Reading and Interactivity in Virtual Reality

Kevin Chandra

Undergraduate Honours Thesis
Faculty of Science (Computer Science)
Ontario Tech University

Supervisor:
Christopher Collins

Abstract

Reading in virtual reality can be difficult due to the current Head Mounted Display (HMD) hardware limitation. Different HMDs have different resolution specifications on their display. Lower resolution can cause small texts from a distance illegible, and close text can be blurry or out-of-focus due to the Vergence Accommodation Conflict. This paper aims to provide a method by using Rapid Serial Visual Presentation to overcome the problem of illegible text and explore the possibility, interactivity, and freedom of virtual reality.

List of Content

1. Introduction

1.1 Motivation

Virtual reality (VR) devices are becoming more and more accessible to the public with products such as Valve Index, Meta Quest, and HTC Vive. However, there are hardware limitations with the current head-mounted display (HMD). One such problem due to the hardware limitation is the readability of texts in VR. The main motivation of this thesis is to overcome or introduce another way to mitigate the problems while also exploring the medium and possibilities of VR.

1.2 Problem statement

Reading is an integral part of our daily lives. The normal visual acuity for humans is 20/20. However, due to the limitations of current HMDs with lower resolution than our normal eyes, small texts in VR can be illegible. For example, the three well-known HMDs are Meta Quest 2, Valve Index, and HTC Vive Pro 2. Respectively they have a resolution of 1832x1920 [1], 1440x1600 [2], 2448x2448 [3] per eye. Although the resolution seems great, remember that these displays are close to the user's eyes.

There are multiple ways to overcome this problem; one of them is by making the text bigger in the virtual world. For example, the developers can put a huge text board, bigger than what in reality would look like, so that the user can read it from a distance. This text board size is to justify the text size within. However, this solution can be a problem if multiple documents or a great quantity of text are placed in the virtual environment. If the virtual environment is the user's workplace and contains many documents that need to hold or placed, the virtual environment will be littered with huge text boards that obscure the user's view. Unlike in reality, where the A4 paper size is 8 x 10 inches, this paper will be much bigger to accommodate the small texts in VR.

1.2.1 Vergence-Accommodation Conflict

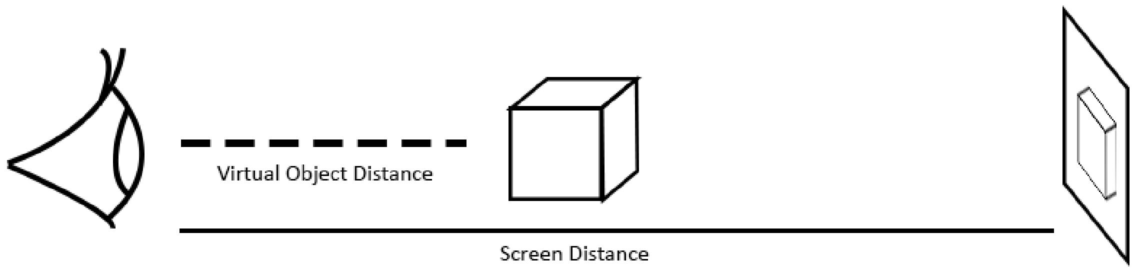


Figure 1: An eye focusing on the distance of the virtual object, shorter than the screen distance

A different way to overcome the problem is by bringing the text closer to the user's eyes. However, there is a unique problem with HMD: Vergence-Accommodation Conflict (VAC) [4]. This problem occurs when there is a mismatch between the virtual 3D object's distance and the focusing distance of the eye. Because HMD is just two 2D displays for each eye, there is no depth, and the display stays within the fixed position. If the virtual 3D object is closer to the eye than the display's distance, it will cause the eyes to focus on the wrong distance. This scenario will cause the virtual object to be blurry. This issue seems to be supported by another research paper where the research provided multiple potential solutions to address the VAC issue [8]. VAC also appears in the opposite scenario where the virtual object is further away than the display. However, it is less pronounced. Prolong use of this issue can cause vision problems such as headaches and eye strain. This issue inhibits the solution of bringing the texts closer to the user's eyes as such distance can render the text unreadable due to the VAC issue and cause the user to get eye strain.

1.3 Goals

This thesis will focus on designing a solution to overcome the problem of reading small texts using current HMD hardware with the provided medium interactivity from VR. The main goal is to serve as an accessibility option for people who have trouble reading in VR. As technology progresses and better HMD with better resolution will be produced, the solution will be helpful to those who have lower specifications HMD. While designing the solution, this thesis will explore ways to make the interaction of reading and managing texts more natural and comfortable.

2. Related Work

2.1 Reading without saccadic eye movements

This research paper presents text without saccadic eye movements in two ways; PAGE text, where an entire passage can be presented in static paragraph format, and rapid serial visual presentation (RSVP) [6]. Reading with RSVP was a concept first introduced by Foster [7]. Foster's research paper was an experiment to determine whether or not the syntactic complexity affects the visual perception of rapidly presented word sequences. However, in the "*Reading without saccadic eye movements*" research paper, RSVP was used to assess the limitation on reading speed imposed by saccadic eye movements. The RSVP in this research paper represents text sequentially one word at a time at the same location in the visual field.

The research paper results in the participants being able to read the RSVP at a rate faster than 1000 words/min. One of the participants was able to read the RSVP at up to 1800 words/min with high accuracy. The PAGE median reading rate was 278 words/min. It is also mentioned that the RSVP method could rely on visual stimulus. Contrast text, luminance, letter size, and range are important factors in making RSVP effective.

In the research paper discussion, it suggested that the RSVP could be made less taxing on the reader if they were given control to pause the RSVP. With the result from this experiment, it is safe to assume that the RSVP method could improve reading speed and will not cumber the user who chose to read in the RSVP method. The RSVP method can be offered in VR to help the user read faster.

2.2 Reading in VR: The Effect of Text Presentation Type and Location

This research paper investigated text position, orientation and presentation in a virtual environment [5] and has been the main influence on this thesis. Similar to the "*Reading without saccadic eye movements*" research paper, this research used paragraphs (PAGE) and Rapid Serial Visual Presentation (RSVP). This research paper used two types of display to represent the text. There are three conditions for the study where texts are placed. The first was world-fixed, where the text was displayed on the environment statically. The second was edge-fixed, where the text had a static position but dynamic orientation in the virtual environment. The third was head-fixed, where text was displayed on a heads-up display.

