LONDON'S GLOBAL UNIVERSITY

Magnification vision in virtual reality

Creating and evaluating a novel gaze-directed user interface

Sondre Milch Agledahl

MEng Computer Science

Submission Date: 5th May 2020

Supervisor: Prof. Anthony Steed

This report is submitted as part requirement for the MEng Degree in Computer Science at UCL. It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged.

Abstract

The recent introduction of eye-tracking technology in consumer-level virtual reality devices is opening up a brand new avenue for user interaction. Eye movements are fast and precise, and their use as an input modality in a virtual 3D world has not yet been readily explored.

This project presents a novel magnified user interface for virtual environments controlled by real-time eye gaze data, allowing users to explore their virtual surroundings in a new way. The system builds on the optics of real magnifying glasses and past work in gaze-directed interfaces, but is developed through trial and error, culminating in a Unity application for the VIVE Pro Eye.

An experiment is designed and run to evaluate the system, consisting of a search-based puzzle task where subjects' performance using the gaze-directed interface is measured and their feedback gathered. An analysis of the results of the study is made, finding no significant improvements in performance from the new system. The cause of this result is discussed, and an evaluation of the interface and the project as a whole is made, indicating suggested areas for targeted future work.

Contents

Chapter 1

Introduction

The increasing availability of eye tracking technology for virtual reality head-mounted displays (HMDs) opens up new avenues for user interaction in 3D virtual environments. Eye movements are fast, precise, and – at present – a novel input modality whose potential and most effective use has not yet been established by developers in consumer applications.

Presented here is a novel system for magnification vision in virtual reality – an eye-gaze-directed interaction technique allowing users to magnify their surroundings and zoom in on distant objects.

The aim of this project was to design and implement one, or several variations on, such a system, created to be fast, user-friendly and enjoyable to use. To evaluate the system's applicability, we aimed to design and run a user study that could assess 1) whether the system improved subjects' performance in a VR puzzle task, and 2) whether the system was easy and pleasant to use.

The gaze-directed magnification system was developed iteratively in the Unity game engine, based on an initial rough concept design. The final implemented solution draws on the optics of real magnifying glasses, but is the result of trial and error and heuristic tweaking to arrive at an intuitive and effective user experience. The user study was designed to give subjects the opportunity to use the magnification system freely, and to reflect on its utility. The study was conducted with volunteers from UCL.

In the chapters ahead are described the design and implementation of the gaze-directed magnification system, as well as the design of and the results from the user study. Chapter 2 describes the research context in which the project sits, and provides specific technical background necessary to describe the new system accurately. Chapter 3 provides a detailed account of the design of the system and the process of developing it into a complete application, explaining the novel design decisions and features added along the way. Chapter 4 describes the design of the user study, and Chapter 5 provides a summary of the results it produced. Finally, Chapter 6 gives an evaluation of the new system, the experiment, and the project as a whole, discussing suitable changes that might have been made were the project to be re-done.

Chapter 2

Project background

2.1 Academic context and motivation

2.1.1 Eye tracking in virtual reality

The application of users' gaze data in virtual reality (VR) has been researched for decades [1], although the focus has often been on rendering improvements, such as foveated rendering [2] and gaze-contingent level-of-detail rendering [3]. The use of eye gaze as an input modality for computing devices has been extensively studied as an aid for people with motor disabilities (exemplified by [4]). However, the application of gaze interaction to 3D virtual environments has been less explored in the literature, although its potential as a new avenue for interaction design was recently highlighted by Hirzle, *et al.*, who laid out a design space for categorising VR gaze interaction techniques in a human-computer interaction context [5].

Eye gaze is an attractive potential source of user input in VR for several reasons. Firstly, eye movements provide precise three-dimensional input (in terms of the ray from the user's gaze to a point in 3D space) at a much larger speed than traditional button or joystick input (with saccades reaching speeds of up to 900°/sec [6]). Further, eye gaze naturally accompanies most interaction techniques in a predictable way: A user will typically look at an item to select from a user interface before performing the action necessary to indicate selection.

One well-known and unique concern in developing gaze-based interaction techniques for computing devices, however, is the so-called "midas touch" problem, identified as early as 1990 by Jacob [7]. From experience using gaze input as a UI selection mechanism, he notes:

> *This is an unworkable (and annoying) approach, because people are not accustomed to operating devices just by moving their eyes. They expect to be able to look at an item without having the look "mean" something. Normal visual perception requires that the eyes move about, scanning the scene before them. It is not desirable for each such move to initiate a computer command.* [7]

As mitigation, Jacob suggests the use of "gaze dwell time", where selections are only registered when the user fixes his gaze at a specific region for some threshold period. This method is frequently used and has been found to be reliable (at

dwell times around 500 ms [8]), although increasing dwell time clearly limits the speed with which selections can be made.

A more novel attempt at mitigating the midas touch problem for VR by Piumsomboon, *et al.* is worth mentioning, as it directly influenced the solution arrived at in this project: In the "Duo-Reticles" technique described here, one reticle follows the target of the user's gaze inside a HMD, while a second "inertial reticle", following a moving average of the first reticle's positions, lags close behind. Only when the two reticles align (indicating a steady gaze for some small period) is the user's gaze registered as a selection. While no performance benefit was found from this technique, experiments indicated that it provided an improved user experience over gaze-dwell [9].

2.1.2 Virtual reality locomotion

In order to allow HMD users to explore virtual environments that are larger than the constraints of their physical space (which for consumers is very limited and varying in size), finding an appropriate locomotion technique has long been an open problem in VR development to which no universally accepted solution has been established [10]. Whereas traditional 3D simulations have typically used joystick input to control movement, the corresponding technique translated to simulations in HMDs is known to cause motion sickness [10] [11] [12].

The most commonly applied solution to this is to implement a *teleportation* mechanic [10], a popular manifestation of which was initially highlighted by Bozgeyikli, *et al.* [11]: Here, the user aims an arc with his hand that points towards a spot on the ground; by clicking a button he will immediately be transported to the virtual location pointed to without having physically moved. While this technique has been noted as being flexible and easy to use, it does not correspond to any "real life" phenomenon, which is detrimental to the user's sense of presenece, and although fast, it requires time after each application for the user to reorient himself to his new surroundings [12].

A more sophisticated method which more closely emulates real-life movement has been demonstrated in "redirected walking" techniques. These work on top of the physical movements made by a user in a tracked HMD by shifting or rotating her virtual movements by additional imperceptible amounts, to give her the impression of having traversed an environment that is larger than her physical play space [12]. This method was explored directly in the context of eye tracking by Sun, *et al.* [6] by performing redirection during the temporary blindness induced by saccades, showing impressive gains in virtual space for larger physical environments – although far less significant results were seen in regular room-sized tracking areas.

In general, a literature review of VR locomotion by Boletsis suggests there is a preference among users for "continuous locomotion" (without the "jumps" induced by teleporting), as it consistutes a more consistent virtual experience [10], but it still presents limitations in being able to show a consistent, traversable environment in small tracking spaces of varying, unpredictable dimensions.

Gaze-directed magnification does not by itself constitute a locomotion technique, but aims to allow users to explore a large virtual environment in a small physical tracking space without discontinuous "jumps" in virtual location. Limiting the extent to which the magnified area of the environment is visible will prevent the mismatching visual and vestibular cues known to cause motion sickness.

2.2 Technical background

2.2.1 Magnification

Content in this section is adapted from an online physics textbook by Ling, *et al.* [13], and aims to summarise the aspects of real-life magnification simulated by this project.

The *magnification* of an image of an object, as seen through a convex lens, is a measure of how many times larger the image is than the object when seen by the naked eye. Formally, it is defined as the ratio of the angle subtended by the magnified image from the centre of the eye to the angle subtended by the unmagnified object.

An average adult is estimated to be able to see objects held 25 cm away from her eye at closest (known as the "near point" of the eye). The maximum magnification achievable through a given lens is determined by its *focal length* (the distance from the centre of the lens to its focal point), a physical property that depends on the size and curvature of the lens.

With this in mind, when looking through a real magnifying glass, the magnification of an object held at the near point is given by

$$M = \left(\frac{0.25}{L}\right)\left(1 + \frac{L - l}{f}\right)$$

where f is the focal length of the lens, and the relationship between the distance parameters L and l is indicated by **Figure 1**.

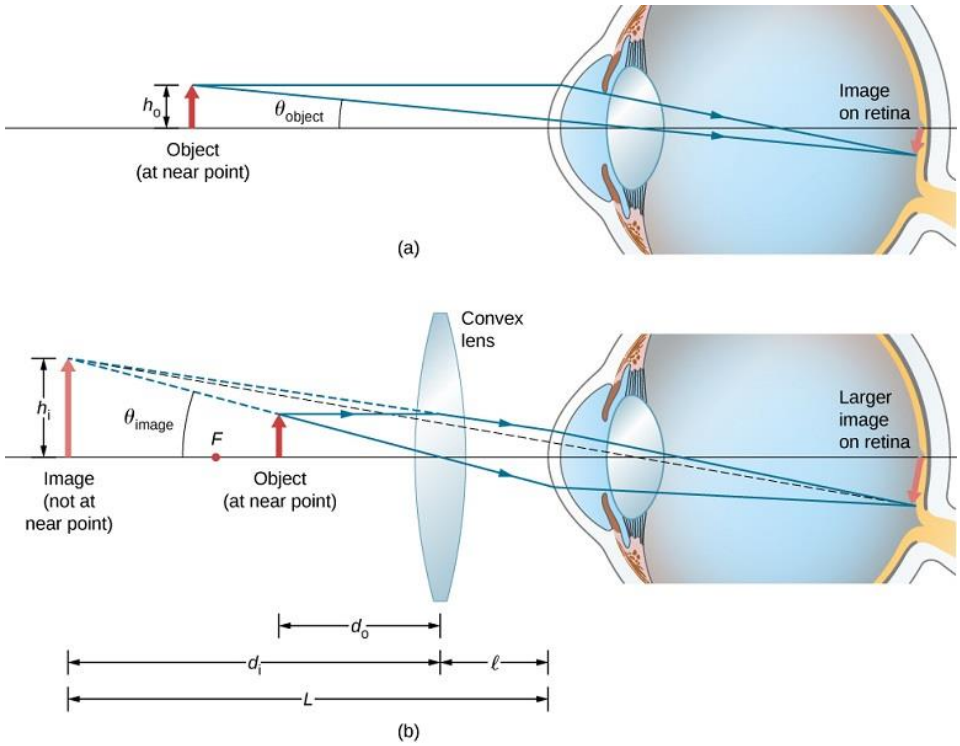

Figure 1: Diagram showing the relationship between the variables involved in producing a magnified image through a convex lens. (a): An object is placed 25cm from the eye; (b) a convex lens is placed between the eye and the object, producing a magnified image on the retina. F indicates the focal point of the lens.[1]

2.2.2 Unity VR development

The Unity game engine and development suite is a popular tool for developing video games and applications, with build support for most contemporary HMDs. The engine provides (among other things) graphics rendering, physics simulation, and a GUI interface; it uses C# as a scripting language [14].

