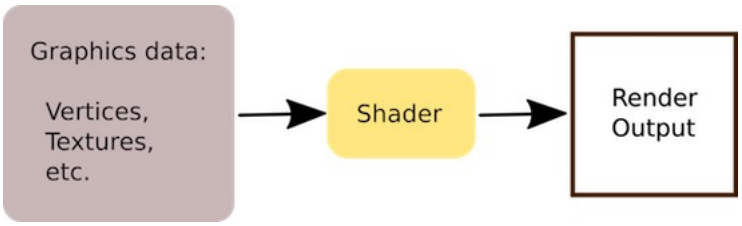

Figure 22: An illustration of how shaders work in Unity using the graphics data as input and rendering the output (11).

Here, a shader is applied to the two cameras for the left and right VSHD displays using a post-processing script. The script is based on an algorithm for barrel distortion using the vertices as graphical data and modifying their position and changing the outer pixels to a black color. After the shader is applied, the images appear distorted on a monitor as seen in Figure 23. However, inside the VSHD goggles they appear without any distortion.

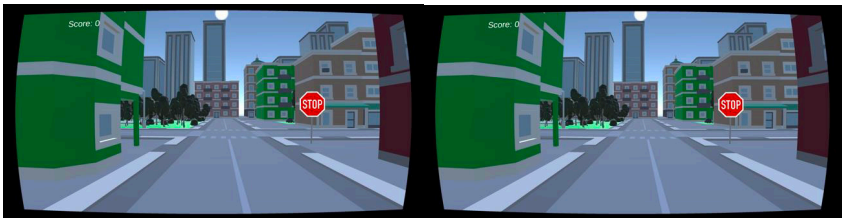

Figure 23: The displays of the left and right eyes after applying the barrel distortion.

Determining the distortion amount

In order to determine how much pre-barrel distortion should be applied to correct for the pincushion distortion effect, a slider was created in the preview screen to adjust the k value which determines the level of correction (see Figure 18). The optimal level of correction may differ from system to system and with larger differences in distance between the subject´s eyes and the VSHD lenses.

A grid can also be applied in front of the camera as a reference to observe how the image is changing. A toggle function to remove or apply the grid has been then implemented using a UI toggle component and scripting (see Figure 18). The k value should be adjusted while the subject is looking at the gridlines shown in Figure 24. Start with a k value at 0 and increase it until the lines appear the straightest. A larger k value will increase the pre-barrel distortion, a negative k value will result in pre-pincushion distortion which only increases distortion in the final image. Figure 24 demonstrates the effect for two different distortion coefficients. On the left the distortion coefficient k is set to 0.5, and on the right k is set to -0.5. These were created by adjusting the slider on the preview screen shown in Figure 18.

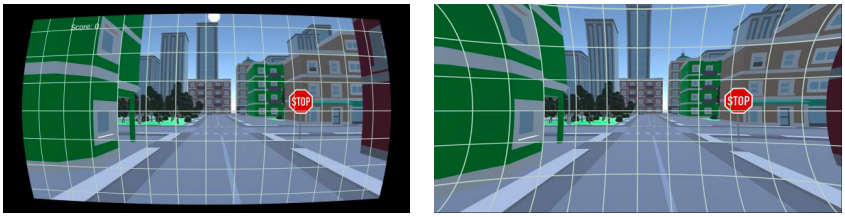

Figure 24: Images displayed to the VSHD with a positive (left) and negative (right) distortion coefficient.

4 References

1. Sivapiragasam, Arja og Lanto, Enah.

   Visual Stimulation for fMRI Scanning Using VR Technology. 2020

2. Bohil, Corey J., Alicea, Bradly og Biocca, Frank A.

   Virtual reality in neuroscience research and therapy. Nature Reviews Neuroscience. [Internett] 3 November 2011. https://doi.org/10.1038/nrn3122.

3. Jason, Jerald.

   The VR-Book: Human Centered Design for Virtual Reality. Waterloo : Morgan & Claypool Publishers, 2015. 978-1970001129.

4. NordicNeuroLab.

   Display Setup for VisualSystem HD. [Internett] https://www.nordicneurolab.com/resources#user-guides.

5. Gauthier, Baptiste, et al.

   Virtual Reality platform for funtional magnetisk resonance imaging in ecologically valid conditions. Association for Computing Machinery. [Internett] 8 December 2021. https://dl.acm.org/doi/abs/10.1145/3489849.3489894.

6. Menard, Michelle og Wagstaff, Bryan.

   Game Development with Unity 2nd Edition. s.l. : Cengage Learning PTR, 2015. 978-1305110540.

7. de Gelder, Beatrice, Kätsyri, Jari og de Borst, Aline W.

   Virtual reality and the new psychophysics. the british psychological society. [Internett] 28 May 2018. https://bpspsychub.onlinelibrary.wiley.com/doi/10.1111/bjop.12308.

8. US Army- Medical Department.

   Notes on Eye, Ear, Nose, and Throat in Aviation Medicine.

9. Parisi, Tony.

   Learning Virtual Reality: Developing Immersive Experiences and Applications for Desktop, Web, and Mobile. Sebastopol : O'Reilly Media, 2015. 978-1491922804.

10. Hibbard, Paul B, van Dam, Loes C.J. og Scarfe, Peter.

    The Implications of Interpupillary Distance Variability for Virtual Reality. [Internett] 2020. chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://peterscarfe.com/papers/Hibbard_etal_2020.pdf.

11. Hocking, Joseph.

Introduction to Shaders in Unity. [Figure] 2020. https://www.kodeco.com/5671826-introduction-to-shaders-in-unity.

12. Foundry.

Camera Matching. [Internett] https://learn.foundry.com/modo/content/help/pages/rendering/camera_matching.html

Nordic Neuro Lab

CE

VisualSystem HD

Headquarters
NordicNeuroLab

Møllendalsveien 1
N-5009 Bergen
Norway

Regional Office
NordicNeuroLab, Inc.

234 W. Florida St., Suite 202
Milwaukee, WI 53204
USA

NL-100-0121-01