The experiment resulted in the RSVP method best displayed on head-fixed or edge-fixed if the user needs to move within the virtual environment. The paragraph method is best displayed on world-fixed or edge-fixed, but the participants could not read and move in the virtual environment simultaneously. Although these are the design recommendations at the end of the research paper, these recommendations are the result of user preference ratings. The objective result was that the RSVP readers made fewer errors than the paragraph readers.

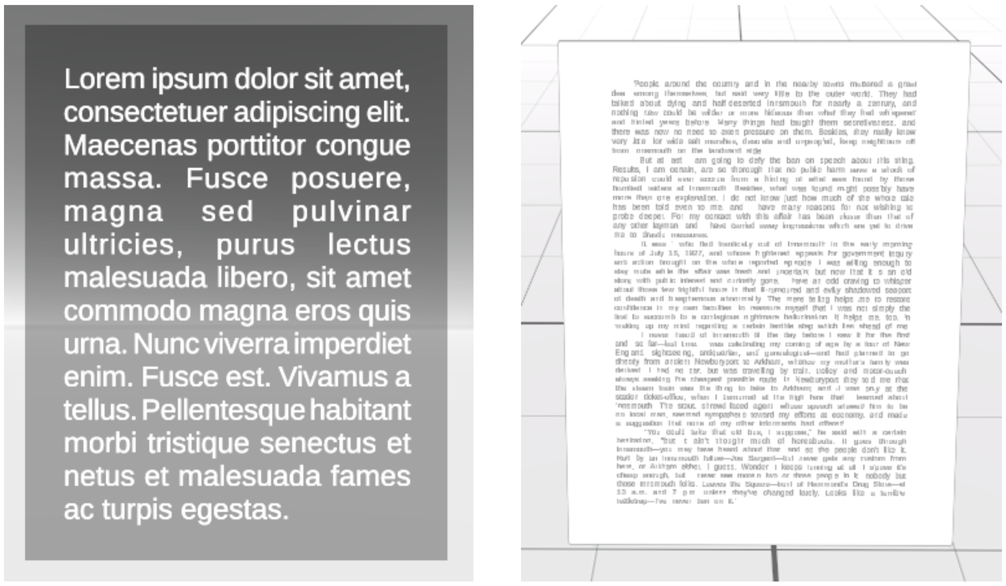
In the qualitative feedback section, participants voiced their feedback about the study. One major complaint from the participants during the world-fixed paragraph was the non-interactivity. One participant mentioned that they would like to be able to move the text, and another mentioned that the non-interactive museum might be redundant. When it comes to RSVP reading, the participants paid more attention to the RSVP, but they did not like it after a long duration. In the study, there was no control to change the reading speed of the RSVP nor to pause it.

3. Design

3.1 Interactivity

One of the major advantages of VR is the freedom of movement. The main goal of the interaction design is to give freedom of movement akin to real-life and alternative methods such as grabbing distance objects to overcome the space limitation. From the paper “Reading in VR: The Effect of Text Presentation Type and Location” [5], my design is inspired by the participant’s feedback and my ideas for improving the design choices.

3.1.1 Textboard design


Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor congue massa. Fusce posuere, magna sed pulvinar ultricies, purus lectus malesuada libero, sit amet commodo magna eros quis urna. Nunc viverra imperdiet enim. Fusce est. Vivamus a tellus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

(a) Textboard design (b) A4 paper mimic design

Figure 2: Visual appearance of textboards

To display texts in VR, the text has to be laid out in a flat colour to distinguish it from the background. The design choice of the textboard is white text over a dark semitransparent colour. The white colour of the text is to contrast the dark background (see Figure 2a). The semitransparent colour was chosen to give the user visibility over objects behind the textboard. These textboards are big in scale; they are meant to be read normally without any reading assistance. When it comes to interactivity, these textboards are interactive and can be grabbed.

An alternative visual design was made to mimic real-life papers. It is a page with opaque white colour with black text over it. Its size resembles closely to what an A4 paper should scale. This page will be harder to read because the texts are smaller and denser than the textboard (see Figure 2b). However, if the user has a better resolution HMD, they can pick up the paper and read it close to their eyes. Despite the difference in visual appearance, it still retains the same functionality as a textboard.

3.1.2 Object Interactions

From the qualitative feedback section of the "Reading in VR" research paper [5], many participants made a statement about the non-interactivity of the research software. This section will focus on the participants' feedback on the non-interactivity, mainly about being unable to grab an object. In this thesis, the design to grab an object will be about grabbing the textboard in the virtual environment. Since almost all VR controllers have a grip button, it will be used as a grab button to grab virtual objects. The user has to hold the grip button if they want to grab it and release the grip button to drop it. Grabbing an object can be performed with a left or right controller.

The design of grabbing an object mimics what it feels like in real life. If the user grabs an object, the object will follow the position and rotation of the controller. However, teleporting the object's position to the controller's main point will be jarring since it does not behave like real life. We grab objects by the surface and use that contact as a pivot to interact with or inspect the object. For example, grabbing a piece of paper can be done in multiple ways: using the thumb, index and middle finger as the grip, handgrip from the side of the paper, open palm behind the back of the paper, etc. This example is the main influence of the grab design. Assuming the user is grabbing an object by its side, the object will stay at the side of the controller. The grabbed object will follow the controller's changes of position and rotation. Another example is if the user is grabbing an object by the bottom, the object will stay atop the controller.

Another additional design is to compensate for the lack of locomotion in the software. Because different users have different physical spaces that limit their virtual space, there will be objects out of reach to some users. From these limitations, the idea of grabbing a distant object was designed. The user can grab the distant object by pointing at it with the controller. A pointer will be provided to assist the user, which helps the user aim at the object. Once the pointer hits an object that can be grabbed, it will be indicated by the sphere at the end of the pointer (see Figure

3b). The button to grab a distant object is the same as grabbing a contact object, but the pointer and its function to grab distant objects are only available for the right controller or hand. The pointer is angled to have the same angle, similar to the user pointing with their index finger (see Figure 3a). The behaviour of grabbing a distant object is different from grabbing the contact of an object. As discussed above, grabbing the contact object is similar to real-life. However, for distant objects to overcome the distance, the object has to be teleported to the user's hand. The moment the user grabs the distant object, the object will be teleported to the controller's main point. If the user does not like the object's position from their controller, they can release it and grab it again from the contact grab.