Unity *scenes* consist of *objects* arranged hierarchically. Each object has a *global position* (relative to the scene's origin) and a *local position* (relative to its parent object) in 3D space. An object's properties and behaviour are defined in terms of its *components*, which may include a 3D mesh, a physics component, an audio player or a custom user-defined component. Notably, for an object to be used in the engine's collision system, it needs to have a *Collider* component, which consists of a convex 3D shape that can be adjusted to fit the bounds of the object's mesh. The collision system can simulate the collision of rigid bodies, but

[1] Disclaimer: This image is figure 2.82 of chapter 2.8: "The simple magnifier" of the online textbook referenced by [13]. The contents of the article are licenced under a Creative Commons Attribution Licence 4.0, available at https://creativecommons.org/licenses/by/4.0/legalcode. No changes to the original image were made.

can also be queried manually with API calls such as *Raycast*, which will provide collision information about a virtual ray and a Collider in the scene.

Another provided component is the *Camera*, which works abstractly as a device that captures the view of the scene that will be seen by the player. Its adjustable properties include aspect ratio, choice of projection matrix (perspective or orthographic), and field of view (FOV). In addition to rendering to a device screen, a Camera can also be set to render its view of the scene in real time to a *render texture*, which in turn can be applied to a 3D mesh inside the scene itself.

The *SteamVR* plugin for Unity provides a collection of pre-made objects and boilerplate code for developing VR applications. This includes components and API interfaces for tracking the position and orientation of a connected HMD, as well as commonly used VR controllers. It also includes implementations of abstractions such as the *playspace*, a virtual representation of the user's physical tracking space, with mechanisms in place to alert the user when she is about to step out of the bounds designated for play.

Chapter 3

System design and implementation

3.1 System requirements

3.1.1 Proposed system design

We proposed an initial design for a gaze-based magnification technique in VR, to be iterated upon through trial and error. A formal system requirements list for the technique was not compiled, but the proposed design naturally led to several concrete implementation goals arising; these are detailed in this section.

In this system, the user holds both hands in front of her face and focuses her gaze on the space between them to activate magnification. Once activated, a rectangular screen appears to fill the gap between the user's hands, displaying a magnified view of the environment on the other side – for simplicity, we refer to this rectangle as the *MagRect*. As the user shifts her gaze around on the surface area of the MagRect, the view should zoom in in real time on the object being gazed at. In other words, looking through the MagRect at an object far away from the user should increase magnification, whereas looking at an object close by should decrease it. Whether additional parameters or user input modalities should be used to control magnification was not specified at this stage.

This design gave rise to the following minimum implementation requirements:

- Creating a variable-sized quadrilateral object (the MagRect) that would display a view of the environment on the other side, and that would move, rotate and scale to fill the gap between the user's hands in an intuitive way.
- Mapping eye tracking data about the user's gaze to a spot on the surface of the MagRect.
- Mapping the gaze spot on the MagRect to the world space coordinate corresponding to the object gazed at on the other side of the MagRect.
- Adjusting the view displayed by the MagRect to "zoom in" on the world space coordinate gazed at by the user.

Although these primary components of the system were often developed and revised together, they are presented here separately (as much as possible) for clarity's sake. Finer-grained design decisions were decided upon during

development, either due to constraints in the technology or to maximise usability after experimenting with different solutions.

3.1.2 Technical setup

The system was developed for use with the VIVE Pro Eye head-mounted display [15] using Unity version 2019.2.17f1. The user's hands were tracked using the VIVE Pro controllers in a tracking space set up with VIVE "light house" tracking devices.

Real-time eye tracking data was retrieved through the Tobii XR SDK. This library interfaces with Unity to provide, for every frame, data about the origin and direction in world space of the user's gaze [16].

3.2 Constructing the MagRect

The Unity ecosystem naturally gave rise to a solution for displaying a rectangular shape that gives a view of the virtual world on the other side. A *quad* object (quadrilateral in 3D space) is placed between the user's hands, and a *Camera* object is placed immediately behind it. The image captured by the camera is rendered to a render texture, which is in turn applied as a texture to the MagRect quad.

The MagRect should be placed in the middle of the gap between the left and right hand, and its width should shrink and expand dynamically so it always fills this gap. In other words, given the world space positions of the right and left hands, $\boldsymbol{x}_R$ and $\boldsymbol{x}_L$, the position and width of the MagRect should be

$$\boldsymbol{x}_{MagRect} = \frac{\boldsymbol{x}_R + \boldsymbol{x}_L}{2}$$

$$w = \|\boldsymbol{x}_R - \boldsymbol{x}_L\|$$

From here, to let the user rotate the MagRect as if it were a physical object, it was decided to orientate it to give the appearance of a glass being held between the palms of the hands. Thus, given that the hands are held out in front of the user at an equal height, with fingers pointing upward, let the local y-axis of each hand (the direction of their fingers) be $\boldsymbol{u}_R$ and $\boldsymbol{u}_L$. We then keep the MagRect rotated to align to a plane with normal vector

$$\boldsymbol{n} = (\boldsymbol{x}_R - \boldsymbol{x}_L) \times \frac{\boldsymbol{u}_R + \boldsymbol{u}_L}{2}$$

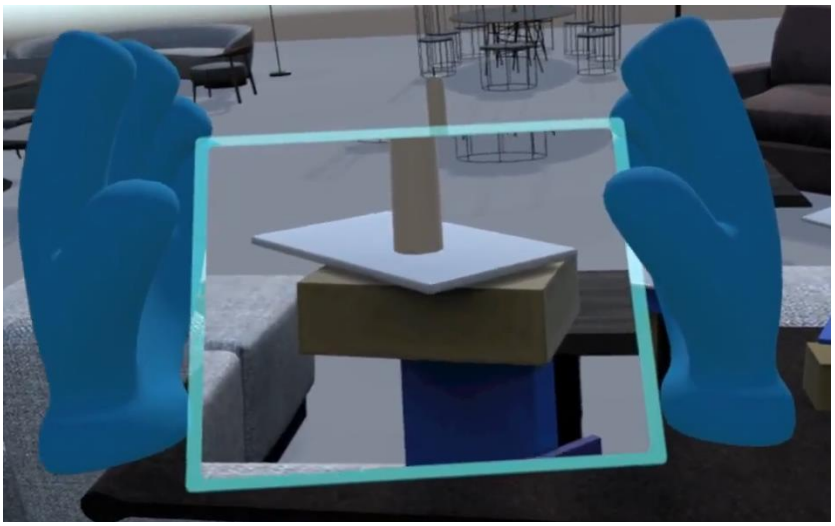

Figure 2: Screenshot of an early implementation of the MagRect.

Finally, to imitate the view of a real magnifying glass, the view of the MagRect rotates to display what is "in front of" the user with respect to his head. In practical terms, the rotation of the MagRect's camera always aligns with the local positive z-axis of the user's head (i.e. the direction of the tip of his nose).

3.3 Finding the target of the user's gaze

To get an idea of "what the user is looking at", it's necessary to create a mapping between the gaze ray data provided via Tobii XR and a correct "gaze target position" in world space. From there, we want gaze targets that fall on the surface of the MagRect to be mapped to another world space position corresponding to the point displayed at the render texture coordinate of the first gaze target.

Without considering the MagRect, the simplest way to accomplish the first step of this process is to perform a ray cast of this ray and consider the returned intersection point (where the ray first intersects the MagRect's Collider) as the gaze target. Although some research (notably [17]) has suggested that using gaze-ray intersections alone is not an ideally accurate solution for determining what a user is looking at, after implementing the gaze smoothening techniques detailed in section **3.4.3**, this method empiricially proved more than accurate enough for our purposes.

Thus, given a gaze target position on the surface of the MagRect, we map it to viewport coordinates relative to the MagRect's camera, i.e. into a 2D coordinate in range [0, 1] relative to the visible bounds of the camera. Finally, we cast a ray from the camera's near plane going through this viewport coordinate, and treat its first intersection with a Collider as our final gaze target position.

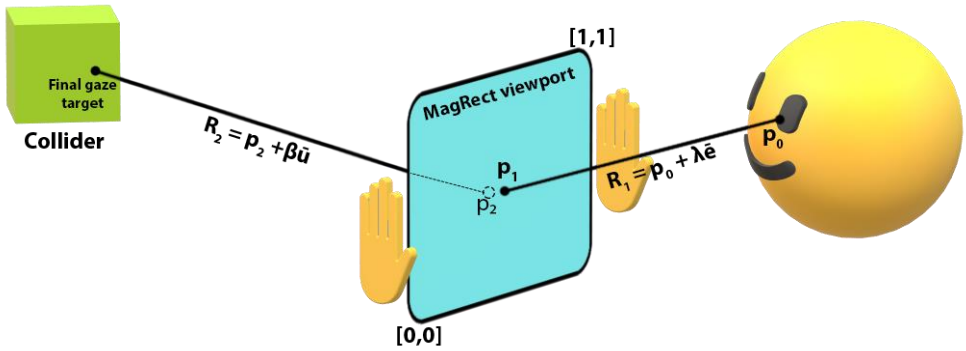

Figure 3: Illustration of the rays and intersection points involved in finding the user's gaze target through the MagRect. The direction of the user's gaze, $\bar{e}$, is provided by the Tobii SDK, and ray R_2 is formed by going from the MagRect camera's near clip plane (p_2) to the viewport coordinate corresponding to p_1.

The above method for determining gaze targets through the MagRect proved highly effective and was used for the final solution, but it does require the scene to be constructed specifically with the technique in mind. It is reliant on having a scene where each visible object is equipped with a Collider component that is tightly fitted to its mesh. Aside from being a potential detriment to performance, this caveat could pose an issue for non-convex meshes and for objects contained inside other objects, resulting in gaze targets being registered that do not correspond exactly to what the user is looking at. This did not ultimately prove to be a problem while testing the solution, nor for the experiment outlined in **Chapter 4**, but is worth noting.