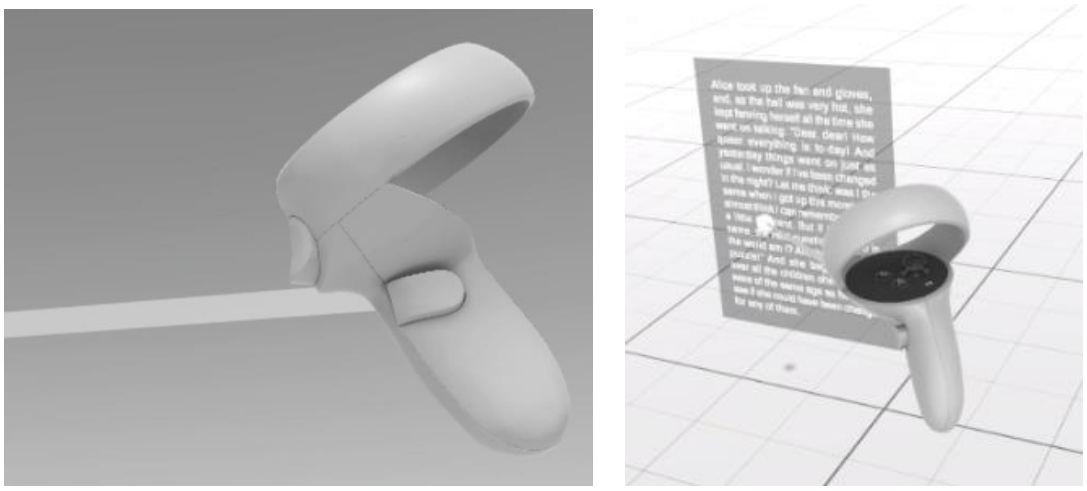

(a) Angle of the pointer (b) Pointer hitting an object

Figure 3: Visual appearance of the pointer

Because distant grab and contact grab share the same button, there will be a problem where the user accidentally grabbed two objects from their contact range and pointer. The priority of grabbing an object should focus on the contact object first and then the distant object. The controller that grabbed an object will be invisible to reduce visual clutter while grabbing an object. If the user grabs an object with their right hand, the pointer will be disabled, and the controller will be invisible until the player releases the object.

3.1.3 Cluttered Objects

Introducing the interaction of letting the user grab objects, especially distant objects may clutter the virtual environment. Textboards can pollute the virtual environment because the user did not return them to their original position. There could be a mismatch or out-of-order textboards due to this. If the user wants to place it back to the original position, it will be near impossible to return the object with hand to its original position and rotation. There should be a function to return the object to its original point.

The first design decision of returning an object function is that it should not use a button, specifically with VR controllers where input buttons are fewer in quantity than a keyboard or a gamepad. Buttons are often already designated for other functions. Therefore, the decision was made to use a gesture-based activation. One key idea of what gesture should be used is throwing or tossing. When the user grabs an object, they can throw it, and it will return to its original position. There will be an angular momentum check to discern between throwing and placing gestures. When the user releases the object while in a throwing motion, there will be an angular momentum check when the user releases the button; if the momentum is higher than the threshold, it will be teleported back to its original position. The design to only check the angular momentum value in the controller slowly changed the gesture to a wrist flick as it reduces the effort to return the textboard.

3.2 Rapid Serial Visual Presentation (RSVP)

The RSVP design is taken from the design recommendation section of the “Reading in VR” research paper [5] and improves and introduces new ideas to it. The main goal of the RSVP design in this thesis is not to cumber the user with too much information and properly align the user’s focus on what they want to see or read. The functionality will be similar to the RSVP that is being used by speed reading applications such as AccelaReader [15] or Spreeder. [16]

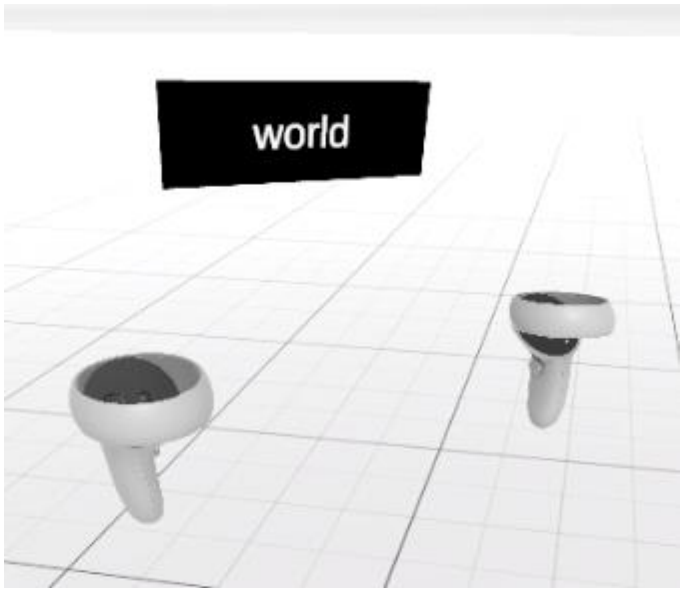


Figure 4: RSVP in third person view

The visual presentation of the RSVP will be a flat black colour rectangle with white text. The rectangle RSVP user interface (UI) will stay at the front of the user’s head direction, directing the user’s attention. The distance of the RSVP UI is around 15 cm from the user’s eyes (see Figure 4). The distance value shifted throughout the development, and it is finalized at around 15 cm to avoid eye strain from crossed-eye and give the user more peripheral vision. The location of the UI is placed just below eye level to give the user some vision of the surrounding. An early iteration of the RSVP rectangle had a semitransparent dark rectangle background. A problem arises if the user is moving their head while reading the RSVP, the non-static dark background of the rectangle can distract the user’s attention, and the user will lose their focus. Hence the flat dark rectangle was the final design of the RSVP UI.