3.4 Implementing gaze-based magnification

3.4.1 Initial experimentation

As no pre-existing published work (or indeed any informal sources or documentation) on creating a dynamic magnification effect in virtual reality seemed to exist, the final solution was arrived at through iterative trial and error.

The simplest magnification system implemented was to simply move the MagRect's camera backwards and forwards along the forward direction of the user's head to zoom in and out on objects on the other side. This implementation was tweaked and iterated on, but ultimately proved unconvincing for several reasons: Firstly, the effect did not give the impression of "magnifying" objects, but rather quite transparently gave the appearance of forward locomotion. Moreover, it did not conform to the occlusion behaviour one would expect from looking through a magnifying glass, as the camera could, for example, move *through* objects and emerge on the other side, revealing parts of the image that

would be obscured from the user's position. Mitigating the latter problem would have required implementing a highly complex workaround, and so the approach was scrapped.

A more fruitful approach to magnification was found by keeping the MagRect's camera in place and simply adjusting its field of view (FOV). This produced a convincing magnification effect which is easily explained: Since reducing the FOV reduces the size of the camera's view frustum without reducing the size of the display, the resultant image appears to enlarge the environment being captured. Indeed, the inverse relationship between optical magnification (M) and FOV is well established in the study of microscope imaging [18], i.e.

$$M \propto \frac{1}{FOV}$$

For the purposes of this user interface design, establishing this relationship is sufficient, as the exact reduction of FOV necessary to produce a given magnification factor can be ascertained through trial and error (and likely does not exactly correspond to the real optical phenomenon).

Consequently, define $\mathcal{M}$ here to denote *magnification within the context of this system's design*, regardless of its real-life counterpart. To normalise $\mathcal{M} = 1$ (no magnification) to a 60° FOV, the following relationship was used as a standard for the final solution:

$$FOV = \frac{60°}{\mathcal{M}}$$

Initial experimentation with adjusting the MagRect's camera's FOV as magnification was done with touchpad input. The effect was convincing within a limited range of values for $\mathcal{M}$, but created severe distortions outside it. Furthermore, linearly adjusting FOV did not produce a transition in magnification levels equivalent to moving a magnifying glass back and forth (which is not a linear transition).

3.4.2 The simple magnifier replica (Natural magnification)

Hoping to more closely imitate the behaviour of a magnifying glass, attempts were made to compute magnification using the equations and parameters that control the real optical phenomenon – this technique is referred to here as "natural magnification". Although it does not use gaze data to control magnification at all, implementing it was a necessary step to arriving at one of the final magnification techniques used for experiments ("combined magnification" described in section **3.4.4**).

As summarised in section **2.2.1**, optical magnification of a magnifying glass for objects at a near point distance (25cm) from the eye, is a function of the lens's focal length f, the distance between the displayed virtual image from the lens d, and the distance from the eyes to the lens, l.

For our purposes, assume the only time-varying parameter is $l(t)$, which is easily computable every frame. The values of the remaining parameters f and d were assumed constant, and arrived at by experimentation. Thus, we define natural magnification of the MagRect for frame t of the simulation as

$$\mathcal{M}_{natural}(t) = \left(\frac{0.25}{L(t)}\right)\left(1 + \frac{L(t) - l(t)}{f}\right), \qquad L(t) = |d - l(t)|$$

This produced an effect similar to a real-life magnifying glass, where moving one's hands backwards and forwards increased and decreased magnification of the image displayed on the MagRect. Tweaking the values of f and d has the effect of changing the maximum and minimum magnification factor, and thus the rate of change in magnification as the hands are moved back and forth.

Of course, this implementation is a simplification that treats every point seen by the camera as being a fixed distance away. Hence, it does not replicate the exact transition in magnification levels, nor the focus, blur and optical abberations present when looking through a real magnifying glass. This allows the user to see a perfectly crisp view of the other side of the MagRect, while holding it at angles and distances where only a blur would be visible in reality. Since this additional visibility was only beneficial, however, no attempts were made to create a more realistic implementation.

3.4.3 Smoothening gaze input data

It was evident from the beginning that the raw data from the eye tracker provided every frame was prone to jitter. This may be a consequence of tracking inaccuracies, but is perhaps to be expected based on the nature of eye movements: These are often involuntary, not least because moving one's eyes is necessary to actually see the environment one may want to zoom in on. Consequently, there was a clear need for some method of smoothening or averaging the gaze data over several frames so that outliers were filtered out, and a small delay between looking and zooming was felt by the user.

Initial unsatisfactory attempts at solving this by keeping a moving average of gaze distances highlighted the need for some way of visualising the current gaze target on the other side of the MagRect in real time. (Displaying debugging text on-screen was of little use, as looking at the text would distort the beahviour of the system.) Consequently, the *GazeDot* object was created and added to the

solution – this is a small 2D circle sprite that would display on top of the target gaze position so long as the MagRect was active. While immensely valuable during development and for subsequent debugging, the GazeDot proved to be a helpful visual aid to control magnification (and ultimately the teleportation mechanic outlined in section **3.5**) in the final solution as well. The general utility of such a visual feedback mechanism for directing gaze is supported by [8].

This in turn allows the "gaze distance" parameter (how far away the object looked at is from the MagRect) that controls the magnification factor to be expressed as a function of the GazeDot's position. For simplicity, assume a linear relationship between the magnification factor and the distance of the GazeDot to the user,

$$\mathcal{M}_{gaze}(t) \propto Dist(t)$$

One candidate implementation of $Dist$ was to keep a buffer of the last n gaze target positions, and their respective distances from the user. This solution would wait for the buffer to fill up, then calculate the average gaze target position and distance based on the buffered values, then interpolate the position of the GazeDot and the zoom factor between the previously calculated averages and the new ones. The interpolation delay, during which the buffer would re-fill with the newest gaze data, made this method similar to established "gaze-dwell" techniques (as described by [7]), whereby eye movements are only acknowledged as input after some small period of gazing at the same target.

While this method was effective at filtering out outliers in gaze data, the added delay, during which the user could move his eyes without seeing the action immediately acknowledged in the MagRect, resulted in a jagged user experience. Moreover, a common drawback that became evident with this and other attempted solutions, was that while zooming in on distant objects by gazing at them through the MagRect was a simple task, the opposite operation was more difficult: Having already zoomed in on a tiny section of the initial view, zooming back out again would require shifting one's gaze to an object closer to the MagRect, which might be difficult to manoeuvre inside a minimised FOV.

In the end, the implementation that proved most effective was a variation on the mechanics of the "inertial reticle" detailed by [9]. Here the final position of the GazeDot is determined by a constantly moving average of the last n gaze target positions. Denoting the gaze intersection point for frame t by $\boldsymbol{C}(t)$, the position of the GazeDot is set to

$$\boldsymbol{x}_{GD}(t) = \sum_{i=t-n}^{t} \frac{\boldsymbol{C}(i)}{n}$$

This is similar to the previous method, but rather than waiting for the buffer to fill up before calculating the average, a circular buffer is used to cache gaze positions and the average is recalculated every frame. This immediately takes into account the newest gaze data, so the GazeDot smoothly tracks changes in user gaze, while also refraining from assigning excessive importance to data for any given frame. In initial implementations, "gaze distance" was simply measured as the straight-line distance between the user and the GazeDot, i.e.

$$d(t) = \left\| \boldsymbol{x}_{GD}(t) - \boldsymbol{x}_{player}(t) \right\|$$

While this moving-average solution was effective at filtering out outliers, it still did not provide the desired delay between glancing at a far-away object and activating a large amount of zoom on it. This is partly because the moving-average does not weigh more recently sampled gaze positions any less than older ones, so the sudden introduction of a large (or small) distance can shift the average very quickly.

A method of delaying the impact of newer samples without resorting to gaze-dwell mechanics was developed by setting the $Dist$ measure to an exponential moving average by adding an extra constant factor α (in range (0, 0.5)). Here a moving average distance measure is calculated every frame (but using a larger buffer size m, $m > n$), but its contribution to the final $Dist$ value is only a fraction of the total, the rest being contributed by the previous total. In this way, $Dist$ changes slowly in the beginning in response to a change in gaze position, but eventually converges to the new, larger distance, giving users an opportunity to redirect their gaze elsewhere before the FOV shrinks in on a given target.

$$Dist(t) = \alpha \sum_{i=t-m}^{t} \frac{d(i)}{m} + (1 - \alpha) Dist(t - i)$$

Empiricially, this solution proved most comfortable, as it maintained real-time user feedback to gaze changes, but did not change magnification levels too quickly to allow users to see their environment before zooming in. The pseudocode in **Code listing 1** summarises the algorithm for calculating $Dist$ for a given frame.

```
float WeightedAverageDist()
{
    distBuffer[frameIndex] = Distance(player.position, gazeDot.position);
    frameIndex = (frameIndex + 1) % m;

    newAverage = 0;
    for (i in [0..m-1])
    {
        newAverage += distBuffer[i];
    }
    newAverage /= m;

    dist = (newAverage * alpha) + ((1 - alpha) * oldDist);
    oldDist = dist;

    return dist;
}
```

Code listing 1: Pseudocode for calculating the *Dist* value that determines the scale of magnification each frame.

3.4.4 Final magnification solutions

Putting together the components already described, the final implementation of gaze-based magnification amounted simply to

$$\mathcal{M}_{gaze}(t) = 1 + \lambda Dist(t)$$

In other words, the magnification factor scales linearly with gaze distance, and is capped to a minimum value of 1. The value of the scaling parameter λ, which controls both the maximum magnification scale, as well as the rate of change of the scale, should be adjusted to the needs of a particular scene. Although attempts were made to use the moving-average $Dist$ value as a parameter for the "natural" magnification technique described in section **3.4.2**, the resulting magnification effects, with their nonlinear scaling, felt unnatural and difficult to control. Consequently, linearly scaled gaze magnification was settled on on the grounds of being most user-friendly.

Because the "natural" magnification technique by itself proved so intuitive to use, however, an alternate final magnification solution was implemented that combined contributions of gaze-based and natural magnification. This technique is referred to here as *combined* magnification:

$$\mathcal{M}_{combined}(t) = \beta \mathcal{M}_{natural} + (1 - \beta)\mathcal{M}_{gaze}$$

The resulting effect (depending on the value of parameter $\beta \in [0,1]$), is one where the MagRect's magnification is determined partially by the user shifting her gaze around the view, and partly by moving her hands back and forth. This allows her, for example, to zoom in a certain amount on a distant object by gazing at

it, then increase magnification even further by moving her hands towards her face. Likewise, this technique additionally mitigates the difficulty of "zooming out" (as described in the previous section), as by moving her hands backwards, the user can easily decrease the magnification factor (thus increasing her FOV, revealing a larger section of the environment on which to shift her gaze).

Once established, both of these solutions were tested by volunteers from the Immersive Virtual Environments laboratory at UCL. This initial feedback was positive, with some preference for the increased control provided by the combined-magnification technique and for faster zoom speed (λ). The latter was justified by noting that since the MagRect can be activated as needed, it is unlikely to be used for objects that can be seen directly from the user's current position, but that once magnification is needed for looking at a small or distant object, the MagRect should converge as quickly as possible on the object after being activated.

3.5 Implementing gaze-based teleportation

While formulating initial plans for an experimental evaluation of the solution, an additional practical requirement arose naturally: The MagRect system needed to work alongside some additional virtual locomotion technique. To be able to navigate a test scene larger than the physical room he's situated in, the user needs some mechanism for "teleporting" to a different part of the scene that he can't reach by physically walking alone (and which may be obscured from being zoomed in on with the MagRect).

A variation on a common teleportation technique in HMDs (as described by [11]) was initially implemented: Here the user would aim an arc extending from his virtual hand by rotating his controller, then press a button to teleport to the point on the floor where the arc converged. This mechanic was implemented such that only locations where a character could reasonably stand upright would be valid teleport destinations, and ensured the user would never appear inside (or immediately in front of) a virtual object. By itself, this solution was highly effective for teleportation, both in speed and accuracy. After several sessions of testing, however, it became clear that the technique seemed to be distracting from use of the MagRect. Hand-based teleportation required a user to look up from the MagRect (thus deactivating it), aim his controller, teleport, and then reactivate the MagRect to use magnification again. The wide popularity of this teleportation technique in VR games also inherently meant users would be more familiar with it, and might prefer to view objects up close through constant teleportation, rather than taking the time to get accoustomed to zooming in on them with the MagRect.

As a result, it was decided that a novel teleportation mechanic that integrated directly with the MagRect should be implemented. The design for this mechanic was as follows: The user would use the GazeDot as an indicator of where he wanted to teleport to, then hold down the trigger button on his right controller to initiate teleportation.

In addition to the standard considerations for VR teleportation (notably adjusting the position of the *playspace* around the teleport destination) [11], this gave rise to the following requirements for a gaze-based teleportation implementation:

- Teleportation should always move the user to a position on the floor (i.e. at $y = 0$), and never "inside" or immediately in front of another object.
- The user can specify a teleport target at eye level (via the GazeDot), and it is up to the system to *adjust* this target to a valid destination point, if possible.
- The user should have visual feedback as to 1) whether a teleport target is valid, and 2) the exact location he will end up after teleporting.
- A teleport target selected by the user is valid if 1) It is not higher than twice the height of his eyes, and 2) After *adjustment*, the destination point is not still inside of, or immediately next to, another object.

The exact algorithm to solve this was again arrived at through trial and error, after observing which parameters allowed the user to select teleport targets freely without arriving at an awkward or unrealistic destination. This ultimately involved performing two processing steps for each selected target to meet the two conditions for validity.

This algorithm is summarised in pseudocode in **Code listing 2**. Here the *Update* function is called by the engine every frame, and the functions *IsTargetValid* and *IsDestinationValid* perform the two validity checks respectively.

```
void Update()
{
    if (rightTriggerPressed)
    {
        target = LastGazePosition();
        if (IsTargetValid(target))
        {
            target.y = 0;
            destination = GetAdjustedPosition(target);
            teleportMarker.position = destination;
            if (IsDestinationValid(destination))
            {
                teleportDestination = destination;
                isTeleportPending = true;
                holdDownTime = 0;
            }
         }
     }
     if (isTeleportPending)
     {
         holdDownTime++;
         if (holdDownTime >= activationTime)
         {
             Teleport(teleportDestination);
         }
     }
}
```

Code listing 2: Pseudocode for gaze teleportation.

The teleport target selected by the user is adjusted through a separate process. By constructing a virtual capsule around the target and checking its intersections, all nearby Colliders are inspected. From here (with the help of Unity's *ComputePenetration* function), the vector necessary to separate the capsule from the Collider is calculated. The largest vector found is used to shift the target away from the Collider to form a valid teleport destination. This algorithm is summarised in the pseudocode in **Code listing 3**.

```
Vector3 GetAdjustedPosition(target)
{
    largestDist = 0;
    shiftDirection = Vector3.zero;

    adjacentColliders = OverlapCapsule(target, myCapsule);
    foreach (collider in adjacentColliders)
    {
        direction, distance = ComputePenetration(myCapsule, collider);
        if (distance > largestDist && Angle(-localY, direction) > 45)
        {
            largestDist = distance;
            shiftDirection = direction;
        }
    }

    largestDist += epsilon;
    return target + (largestDist * shiftDirection);
}
```

Code listing 3: Pseudocode for adjusting the position of a teleportation target selected by the user.

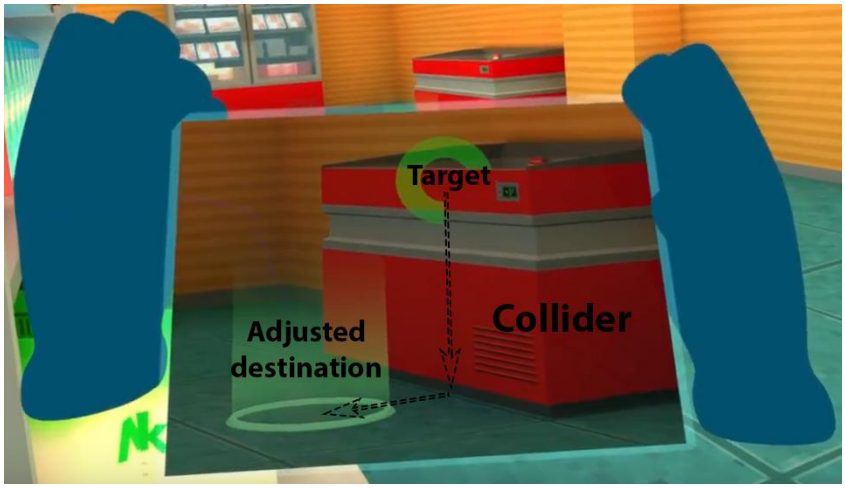

Figure 4: Illustration of the gaze teleportation mechanic, where a user has selected a teleportation target with the GazeDot, and the final teleport destination (indicated by the green halo) has been adjusted to floor height, and away from the closest Collider (the ice box).

3.6 Usability refinements

In the interest of creating a smooth and intuitive experience of the system for first-time users, additional effort was made to optimise presentation and to provide as much visual aid as possible.

In order to keep the MagRect's magnification levels synchronised to what the user is looking at as often as possible, a separate buffer of gaze target positions

from the user's Camera (with its own moving-average gaze target estimate) is kept even when the MagRect is not being used. When the MagRect is activated again (after a threshold delay), its gaze target buffer is initially populated with the contents of this backup buffer, as a best guess to calculate its initial magnification level. The MagRect also smoothly fades into view when the user looks at the space between her hands and fades out again when not in view.

The GazeDot, being used as both a gaze-directing visual aid for magnification, and a target selection indicator for teleportation, was adjusted to give as much useful visual feedback as possible. When looking around the environment, the GazeDot adjusts its opacity depending on the distance to the gaze target, fading into transparency when inspecting objects up close (so as not to obscure the view), and becoming fully visible when looking at targets far away, when more precise gaze control is required. The GazeDot will show up as red or green, depending on whether its current position represents a valid teleportation target. It is rendered as a hollow circle, and when the user holds down the trigger to initiate teleportation, the circle gradually fills up during the confirmation time (500 ms) until teleportation takes place.

During teleportation, a short white flash briefly fills the screen to minimise the jarring effect of the "jump" from one position to another. An additional teleportation effect was implemented initially, where the MagRect would rapidly zoom in to an exaggerated extent on the target position during teleportation, to give the appearance of being catapulted to the destination – but although enjoyable, this effect was reliably found to produce motion sickness and was ultimately dropped.

Chapter 4

Experiment design

4.1 Experiment aims

In order to assess the utility of the magnification-vision technique, a study was proposed to investigate whether users found the system useful and whether it improved their performance in a cognitive task. Given that two slightly different gaze-controlled magnification techniques had been developed, a secondary aim was to test whether one performed better than the other. Constraints on total experiment time necessitated having each subject test each technique in a single session.

This entailed designing a puzzle-based experimental scenario where subjects could use the MagRect freely, and performance could be measured in terms of time taken to complete the task. More indirectly, to assess the extent to which subjects used the MagRect spontaneously, it was decided to create an experiment where magnification might be helpful, but not essential to solving the task.

4.2 Experiment setup

18 subjects (5 female; median age 23) from UCL were recruited as volunteers to participate in the study.

As the base virtual environment for the experiment, a large supermarket scene from the Unity Asset Store [19] was used: This was fully modelled, textured and lightmapped, and contained separate food sections (meat, dairy, pantry, etc.) already filled with appropriate objects.

The task designed was a simple object-search game, where five custom-made items were hidden in disparate sections of the supermarket. These items were similar enough to the existing grocery items to blend in, but distinguishable upon closer inspection. The subjects' task was to walk around the supermarket and find these special items, which could be identified by comparing them with a dynamic "shopping list" attached to their left hand that could be brought up anytime. Items would automatically register as "found" after being gazed at continuously for more than two seconds, and the session ended after all five items were found.

Figure 5: Screenshots from an initial version of the experiment scene, showing a hidden item on a shelf (left) and the shopping list display (right).

Each subject performed the task three times using different implementations of the MagRect: Once with pure gaze-based magnification, once with "combined" magnification, and once with no magnification at all. This third technique was used as a control set, in which the magnification factor is always set to 1, and the MagRect is only used to direct the GazeDot for gaze-based teleportation. The experiment was estimated to take 60 minutes in total.

To prevent subjects' increased familiarity with the search environment on subsequent trials from biasing performance data, each subject would use each of the different magnification techniques (Gaze, Combined and None) in one of six different orders for their three trials. Three different permutations of item locations were also created, so that items would not be in the same location for multiple trials by the same subject (these permutations were presented in the same order for each subject). For each subject's first trial, a short tutorial section with an orientation task was implemented, in which subjects could familiarise themselves with using the MagRect before the main search task began.