The RSVP design and function will show one word at a given time, and quickly change the word to allow for rapid reading without requiring eye movement. The word will stay at the center alignment of the rectangle. When words stay at the center position of the rectangle, the user does not have to move their eyes but rather stays at one point of focus, which is the center of the word. The rectangle will not change size if the word is short or long. It will remain static as changes to the RSVP UI could distract the user. Because it remains static, the dark rectangle UI has to be wide to cover extensive words. The text font size will also remain static during reading. It is important to establish the RSVP visual design pattern and stay consistent throughout the run time. Any changes to the pattern once it is running can distract the user’s focus.

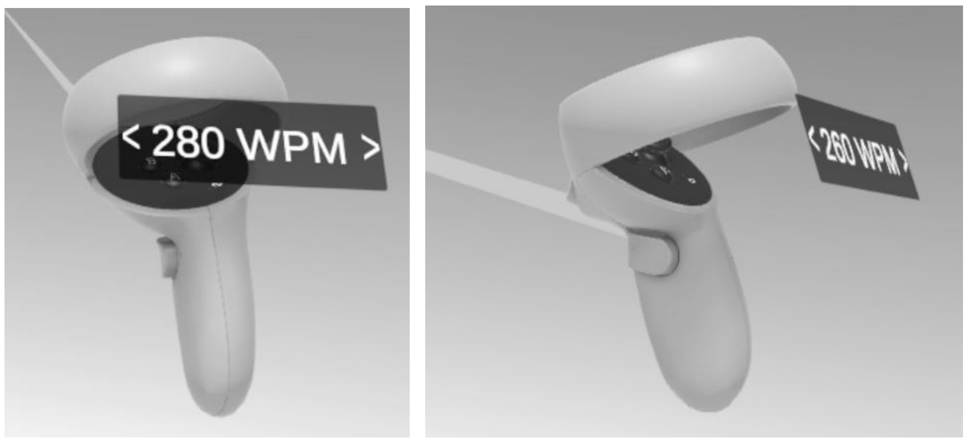
3.2.1 Enable and Disable the RSVP

The main goal of enabling and disabling the RSVP is to make it comfortable and natural for the user. The button that will be used is the trigger button because almost all VR controllers have one. The RSVP needs to have a text or paragraph loaded first to be enabled. Grabbing a textboard and then pressing the trigger button will enable the RSVP. The RSVP will read the text on the textboard and show the word one by one. The user is not required to keep holding the paper while the RSVP is on. The user can release the textboard from their grip, and the RSVP will stay enabled until the user turns it off. The RSVP will also take advantage of the pointer. The user can point with their right hand at the textboard and press the trigger button to enable and load the text to the RSVP without needing to grab the text board. To turn off the RSVP, press the trigger button again. If the RSVP is turned off, the text will be paused and not progressed to the next word. If the text is already loaded to the RSVP, the user does not need to grab or point at the paper anymore. Simply pressing the trigger in the air or anywhere will turn on the RSVP and continue the text. The trigger button must be context-sensitive to switch between enabling the RSVP, disabling the RSVP and loading the text to the RSVP. If the RSVP is still enabled and the user is pointing or grabbing a new textboard, the RSVP will remain active, but it will read the new text or paragraph. However, pointing or grabbing the same textboard and pressing the trigger button will disable the RSVP; another button press will enable the RSVP again and continue where it left off. There is a one-second pause gap to let the user adjusts to the new paragraph or location of the RSVP UI in front of their head. This pause gap will be triggered every time the user loads a new paragraph to read or when the RSVP is enabled again.

An additional function to disable the RSVP was added as a form of implicit interaction to ‘guess’ what the user wants based on observed behaviour. This feature is a derivative of head-tracking. If the user is distracted or notices something in their peripheral vision and turns their head to observe it, this quick motion will disable the RSVP. There is a threshold velocity to discern between normal head movements while reading and a quick head turn. The result is that when reading, some movement is allowed while the RSVP is visible, but when looking away quickly, the RSVP panel turns invisible, allowing the user to attend to the new stimulus without having to explicitly disable RSVP. However, if users want to re-enable RSVP, they have to use the trigger button as the quick head turn only disables the RSVP.

3.2.2 Speed of the RSVP

The RSVP speed is measured in words per minute (wpm). The default speed is 200 wpm. Different users have different reading speeds and preferences, and there is a speed controller embedded in the right controller. It will be using the joystick or touchpad of the controller. The user has to move the joystick/touchpad to the left to decrease the speed. Furthermore, the user has to move the joystick/touchpad to the right to increase the speed. Holding the joystick/touchpad in the right or left direction will change the speed at a faster rate. The speed change is in the incremental of 10.



(a) Front view (b) Side view

Figure 5: RSVP speed controller UI appearance

A floating UI is placed above the controller to indicate the information on the RSVP speed. Placing the UI above the user's controller will avoid obstruction from different controller models. It is a semitransparent dark rectangle with white-coloured text to show the information (see Figure 5). It will only appear if the user changes the speed and will be invisible if there are no changes within 0.5 seconds. The RSVP speed UI will be unaffected by the visibility of the controller. The UI will still appear if the user grabs an object and changes the RSVP speed. The user can change the RSVP speed even if the RSVP is turned on. Whether the RSVP is disabled or enabled, any changes to the speed will affect the RSVP.

4. Implementation

4.1 Background software and hardware

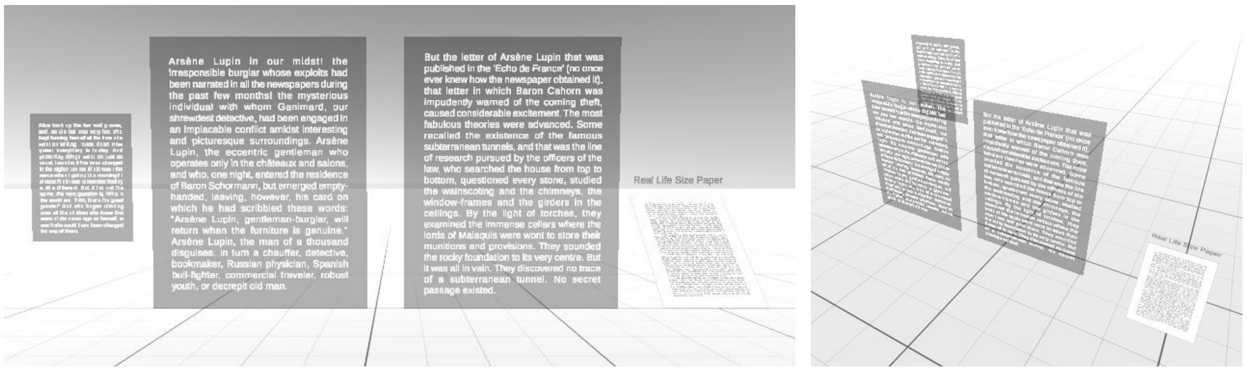
This thesis was implemented with Unity Engine [9] with the SteamVR plugin [10] from Valve Software. Unity engine is a game engine that offers 2D and 3D games, and it offers the ability to create VR games and software. Unity Engine uses C# as the programming language to write the script; The SteamVR plugin was chosen because it allows the game engine to recognize different variations of VR headsets and display the appropriate VR controllers that match the user's VR brand. The plugin allows the software to be compatible with multiple VR platforms and can be performed on other platforms for sharing and testing purposes.