The task was performed with the VIVE Pro Eye HMD, and a calibration routine for the device's eye tracker was run on each subject before each search session (three times). Effort was made during the experiment to provide subjects with only the minimum guidance necessary to use the MagRect and perform the task successfully. In cases where subjects seemed to be getting frustrated or taking excessive time finding an item, additional guidance was given: Out of the 18 subjects, 8 were eventually given outside hints about the location of one or more items during the experiment.

In between each of their three trials, subjects filled in a questionnaire assessing the magnification technique they had used – this included a complete System Usability Survey (SUS) [20] for each technique. After completing all trials, subjects were given a final questionnaire where they were asked to rank the techniques from most to least useful, and to justify their response. Each questionnaire also included the option of additional longform feedback. (A copy of each questionnaire is included in **Appendix B**.)

Data on subjects' performance during each trial was recorded through a logging system (see next section).

4.3 Software additions for experiment

4.3.1 Logging user data

A logging system was implemented to keep track of all relevant user behaviour during experiments. Each run of the application created a separate log file, with data output in a JSON format. A master class (in C#) would be called by disparate parts of the system whenever relevant data became available, while the master synchronised output of each data chunk at the end of every frame.

The data logged included position and rotation of the user's head and hands, controller input, the state of the MagRect and teleportation targets, as well as information about which items were visible, and when each was marked as "found". Several composite values (e.g. averages and local positions) were output directly from Unity, as recalculating them afterwards without global knowledge of the scene's state would require additional complexity. A parser was implemented in Python to extract relevant data after experiments were complete.

4.3.2 Additional implementations

To detect whether a hidden item is registered as "found", the computed average gaze target position for each frame from the main camera (described in section **3.6**) was used. Each frame, a check is made to see whether a hidden item is within a 20 cm radius of the current gaze target – if the same item remains within range for a continuous period of 2 seconds, the item is "found". This was found to be an effective middle ground that prevented items from being found "accidentally", while still registering a user's genuine discovery quickly. The C# event system is used to update the state of the system when an item is found, updating the shopping list GUI to check off the found item, and ending the game automatically when all five items are found.

To supplement the data that could be recovered from logs, functionality was also added to keep track of the user's relationship to the hidden items. By checking

each frame whether an item was not obscured, and within the viewport of either the user's main camera or the MagRect's camera, it was possible to keep track of when an item was visible. This allowed for gathering data about whether an item was first "seen" head-on or through the MagRect, as well as count the number of times an item was visible to the user but was not spotted (an "item miss").

Chapter 5

Experiment results

5.1 Performance data

	Gaze		Combined		None		p
	mean	SD	mean	SD	mean	SD	
Task time (s)	628.38	300.62	593.33	213.61	516.33	220.28	0.39
Mag. time/total time	0.83	0.10	0.83	0.08	0.84	0.09	0.93
Num. teleportations	48.33	19.38	47.44	17.69	42.72	15.42	0.59
Distance covered (m)	393.27	162.12	379.58	93.85	315.76	93.02	0.13
Num. item misses	30.61	17.52	27.17	9.77	28.22	13.08	0.75
Num. finds through MagRect	4.33	0.77	4.39	1.04	4.44	0.86	0.93
Raised hands time/total time	0.59	0.27	0.60	0.21	0.58	0.20	0.96
Mean item-finding distance (m)	1.99	1.14	1.73	0.71	1.33	0.32	0.05

Table 1: Performance results by magnification technique

Performance data were parsed from session logs and grouped by magnification technique. Data for each technique were compared using a one-way ANOVA, which found no significant difference in any performance measure between conditions (with $\alpha = 0.05$). A two-way ANOVA (without replication) conducted on task completion time also did not find a significant difference, either between conditions ($p = 0.43$) or between subjects ($p = 0.80$).

These results are summarised in **Table 1**. Note that "task time" was measured from the start of the main search task until the last item was found (time taken to complete the tutorial for each subject's first trial was not taken into account). "Distance covered" was measured as the cumulative straight-line distance moved by the subject's head based on an average position measure every 90 frames. "Mag. time/total time" is the ratio of time the MagRect was active during a session to the total task time. "Num. finds through MagRect" denotes how many items were "found" by being looked at through the MagRect. Finally, "mean item-finding distance" is a measure of how far away from an item on average subjects were when it was registered as "found".

A Shapiro-Wilk test on the "completion time" data for each technique indicated that data for Gaze and Combined magnification where normally distributed ($p = 0.14$ and $p = 0.08$ respectively), but that data for None-magnification was not ($p = 0.02$). The distribution of completion times is shown in *Figure 6*.

A power analysis comparing completion time data between the Combined and None-conditions indicated that a required sample size of 250 participants would be required to show a significant result with a statistical power of 0.8.

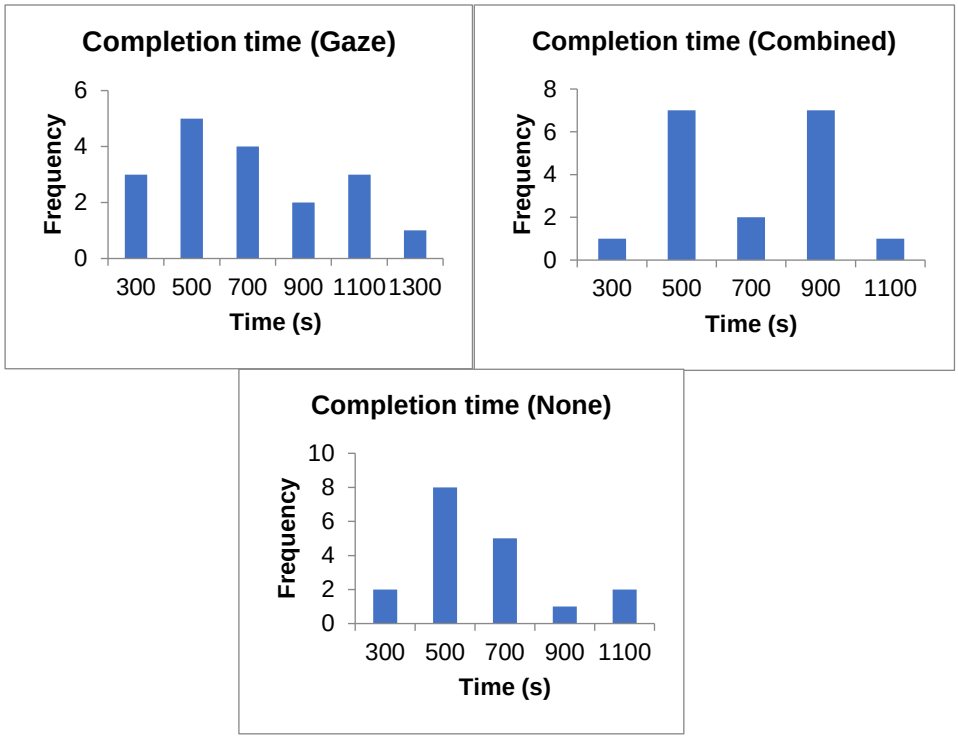

Figure 6: Task completion time by magnification technique.

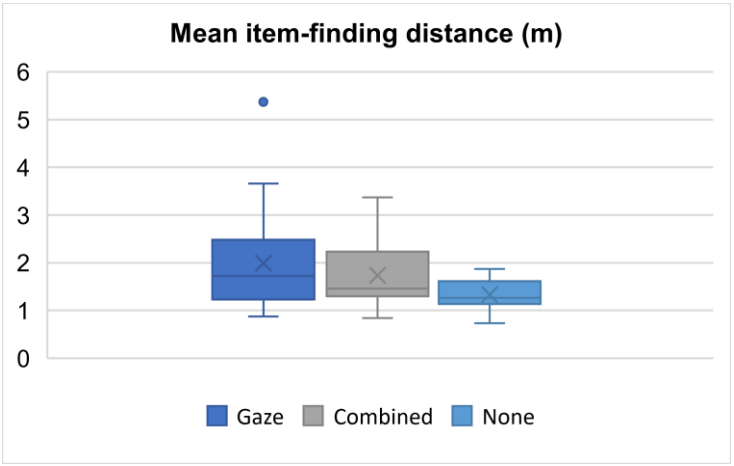

Figure 7: Average distance between a subject and an item when the item was found.

Regrouping the data by trial order (i.e. disregarding which magnification technique was used), the one-way ANOVA does indicate a significant difference in completion time between subjects' first, second and third trial, as seen in **Table 2**. A Shapiro-Wilk test performed on this data grouped by task order indicates that the first and second trials are normally distributed ($p = 0.98$ and 0.26), but that data for the third trial is not ($p = 0.02$).

	1st		2nd		3rd		p
	mean	SD	mean	SD	mean	SD	
Task time (s)	789.52	233.78	528.59	190.53	419.93	155.31	**0.00**

Table 2: Task completion time by trial number.

5.2 Multiple-choice responses

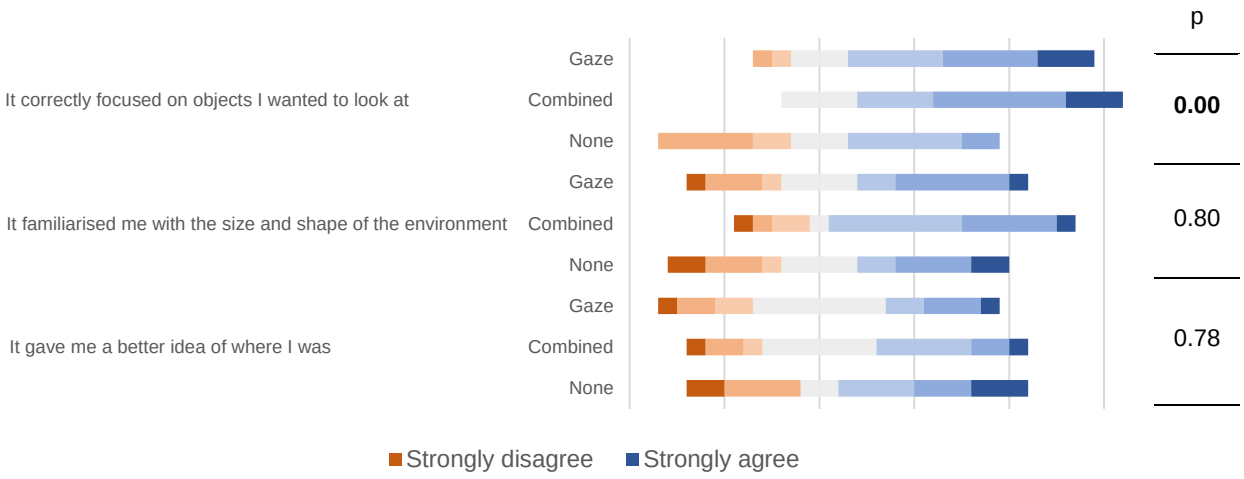

Figure 8: Multiple-choice responses on a 7-point Likert scale.