The PC for this thesis runs Windows 10, Intel i7-10700F, 16GB RAM, and an NVIDIA GeForce RTX 3060 graphics card to render the environment. While developing the thesis' software, it was tested and developed with Meta Quest 2 headset with Oculus Link cable and software [11] to connect between the HMD and PC. Despite being tested and developed on Meta Quest 2, the software is designed and implemented with other brands of VR headsets in mind, such as Windows Mixed Reality (WMR), HTC Vive and Valve Index. It has been successfully tested on the HTC Vive and the Valve Index.

4.1.1 SteamVR Plugin

Since the software utilizes SteamVR API, the user must download the VR client from the Steam store [12]. The PC that developed and tested this software has SteamVR API installed to recognize different VR headsets. However, developing the software in Unity Engine requires installing the SteamVR plugin from the Unity Asset store [10]. There is a folder called prefabs from the SteamVR asset folder, and within it has the [CameraRig] prefab. The prefab can be dragged with a mouse to the scene, and it contains three objects: Controller (left), Controller (right), and Camera. These three objects are important to recognize the controller model and functionality and display the view to the HMD.

4.2 Virtual Environment


Arsène Lupin in our midst! the irresponsible burglar whose exploits had been narrated in all the newspapers during the past few months! the mysterious individual with whom Ganimard, our shrewdest detective, had been engaged in an implacable conflict amidst interesting and picturesque surroundings. Arsène Lupin, the eccentric gentleman who operates only in the châteaux and salons, and who, one night, entered the residence of Baron Schormann, but emerged empty-handed, leaving, however, his card on which he had scribbled these words: "Arsène Lupin, gentleman-burglar, will return when the furniture is genuine." Arsène Lupin, the man of a thousand disguises: in turn a chauffer, detective, bookmaker, Russian physician, Spanish bull-fighter, commercial traveler, robust youth, or decrepit old man.

But the letter of Arsène Lupin that was published in the 'Echo de France' (no once ever knew how the newspaper obtained it), that letter in which Baron Cahorn was impudently warned of the coming theft, caused considerable excitement. The most fabulous theories were advanced. Some recalled the existence of the famous subterranean tunnels, and that was the line of research pursued by the officers of the law, who searched the house from top to bottom, questioned every stone, studied the wainscoting and the chimneys, the window-frames and the girders in the ceilings. By the light of torches, they examined the immense cellars where the lords of Malaquis were wont to store their munitions and provisions. They sounded the rocky foundation to its very centre. But it was all in vain. They discovered no trace of a subterranean tunnel. No secret passage existed.

(a) Front facing view (b) Bird's eye view

Figure 6: Virtual Environment

The virtual environment is a simple open plane and consists of 4 boards with different text (see Figure 6). It was designed to showcase all of the features in the software. All of the boards are hovering close to the ground. The three boards are bigger in scale, and the fourth is relatively similar to the size of an A4 paper. The first two boards are the closest with different paragraphs of text. These two boards are placed near the user the first time the user is in the virtual environment. They are there to grab the user's attention. The third board is at a distance that the user cannot reach unless they walk out of the playfield or have a bigger room to walk towards it. The fourth is much smaller in scale, denser in text and resembles the size of an A4 paper. It was designed to emulate real paper. It contains a page from the novella "The Shadow over Innsmouth" by H.P. Lovecraft. Although the fourth textboard is visually different from the other three, all of these textboards can use the RSVP function.

4.2.1 Textboard Implementation

The textboard will be using Unity's Canvas object that allows UI components to be drawn within the Canvas. The Canvas' render mode is set to "World Space" to exist in the virtual environment, and the event camera is set to the SteamVR's camera. To draw the semi-transparent dark rectangle, it will be using the Panel UI. It has a colour picker that can change the colour and an alpha value to change the rectangle's opacity. There are two choices to draw the texts in the Unity engine: using Text or Text - TextMeshPro (TMP). The software will be utilizing TMP to give the user better clarity of text and sharp edges. The Panel UI will be the child of the Canvas, and the text will be the child of the Panel. A Rigidbody component is added to give some physics simulation to the textboard. However, gravity will not affect the textboard as the use of gravity is disabled. Next, another component is added which will help with the interactions, and that is Box Collider. This collider will be helpful in the programming section because it can detect if two colliders are colliding with each other. It has a separate function that can be triggered if it enters a collision or exits from a collision. It is important to tag the textboard as “Interactable” to distinguish between non-interactable objects (untagged objects) and interactable objects.

The script component of the textboard is simple. The script will be in the “CustomInteractable.cs”. It contains the attributes for the original position and original rotation. Create an object from the “CustomHand” class, which will be created later in section 4.3.2. This object is useful for storing the hand holding the textboard and avoiding multiple hands grabbing the same textboard. The original position and rotation attributes are coded as null, but when the software starts, it will fetch the value where the textboard was placed in the Unity Editor. It can be done with the *Awake* function, which is from Unity that will call the function once the software starts. The reasoning was to make the script reusable for multiple different textboards without hardcoding each position and rotation value. A function was added to fetch the text within the textboard, and it will return the string from the TMP. This function will be useful later to fetch the string from the textboard to the RSVP class.