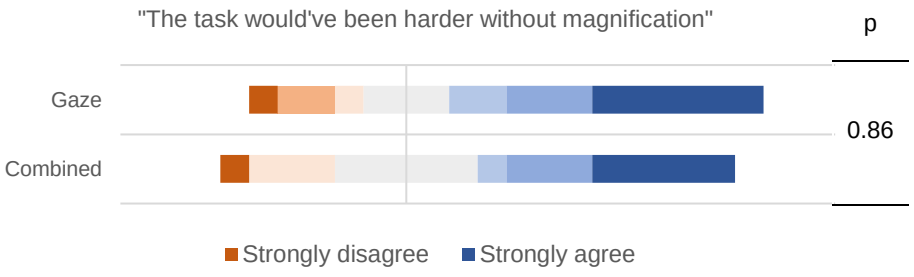

Figure 9: Multiple-choice response on a 7-point Likert scale.

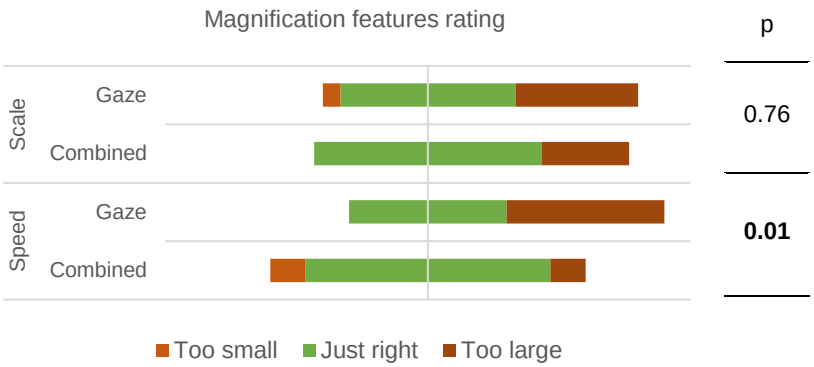

Figure 10: Multiple-choice response rating the scale and speed of magnification for the two magnified techniques.

	Gaze		Combined		None		p
	mean	SD	mean	SD	mean	SD	
SUS score	71.57	16.38	76.76	10.64	72.96	15.84	0.54

Table 3: Average calculated SUS scores for each condition.

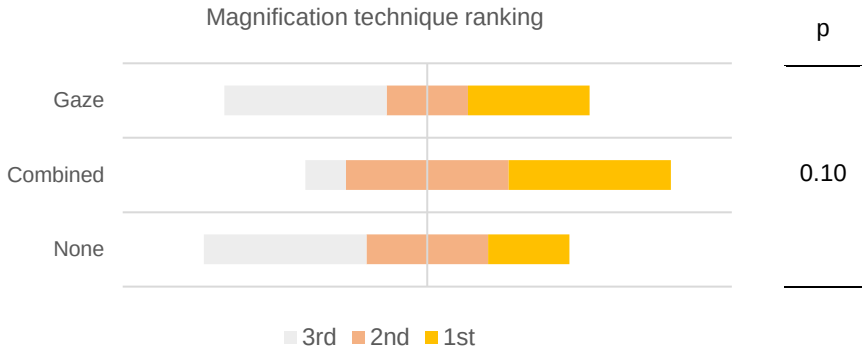

Figure 11: Subjects' aggregated ranking of the three conditions.

5.3 Longform responses

For a full list of subjects' freetext responses in questionnaires, see **Appendix C**; some of the most notable and repeated responses are highlighted here.

Subjects who rated Combined magnification highest often cited the ability to deliberately control the magnification factor as justification. Furthermore, a desire for more "control" over zooming was expressed by five different subjects, who suggested that the method might be improved by having an additional input

modality for "controlling" the MagRect, such as joystick or button input. Less commonly, several subjects who preferred Gaze magnification justified their ranking by calling the method "simpler".

One notable subject response was "I used the magnifying glass more for transportation than magnification", and this was echoed multiple times ("Teleportation was all I needed" was mentioned by another). This was expressed more frequently (but not exclusively) among subjects who preferred the None-condition.

More common feedback for the None-condition was that it was difficult to control, especially when attempting to teleport to a distant location, partly because the destination was more difficult to see without magnification.

Chapter 6

Discussion and evaluation

6.1 Results discussion

As no significant performance change could be observed from use of gaze-assisted magnification techniques, the study has not been able to find a practical benefit to the system.

It is important to note that task completion time is the primary measure of subject performance, and that it influences the value of most of the other values for the same condition. Since a slightly lower (but not significantly so) task time was observed for subjects in the None-condition, this is likely to have affected the average distance moved and number of teleportations and item misses – especially since none of the other measures proved significant on their own either.

The performance-related measure which is closest to statistical significance is "item-finding distance", which suggests, for unsurprising reasons, that subjects with Gaze or Combined-magnification preferred to stand some further distance apart from items and zooming in on them, rather than teleport right up to them (as subjects in the None-condition more frequently did). Although this may very faintly suggest a convenient time-saving benefit, the lack of other results to back it up makes it difficult to draw conclusions from.

The lack of any statistical differences in performance (and by corollary the very similar mean values for each measure across conditions) is likely explained by the strong coupling between magnification and teleportation. In the current system design, there is no clear way to tell whether a user is activating the MagRect and directing her gaze around it to zoom in on an object or to select a teleportation target, which makes interpreting data about just one of the two use cases difficult. Since teleportation around the supermarket scene is necessary to complete the task, but magnification is not, subjects likely behaved similarly (in patterns of teleporting, looking around their new position, and teleporting again) across conditions, even when the MagRect did not zoom at all. Recalling the sentiments of several subjects (section **5.2**) who seemed to find themselves using the MagRect more for teleportation than static magnification, further reinforces this conclusion.

On the other hand, answers to multiple-choice questions indicate some user preference for MagRect conditions with magnification (Combined magnification especially). The questions where this is significant relate to the ability of the

MagRect to correctly focus on desired targets, and to the speed of magnification. These again relate to the most clearly visible strength of Combined magnification in being the technique over which subjects felt most "in control". There seems to be a similar tendency (though the result is not quite significant) in subjects' rankings of the three conditions, where Combined magnification was ranked most favourably. The preference for magnification (in either form) is not clearly reflected in responses to SUS questions, however.

Given these results, one could tentatively note that while the ability to magnify their surroundings did not appear to alter their behaviour, subjects still preferred the effect to be present. This may be the case even if magnification was used only as an aid to teleportation, rather than for its own sake to inspect objects more closely, as it appears to have made it easier to exactly control and visualise the target of a teleportation to a point far in the distance.

6.2 Experiment evaluation

In light of the inconclusive results acquired from the user study and with knowledge of which factors contributed to them, it's possible to assess the effectiveness of the experiment overall.

As indicated in the previous section, some ambiguity exists in the data regarding the extent to which the MagRect was used primarily for teleportation, instead of as a tool to inspect distant objects up close. This ambiguitity is partly a consequence of the MagRect's final design, but there were factors in the design of the experiment that may have encouraged teleportation over magnification. In most cases, the items subjects were required to find could be identified without the need for magnification – standing immediately in front of the item was sufficient to distinguish it from its surroundings. Once subjects had familiarised themselves with how to control teleportation (which was necessary), they could just as well teleport up to an item rather than zoom in on it from afar. This may have been exacerbated by the fact that the supermarket layout consists of ailes that are too high to look above, and ice-boxes that are too high to look inside of from a distance. This seems to have encouraged frequent teleportation rather than surveying the environment from a static location.

Having considered the factors that seemed most clearly to have affected the nature of the resulting data, however, it can be noted that the experiment process itself went very well. The nature of the task was self-explanatory enough to require minimal additional guidance for subjects beyond initial instructions; the same held true for using the MagRect itself (in each of its variations). The large size and homogenous layout of the supermarket forced subjects to pay careful attention to their surroundings, which was germaine to the system under test –

even if this occasionally led to a single hidden item in an unexplored corner of the environment being the bottleneck for task completion.

6.3 Project evaluation and future improvement

While no evidence to the utility of gaze-directed magnification in VR could be gleaned, several important design considerations for future systems with similar designs have been highlighted. The system has proved fun to use. An unexpected utility of gaze-based magnification indicated from experiments should also be noted: Offering more fine-grained control and visualisation of a distant target destination for teleportation-based locomotion.

6.3.1 Changes to experiment

Were the experiment to be recreated, there are clear changes that could be made to increase the likelihood of a significant result.

Most notably, the experiment scene should be redesigned to increase the utility and impact of magnification. For example, the environment could be made small enough to fit inside a physical room (removing the need for teleportation), and the items could be made to blend more subtly into the environment (requiring more careful inspection to identify them). In such a setup, the MagRect would be used purely for magnification, but would still be strictly optional, as subjects could still identify items with careful up-close examination – allowing the utility of magnification to be studied in isolation.

Even within the supermarket scene used for this subject's study, changes could be made to disambiguate teleportation and magnification usage data. While traditional VR controller-pointing teleportation mechanics proved to be distracting from use of the MagRect, another bespoke teleportation mechanic could be implemented that is still entirely separate from magnification. For example, the virtual scene could be divided up into a grid, with each square being the size of the user's playspace. From here, teleportation could be done with a button press indicating which adjacent square one wants to teleport to. A slightly more involved and less convenient locomotion technique such as this could not be construed as a replacement for magnification, but would still allow for use of a virtual environment much larger than a physical room.

With the larger size of the supermarket scene still in mind, there are additional mechanisms that could be added as aids to make the task more user-friendly without affecting the validity of the data. Similar to how gaze data is used to inform navigation aid in [21], the GazeDot could leave a trail in the environment to indicate which parts of a scene have already been inspected through the MagRect. This would give subjects clear visual indication of where they have

already searched, making their efforts more efficient, and causing less frustration from missing a single spot in an already visited area.