4.3 Interactions

Most of the interactions section will be using the provided SteamVR API’s controller. Within the [CameraRig] folder, there are left controller and right controller prefabs. These will help show the appropriate model for the user’s controllers and recognize the controller inputs. For example, if the user is using a Meta Quest 2 headset, it will show the model of Meta Quest 2 controllers in the software. Recognizing controller input is important because different headset brands have different controller designs. For example, the HTC Vive controller lacks a joystick, but the software was developed and tested using Meta Quest 2. Instead, the SteamVR API will map the Meta Quest 2 joystick input to the HTC Vive trackpad.

4.3.1 Grabbing objects

Before going into the programming section, add the Rigidbody, Fixed Joint and Sphere Collider components within each controller. Rigidbody is useful for simulating physics that allows interactions with the textboard. Fixed Joint works with Rigidbody to join two Rigidbodies together—in this case, joining the controller Rigidbody and the textboard Rigidbody (Remember that the textboard has a Rigidbody component). Sphere Collider will be used for the programming portion of grabbing objects to detect if two colliders are entering or exiting the collision zone.

The script portion to grab an object will be in the "CustomHand.cs" script. There are many variables to be declared. SteamVR_Action_Boolean is a class that will return true or false if the specific action is pressed down or released. In this case, it will be used for recognizing the grip action. Declare it as a public object and in the Unity Editor, apply the "GrabGrip" action in the script inspector. However, for this function to return a boolean value, it requires an input button in the parameter. Create the object from SteamVR_Behaviour_Pose class for the main purpose of grabbing objects, and it has a variable input source that can be used within the SteamVR_Action_Boolean parameter. The component FixedJoint is used in the script for joining the textboard and the controller. Create the FixedJoint object in the script. For design purposes, create an object for SteamVR_RenderModel as that class has the attribute of the render model. Create objects for CustomInteractable and a list of CustomInteractable. The CustomInteractable object is for assigning the textboard that is being held. The list is important for later when there are multiple textboards within the hitbox of a controller.

In the Awake function, get the components for the variables that require it. SteamVR_Behaviour_Pose can be fetched from the component SteamVR_Behaviour_Pose. FixedJoint by getting the FixedJoint component. Moreover, get the component for SteamVR_RenderModel. Because there are collider components within the controller object and the textboard object, create two functions whenever they enter the collision (OnTriggerEnter) and when they exit the collision (OnTriggerExit). OnTriggerEnter function adds the textboard object to the list of CustomInteractables whenever the textboard collider hits the controller collider. OnTriggerExit removes the textboard object from the list if the textboard collider exits the controller collider. In Unity, there is an *Update* function that will be called for every frame. The *Update* function has multiple if statements for checking if the button is pressed or released.

For the purpose of grabbing an object, there are simply two if-statements in the *Update* function: grab press and grab release. Grab press has a call function for Pickup, and grab release has a call function for Drop. The Pickup function will find the nearest textboard to the controller within the list of CustomInteractables by using the Pythagorean theorem. An if-statement was added if, for example, a textboard is being held, the other hand can grab it, and the textboard will move to the other hand. To attach the textboard to the controller, assign the connectedBody attribute within the FixedJoint as Rigidbody from the CustomInteractable. Once the object is grabbed, set the render model to false to be invisible. If the CustomInteractable list is empty, the Pickup function will not do anything. For the Drop function, it will release the textboard by assigning the connectBody attribute to null and the CustomInteractable variable to null. It will also set the render model to true to be visible. Lastly, there will be an if-statement in the Drop function to check if the controller angular velocity is higher than the threshold value. Retrieving the angular velocity can be done by calling the SteamVR_Behaviour_Pose behaviour named GetAngularVelocity. If the velocity is higher than the threshold value, the textboard will return to its original position by setting the position and rotation to the stored original position and rotation attributes within the CustomInteractable class. The threshold value is hard-coded, and the value was chosen from multiple trial-and-errors.

4.3.2 Pointer for distant objects

The implementation of the pointer requires a camera object as a child object of the Controller (right); as stated in the design section, only the right controller has the pointer. Setting the field of view to 1 will reduce the rendering workload significantly as it only renders a narrow field of view. Afterward, add a new component in the camera object for Line Renderer. This component will draw the line or laser-like pointer. A sphere object is later added as the child of the camera to be an indicator whenever the pointer hits interactable objects. In this case, it will be the textboard. Within the sphere object, there is a Sphere Collider component.

The script for the pointer is called “Pointer.cs”, and it requires modification with the CustomHand class later for the grabbable portion. In the pointer script, there are three variables: The sphere object, CustomInteractable object and LineRenderer object. The sphere object moves the sphere to the location where the camera hits an object. However, the sphere, by default, is invisible. CustomInteractable will be the attributes to store which textboard was hit by the camera.

LineRenderer is to draw a line from the camera to the given line limit length. In the Awake function, get the component of LineRenderer and assign it to the attribute. In the Update function, create a RaycastHit attribute to find the distance between the controller and the hit location. This distance is to draw a line from the controller point to the end point by simply using the function within the LineRenderer object.

Move the sphere to the end of the line to appear as an indicator. RaycastHit also has the attribute to retrieve the object that was hit. An if-statement is made to distinguish between objects by the tag. Remember that the textboards are tagged as “Interactable” in the textboard implementation section. If the RaycastHit has an “Interactable” tag, the interactable object will be assigned to the CustomInteractable attribute. This attribute is public and can later be accessed by CustomHand class. If the RaycastHit hits an object tagged as “Interactable”, it will also set the sphere to be visible. If the Raycast does not hit anything or any “Interactable”, the CustomInteractable attribute will be null, and the sphere will be invisible.

Modifying the CustomHand script is required so that the distant object can be taken into the script and grabbed. Pointer object is required and set to public so that the script from the Pointer object can be assigned to the CustomHand script in the Unity Editor. The Pickup function has to be modified to check both the list of CustomInteractables and the Pointer attribute of CustomInteractable. It will prioritize the interactable in the list of CustomInteractables first. However, if the list is empty, the Pointer’s CustomInteractable will be the one that will be attached to the controller, and it will teleport the textboard to the location of the controller. Create a GameObject object for the camera and get the camera’s component in the Awake function. The GameObject will be the object to set the pointer’s LineRenderer invisible once the right controller grabs an interactable.