6.3.2 Changes to magnification

As already indicated (throughout **Chapter 3**), there are numerous ways the current MagRect system can be tweaked, and the varying of each parameter used or control convention chosen could each constitute a valid study in themselves. At a higher level, however, there are possible changes that have revealed themselves throughout the project that may bring improvements and are worthy of being explored.

Firstly, the parameters that control the scale of magnification are currently set once per scene, and depend only on the distance to the object under inspection. This means, for instance, that a blank wall and an open book placed the same distance from the user, will be magnified by the same amount (once the algorithm described in section **3.4.3** converges) – even though the latter might need to be magnified much more to be legible by the user. To remedy this, one could implement a system whereby objects (or regions of space) are tagged based on how much they should be magnified, but this would require excessive offline preparation for each scene, and it isn't clear whether these differing magnification levels will seem intuitive to the user. A simpler, and more generally applicable solution to this would be to slowly increase magnification parameters over time as the user glances at the same target position for a sustained period. Assuming that this sustained focus indicates interest, the user will thus get a progressively closer view of the target of their gaze over time, lessening the need to set rigid zoom factor parameters for each scene.

One area with clear potential for beneficial exploration that builds on this idea is to implement a magnification-aware texturing system. Building on the concept of gaze-contingent level-of-detail rendering [3], objects seen through the MagRect could be rendered with textures of a much higher resolution than normal. This would allow magnification to reveal fine details in objects that would not performance-wise be feasible to display otherwise, such as fine-printed text. To prevent artefacts, such a system would require handling for partially applying textures to objects dynamically when they are only partially obscured by the MagRect, but would only be a benefit to the utility of magnification.

The suggestion made by several subjects (section **5.2**) for more manual control of the zoom should also be taken into consideration, even if it isn't clear exactly how this should be done without making the system too complex for intuitive use. Early experimentation with controlling the zoom factor directly with touchpad input was done during development, and this seemed both difficult to

handle and significantly slower than using gaze input. With this in mind, there is still room for users to customise magnification to their preference by tweaking the parameters that control it. Most notably, the speed of magnification convergence can be opened up for users to control, in a similar way to adjusting mouse movement sensitivity in desktop video games.

Finally, there is potential to discard some of the simplifications made to approximate real magnification (as discussed in section **3.4.2**). While replicating the blurry optical abberations seen in real magnifying glasses seem to provide little practical benefit aside from a realistic visual effect, there may be utility in more closely approximating the change in magnification as a magnifying glass is moved back and forth, as this may make the effect appear more natural. This would require taking into account the distance of every point on the MagRect from the point being magnified – in other words, it would require a custom ray tracing mechanism to calculate the shading of the MagRect's virtual pixels every frame. This constitutes a complete reconstruction of the magnification system from scratch, and though there may be concern about its performance compared with the system described in this project, it would be a worthy alternative to implement that could prove easier to understand and control.

Bibliography

[1] M. Stengel and M. Magnor, "Gaze-Contingent Computational Displays: Boosting perceptual fidelity," *IEEE Signal Processing Magazine,* vol. 33, no. 5, pp. 139-148, 2016.

[2] A. Patney, M. Salvi, J. Kim, A. Kaplanyan, C. Wyman, N. Benty, D. Luebke and A. Lefohn, "Towards foveated rendering for gaze-tracked virtual reality," *ACM Transactions on Graphics,* vol. 35, no. 6, 2016.

[3] H. Murphy and A. T. Duchowski, "Gaze-contingent level of detail rendering," in *EuroGraphics*, Manchester, 2001.

[4] C. A. Chin, A. Barreto, J. G. Cremades and M. Adjouadi, "Integrated electromyogram and eye-gaze tracking cursor control system for computer users with motor disabilities," *Journal of Rehabilitation Research & Development,* vol. 45, no. 1, pp. 161-174, 2008.

[5] T. Hirzle, J. Gugenheimer, F. Geiselhart, A. Bulling and E. Rukzio, "A Design Space for Gaze Interaction on Head-Mounted Displays," in *CHI Conference on Human Factors in Computing Systems*, Glasgow, 2019.

[6] Q. Sun, A. Patney, L.-Y. Wei, O. Shapira, J. Lu, P. Asente, S. Zhu, M. McGuire, D. Luebke and A. Kaufman, "Towards virtual reality infinite walking: dynamic saccadic redirection," *ACM Transactions on Graphics,* vol. 37, no. 4, 2018.

[7] R. J. K. Jacob, "What You Look at is What You Get: Eye Movement-Based Interaction Techniques," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, Seattle, 1990.

[8] B. Velichkovsky, A. Sprenger and P. Unema, "Towards gaze-mediated interaction: Collecting solutions of the "Midas touch problem"," in *Human-Computer Interaction INTERACT*, Sydney, 1997.

[9] T. Piumsomboon, G. Lee, R. W. Lindeman and M. Billinghurst, "Exploring natural eye-gaze-based interaction for immersive virtual reality," in *IEEE Symposium on 3D User Interfaces*, Los Angeles, 2017.

[10] C. Boletsis, "The New Era of Virtual Reality Locomotion:," *Multimodal Technologies Interact,* vol. 1, no. 24, 2017.

[11] E. Bozgeyikli, A. Raij, S. Katkoori and R. Dubey, "Point & Teleport Locomotion Technique for Virtual Reality," in *Annual Symposium on Computer-Human Interaction in Play*, Austin, 2016.

[12] E. Langbehn, P. Lubos and F. Steinicke, "Evaluation of Locomotion Techniques for Room-Scale VR: Joystick, Teleportation, and Redirected Walking," in *Virtual Reality International Conference*, Laval, 2018.

[13] S. J. Ling, J. Sanny and B. Moebs, "LibreTexts University Physics, Chapter 2: Geometric Optics and Image Formation," 22 September 2019. [Online]. Available: https://phys.libretexts.org/Bookshelves/University˙Physics/Book%3A˙University˙Physics˙(OpenStax)/Map%3A˙University˙Physics˙III˙-˙Optics˙and˙Modern˙Physics˙(OpenStax)/02%3A˙Geometric˙Optics˙and˙Image˙Formation. [Accessed 10 April 2020].

[14] "Unity Real-Time Development Platform," Unity Technologies, 2020. [Online]. Available: https://unity.com/. [Accessed 26 April 2020].

[15] "VIVE Pro Eye Overview," HTC Corporation, 2020. [Online]. Available: https://www.vive.com/uk/product/vive-pro-eye/overview/. [Accessed 26 April 2020].

[16] Tobii Technology, "HTC VIVE Pro Eye Development Guide," [Online]. Available: https://vr.tobii.com/sdk/develop/unity/getting-started/vive-pro-eye/. [Accessed 26 April 2020].

[17] M. Bernhard, E. Stavrakis, M. Hecher and M. Wimmer, "Gaze-to-Object Mapping during Visual Search," *ACM Transactions on Applied Perception,* vol. 11, no. 3, 2014.

[18] X. Tao, F. Janabi-Sharifi and H. Cho, "An Active Zooming Strategy for Variable Field of View and Depth of Field in Vision-Based Microassembly,"

IEEE Transactions on Automation Science and Engineering, vol. 6, no. 3, pp. 504-513, 2009.

[19] D. Demchenko, "Supermarket Interior," Unity Technologies, 12 June 2019. [Online]. Available: https://assetstore.unity.com/packages/3d/environments/urban/supermarket-interior-38178. [Accessed 26 April 2020].

[20] J. Brooke, "SUS - A quick and dirty usability scale," *Usability evaluation in industry,* vol. 189, no. 194, pp. 4-7, 1996.

[21] R. Alghofaili, Y. Sawahata, H. Huang, H.-C. Wang, T. Shiratori and L.-F. Yu, "Lost in Style: Gaze-driven Adaptive Aid for VR Navigation," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, Glasgow, 2019.

Appendix A

System manual

The code for this project is hosted in a GitHub repository[2]. The repository was written for Unity version 2019.2.17f1, and contains library code for SteamVR, ViveSR and TobiiXR. It was built for and tested with a VIVE Pro Eye HMD running against a Windows 10 machine with the VIVE SRanipal eye tracking software running in the background.

All scripts controlling the magnification technique and the experiment described in section **4.2** are written in C#, and are contained in the *Assets/Scripts* folder. No standalone application for the project has been built, hence the application will automatically compile when launched by the Unity editor.

To run the experiment (in the Unity editor's "Play Mode"), open the Unity scene at *Assets/SupermarketInterior˙v2.5/DemoScenes/Supermarket_01.unity*. Before launch, a VIVE Pro Eye must be connected to the machine, with SteamVR running and two VIVE Pro controllers connected; the VIVE SRanipal eye tracking software must be running in the Windows system tray. No further setup is required.

The parameters of the experiment can be adjusted by selecting the *GameManager* object in the scene hierarchy, as shown in **Figure 12**.

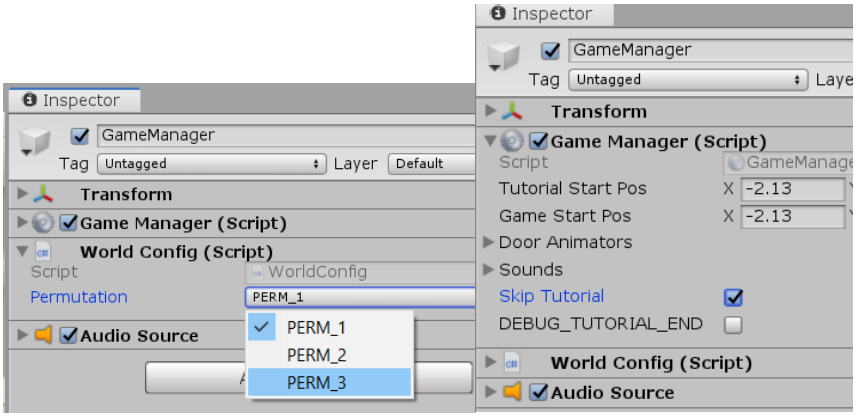

Figure 12: Adjusting experiment parameters via the *GameManager*. "Permutation" indicates which of the three item position configurations will be used. If "Skip Tutorial" is checked, the search task will begin immediately. The "Start Pos" parameters indicate where in the scene the player will appear for the start of the tutorial and for the start of the search task respectively.

[2] At the time of writing, this repository is private to GitHub user *colonelsalt*; after marking, the plan is for this to be made public via a Creative Commons Attribution licence.

The parameters controlling the MagRect can be adjusted by selecting the *Player/Magnification manager* object in the scene hierarchy, as indicated in **Figure 13**.