4.3.3 The first distant grab logic

The first distant grab logic was different from the current one. The current one considers the list of CustomInteractables in the collision zone and the Pointer's CustomInteractable as separate variables. However, in the first iteration, it added the Pointer's CustomInteractable to the CustomInteractable list. The Pickup function still prioritized the nearest textboard but only under the threshold distance. Above the threshold distance, it would teleport the textboard toward the controller. This old logic caused many issues, such as the pointer's textboard attached to the controller but did not teleport to the controller because it was still under the threshold. The textboard in the controller's collision zone would teleport to the hand because it was too far from its centre despite the controller colliding with the textboard. Furthermore, there was a loop where the textboard did not get removed properly from the list of CustomInteractables, causing the grabbed object to be forever stuck to the hand whenever the grip button is pressed. It was later redesigned to the current one, which is more predictable than the first iteration by considering the CustomInteractable list and the Pointer's CustomerInteractable as separate variables.

4.4 RSVP implementation

To create the UI for the RSVP, create a Canvas object with a Panel object as its child. The Panel will have a TMP object like the textboard implementation. In the Canvas inspector, set the Render Mode to "Screen Space - Camera" and Render Camera to the Camera from [CameraRig] so that it appears in front of the VR screen like a heads-up display. Set the Panel colour to black with 100% alpha value.

The script to make the RSVP functional will be called "RSVP.cs", and it contains multiple attributes to maintain the RSVP status, for example, a boolean variable if the RSVP is running, a string variable to store the text content from the textboard, and the speed of the RSVP. The object variables are Canvas and TMP. In the Awake function, get the component for Canvas and TMP and assign it to the respective attributes and set the Canvas to be invisible. Loading the text from the textboard to be displayed on the RSVP UI requires the string content from the textboard. Since the CustomInteractable class has a function to return the string within the textboard text, the string can be sent to the RSVP class. However, there is a string check to see if the content is the same as the previously loaded content. The check will be a simple string equal check that if it has the same content, it will not replace the current content. If the content is different, it will parse the string and split each word by spaces into a string list of words. The RSVP will read each word in the string list, and it will stop if it is at the end of the list.

To make the RSVP show each word in the TMP object, using the IEnumerator function with Coroutine from Unity will simplify the programming logic without the deltaTime function. IEnumerator has the function to wait in real-time without using deltaTime. deltaTime calculates the time of each frame. Since there is the speed of the RSVP variable, put the variable as the wait time before it iterates to the next word. The default speed is 0.3 seconds per word. In words-per-minute (WPM), it will be 200 WPM. The calculation to find the frequency from WPM is $f = \frac{1}{wpm\ /\ 60}$. WPM will be the default unit to show the RSVP speed. There is a 1.0-second pause gap every time the RSVP is enabled or loaded a new text by adding another wait time function.

4.4.1 Turning on or off the RSVP

The RSVP script has multiple functions called in the CustomHand script instead of the RSVP script. These functions are: turning off the RSVP, turning on the RSVP and toggling on/off the RSVP. Each of these functions is to cover multiple contexts result. A string check function is placed in the enable RSVP function to differentiate between loading a new text or continuing the previous one. The actions to enable or disable RSVP will be implemented in the CustomHand script and not the RSVP to accommodate multiple button presses of grip and trigger.

In the "CustomHand.cs" script, add a new SteamVR_Action_Boolean public object to recognize trigger press action. In the Unity inspector, the action will be "GrabPinch". The variable will utilize the same SteamVR_Behaviour_Pose object previously implemented in the grabbing objects section. Modify the Update function to consider the new trigger press. The first if-statement covers the pointer if it hits an interactable. When the pointer hits an interactable, a trigger press will allow the RSVP to be enabled. The second if-statement covers the current held interactable. While holding the textboard, pressing the trigger will enable the RSVP. The third if-statement is to toggle on or off by pressing the trigger button without holding or aiming at the textboard.

4.4.2 Quick-Head-Turn

The Camera object in the [CameraRig] will have a new script to implement the quick-head-turn function to disable the RSVP. This script is called “Headset.cs”. It has the RSVP object and the list of float for the history of velocity. When the software starts, it will record the current angle value of the headset as Vector3 type and keep it as both the current frame angle and previous frame angle. These two variables store the same angle to avoid the sudden change from 0 to the current angle that can cause high-value velocity at the start. This pre-caution was implemented in the *Awake* function.

The *Update* function will be the primary function to implement the quick-head-turn algorithm. Every frame will record the current angle value of the headset and find the velocity between the current and the previous angle frame. Once the velocity is obtained, append it to the list of velocities. The list’s maximum size is only 3, and once it is filled, it will replace the oldest velocity value with the newest value. The size of the list is 3, so it only records three frames of velocity. Once the list is filled, it will find the average velocity of the three stored velocities and see if it is above the threshold value. The averaging of velocities is to stabilize and reduce the possibility of accidental activation. Once the average velocity is above the threshold value, it will disable the RSVP.

Although the velocity is accurate, the angle value is retrieved from Unity’s Vector3 transform forward attribute. Upon printing the current angle value, it seems that the value did not retain consistency. For example, during debugging, rotating the head on the X-axis could affect the X angle value and Y angle value. Other times, rotating the headset on a specific axis did not affect the axis angle value but affected another value greatly. Because of this, it is only possible to use the velocity value as the angle records the delta change properly but not the specific axis angle value. There was an additional design to turn on the RSVP back. The concept was that if the headset returned near the angle that disabled the RSVP, it would enable the RSVP back. However, because the angle is not stored properly, this concept was impossible with the given way to obtain the headset’s angle value.

4.4.3 Changing the speed of RSVP

Implementing the speed controller for the RSVP requires the UI attached to the right controller. For this purpose, add a Canvas child under the Controller (right). Within the canvas is the Panel child to create a dark semi-transparent rectangle. The canvas Render Mode will be in World Space as it will hover just above the controller model and exist in the virtual environment. Within the Panel will be the TMP child to display the speed information. Move the Canvas on the Y and Z axis from the original zero position to make it hover above the controller. Within the Controller (right) object, add a new script that will change the speed of the RSVP and recognize the joystick from the controller.

The script will be called “RSVPController.cs”, and there are many variables inside. There are two SteamVR_Action_Boolean for recognizing the action of moving the joystick left and right. In the Unity inspector, it will be set as “SnapTurnLeft” for joystick left and “SnapTurnRight” for joystick right. Afterward, create another SteamVR_Behaviour_Pose object to recognize the input for the SteamVR_Action_Boolean. An RSVP object is required to update the RSVP speed attribute. In the *Awake* function, obtain the SteamVR_Behaviour_Pose and Canvas components. By default, the Canvas component is invisible. Another object in the script was created for the TMP object within the Panel. This TMP contains the text shown in the controller UI, and its value can be changed.

There are multiple if-statements for the joystick being held left and right or released from the hold in the *Update* function. If the joystick is held in the right direction, it will increase the speed by 10 WPM, and the UI will show. To show the UI, it uses another IEnumerator function and Coroutines, similar to the method to show the RSVP UI. The speed will be increased by 10 WPM, but if the joystick is still being held after 0.2 seconds, it will rapidly increase the speed by 10 WPM per frame. This works the same way in the opposite direction to decrease the speed. The UI will turn invisible if the speed does not change after 0.5 seconds. To update the RSVP speed, every time there is a change on the WPM variable, the formula $f = \frac{1}{wpm\ /\ 60}$ will change WPM to word frequency, which the RSVP uses.

5. Conclusion

5.1 Informal Feedback

When the software was completed, the executable was sent to be tested on HTC Vive and Valve Index. Both headsets were able to run the software, and the functions such as grab, pointer, speed controller, RSVP, and quick-head-turn were working as intended. The software was able to remap the input from the joystick of Meta Quest 2 to HTC Vive’s trackpad. Since the Valve Index controller has a trackpad and a joystick, the input is still located on the joystick. The only change on Valve Index is the Meta Quest 2 grip button, as it is remapped to the Valve Index squeezer.

However, there was an issue with the software running on the Valve Index. The RSVP did not get enabled at some time, and the cause of the bug is unknown. An effort to reproduce the bug on the Meta Quest 2 headset did not cause the bug to reappear. The cause of this bug could be the grip action in Valve Index. The controller for Valve Index does not have one button to grab but rather a squeezer for the middle, ring and pinky finger. It could be because it calls two functions (on and off) as it satisfies two if-statements. It could also be because the parameter to fulfil the if-statement was not met.

There was feedback from the participant who tested the software. They mentioned that the RSVP UI location should be adjustable by simply grabbing it. The participant preferred the RSVP placement to be similar to a subtitle at the bottom field of view. However, the distance of RSVP from the eye was comfortable, and the RSVP speed controller was working as intended. The pointer helped the participant overcome the physical space limitation and interact with the distant objects.

5.2 Future Work

The software can be improved with more features, such as a file import functionality to import PDF or Word document files. If a document were imported, it would make a copy of its content in the textboard, and each page in the document would create a new textboard. This functionality will require some work to adjust the textboard layout to be able to lay all the content like headings and images. It would also require a menu system to be made so that the user can open a file explorer and pick the document within the virtual environment.

Another feature that can improve upon the software is eye-tracking. Multiple VR headsets, such as HP Reverb G2 Omnicept [13] and VIVE Pro Eye [14], have eye-tracking hardware in the headset. This feature could help the user turn on or off the RSVP without pressing a button. For example, the user can turn on the RSVP by looking at the textboard for a specific duration. Furthermore, the user can disable the RSVP if the user does not focus on the UI anymore.

Improving the software could also be done, for example, by showing the user where they paused the RSVP by highlighting the word in the textboard. Another feature would be to pick the word where the RSVP will start. Locomotion functionality could also be added if the virtual environment becomes bigger than the current one. For example, turning the virtual environment into a museum.

References

[1] Oculus Developers. Accessed March 24th, 2022, from Oculus for Developers Website: https://developer.oculus.com/resources/oculus-device-specs/

[2] Valve Index. Accessed March 24th, 2022, from Valve Software Website: https://www.valvesoftware.com/en/index/headset

[3] VIVE Pro 2. Accessed March 24th, 2022, from HTC Vive Website: https://www.vive.com/eu/product/vive-pro2/specs/

[4] O. Kreylos. “Accommodation and Vergence in Head-mounted Displays” Accessed March 22th, 2022 from Oliver Kreylos’ website: http://doc-ok.org/?p=1602

[5] R. Rzayev, P. Ugnivenko, S. Graf, V. Schwind, N. Henze. 2021. “Reading in VR: The Effect of Text Presentation Type and Location”. In Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems (CHI '21). Association for Computing Machinery, New York, NY, USA, Article 531, 1–10. DOI: https://doi.org/10.1145/3411764.3445606

[6] Gary S. Rubin, Kathleen Turano, “Reading without saccadic eye movements”. In Vision Research, Volume 32, Issue 5, 1992, Pages 895-902, ISSN 0042-6989. DOI: https://doi.org/10.1016/0042-6989(92)90032-E.

[7] Forster, K.I. “Visual perception of rapidly presented word sequences of varying complexity”. Perception & Psychophysics 8, 215–221 (1970). DOI: https://doi.org/10.3758/BF03210208

[8] G. Kramida, "Resolving the Vergence-Accommodation Conflict in Head-Mounted Displays," in IEEE Transactions on Visualization and Computer Graphics, vol. 22, no. 7, pp. 1912-1931, 1 July 2016, DOI: https://doi.org/10.1109/TVCG.2015.2473855

[9] Unity Engine. Accessed November 16th, 2021, from Unity Website: https://unity.com/

[10] SteamVR Plugin for Unity Engine. Accessed November 16th, 2021, from Unity Asset Store: https://assetstore.unity.com/packages/tools/integration/steamvr-plugin-32647

[11] Oculus Link Software and Hardware. From Meta Quest Website: https://www.oculus.com/accessories/oculus-link/

[12] SteamVR Client. From Steam Store: https://store.steampowered.com/app/250820/SteamVR/

[13] HP Reverb G2 Omnicept. Accessed April 24th, 2022, from HP Website: https://www.hp.com/us-en/vr/reverb-g2-vr-headset-omnicept-edition.html

[14] VIVE Pro Eye. Accessed April 24th, 2022, from HTC Vive Website: https://www.vive.com/ca/product/vive-pro-eye/overview/

[15] AccelaReader. Accessed on April 24th, 2022, from AccelaReader Website: https://accelareader.com/

[16] Spreeder Free App. Accessed on April 24th, 2022, from Spreeder Website: https://www.spreeder.com/app.php?intro=1