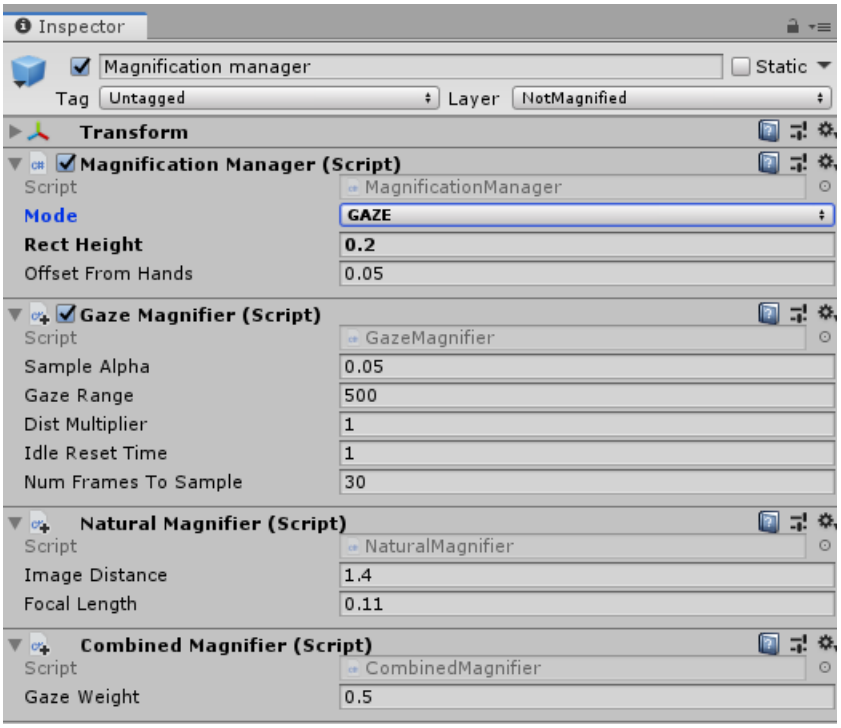

Figure 13: Adjusting magnification parameters via the *Magnification manager*. The "Mode" parameter can be set to one of the three techniques *Gaze, Combined,* or *None*. The parameters controlling each of these techniques correspond to those described in section **3.4.4**.

Every time the experiment scene runs, a JSON log file will automatically be created in the *logs* folder (as described in section **4.3.1**).

Appendix B

Experiment materials

The following questionnaires were given each subject during the study.

Background questionnaire

Background Questionnaire *Required

1. **Participant ID** *
Please ask the researcher if you do not remember your ID

Background Information

2. **What is your gender?** *
Mark only one oval.

- Female
- Male
- Prefer not to say
- Other:

3. **What is your age?** *
Mark only one oval.

- [ ] 18 - 21 years old
- [ ] 22 - 25 years old
- [ ] 26 - 29 years old
- [ ] 30 - 33 years old
- [ ] 34 - 37 years old
- [ ] 38 - 41 years old
- [ ] 42 - 45 years old
- [ ] 46 - 49 years old
- [ ] 50 years or older

4. **How familiar are you with using virtual environments?** *

Mark only one oval

	1	2	3	4	5	
Not at all	◯	◯	◯	◯	◯	Very

5. **Do you play videogames?** *

Mark only one oval.

	1	2	3	4	5	
Never	◯	◯	◯	◯	◯	Often

6. **What type of videogames do you play?** *

Tick all that apply.

- [ ] Not applicable
- [ ] First-person
- [ ] Third-person
- [ ] Other: _____

7. **How many times have you used virtual reality?** *

Mark only one oval.

- ◯ Never
- ◯ 1-2 times
- ◯ 3-5 times
- ◯ 6-10 times
- ◯ 10-50 times
- ◯ More than 50 times
- ◯ Other: _____

8. **Are there any additional comments you would like to make?**

_____

_____

_____

_____

_____

Between-trials questionnaire (Gaze and Combined conditions)

1. **Participant ID**

Please ask the researcher if you do not remember your ID

_____

2. **I felt the magnifying glass correctly focused on the objects I wanted to look at.**

(Mark only one oval.)

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

3. **I felt like using the magnifying glass helped me become familiar with the shape and size of the virtual environment.** *(Mark only one oval.)*

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

4. **I felt like the magnifying glass gave me a better idea of where I was in the virtual environment.** *(Mark only one oval.)*

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

5. **The task would have been harder if I could not zoom in on distant objects.**

(Mark only one oval.)

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

6. **When I looked through the magnifying glass, the world was usually magnified (scaled up)…** *(Mark only one oval.)*

◯ Too little to be useful.

◯ Just right to be useful.

◯ Too much to be useful.

7. **I felt the magnifying glass zoomed in on distant objects…** *(Mark only one oval.)*

- ◯ Too slowly
- ◯ At an appropriate pace
- ◯ Too quickly

8. **Did you experience any motion sickness (nausea)?** *Mark only one oval.*

- ◯ None
- ◯ A little, regardless of what I was doing
- ◯ A little, but only when using the magnifying glass
- ◯ A lot, regardless of what I was doing
- ◯ A lot, but only when using the magnifying glass

8. **I think that I would like to use this system frequently.**

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

9. **I found the magnifying glass unnecessarily complex.**

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

10. **I thought zooming in on objects with the magnifying glass was easy.**

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

11. **I think that I would need the support of a technical person to be able to use the magnifying glass properly.**

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

12. **I found the various functions in this system were well integrated.**

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

13. **I thought there was too much inconsistency in how the magnifying glass was used.**

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

14. **I would imagine that most people would learn to use the magnifying glass very quickly.**

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

15. **I found the magnifying glass very cumbersome to use.**

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

16. **I felt very confident using the magnifying glass.**

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

17. **I needed to learn a lot of things before I could get going with zooming in/out on objects.**

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

18. **Additional comments (optional):**

Between-trials questionnaire (None-condition)

1. **Participant ID**

Please ask the researcher if you do not remember your ID

_____

2. **I felt the glass between my hands correctly focused on the objects I wanted to look at.**

(Mark only one oval.)

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

3. **I felt like using the glass helped me become familiar with the shape and size of the virtual environment.** *(Mark only one oval.)*

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

4. **I felt like the glass gave me a better idea of where I was in the virtual environment.** *(Mark only one oval.)*

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

5. **Did you experience any motion sickness (nausea)?** *Mark only one oval.*

- ◯ None
- ◯ A little, regardless of what I was doing
- ◯ A little, but only when using the magnifying glass
- ◯ A lot, regardless of what I was doing
- ◯ A lot, but only when using the magnifying glass

6. **I think that I would like to use this system frequently.**

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

8. **I found using the glass unnecessarily complex.**

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

9. **I thought using the glass was easy.**

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

10. **I think that I would need the support of a technical person to be able to use the glass properly.**

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

11. **I found the various functions in this system were well integrated.**

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

12. **I thought there was too much inconsistency in how the glass was used.**

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

13. **I would imagine that most people would learn to use the glass very quickly.**

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

14. **I found the glass very cumbersome to use.**

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

15. **I felt very confident using the glass.**

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

16. **I needed to learn a lot of things before I could get going using the glass.**

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

17. **Additional comments (optional):**

Post-experiment questionnaire

*Required

1. **Participant ID** *

Please ask the researcher if you do not remember your ID

_____

2. I noticed a big difference between the three magnification techniques. *

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

2. If you had to rank the three magnification techniques in order of most to least useful, how would you rank them? * *(Mark only one oval per row.)*

	1	2	3
First technique:	◯	◯	◯
Second technique:	◯	◯	◯
Third technique:	◯	◯	◯

3. **Why did you prefer the technique you ranked as number one?** *

_____

_____

_____

_____

_____

4. **Do you have any suggestions for how to improve this magnification technique?** *

_____

_____

_____

_____

_____

Appendix C

Experiment results

This section contains a complete listing of the data from the user study, as presented in **Chapter 5**. Data analysis was done in Microsoft Excel. All performance data was parsed from application logs; questionnaire responses (with the format presented in **Appendix B**) were copied into Excel by hand.

Background information

Subject	Gender	Age	VE experience	Video game experience
613	M	22-25	1	1
445	F	22-25	3	2
378	F	18-21	3	2
584	F	30-33	2	1
850	M	18-21	2	5
916	F	22-25	3	2
669	M	22-25	4	3
510	M	22-25	4	2
762	M	18-21	3	3
224	F	22-25	3	2
521	M	22-25	3	2
291	M	26-29	3	3
366	M	22-25	2	1
710	M	22-25	1	2
851	M	22-25	4	5
639	M	22-25	4	5
293	M	22-25	3	3
977	M	18-21	4	5

Experiment data by magnification technique

Gaze

Subject	SUS1	SUS2	SUS3	SUS4	SUS5	SUS6	SUS7	SUS8	SUS9	SUS10	Score	Task time	Mag time	Teleports	Distance	Item misses	Order	Focus	Shape/size	Location
613	4	2	5	3	6	3	6	1	5	2	75	617.0708	389.1202	35	442.955	63	1	5	6	6
445	5	3	6	2	5	3	6	4	5	2	71.7	1296.819	942.11	82	666.838	40	1	4	3	4
378	7	1	7	1	6	1	7	4	6	1	91.7	468.5083	419.0495	48	220.793	17	2	7	6	6
584	2	3	6	1	4	4	6	3	7	2	70	289.0132	232.8745	17	147.86	12	3	2	2	2
850	2	4	5	2	3	4	3	6	4	3	46.7	947.8473	805.1549	87	675.814	27	2	3	3	4
916	3	5	5	1	5	4	5	4	6	1	65	306.229	286.3327	25	187.693	17	3	6	5	4
669	6	1	7	1	6	2	6	2	6	2	88.3	623.6785	449.7223	58	402.318	35	1	6	4	3
510	5	3	5	1	5	3	6	3	6	3	73.3	1029.057	661.6931	58	612.966	73	1	5	5	4
762	2	1	7	1	7	1	7	1	7	1	91.7	461.1494	411.5072	40	266.722	19	2	7	6	5
224	1	6	4	2	4	6	5	4	4	2	46.7	578.8728	475.3153	73	540.66	15	3	4	4	2
521	3	1	1	1	3	6	7	6	2	2	50	291.1975	260.1859	31	319.44	15	2	6	1	1
291	6	1	7	1	6	5	7	1	7	1	90	885.6815	793.1909	57	432.092	39	3	7	7	7
366	4	2	5	2	5	1	7	2	6	1	81.7	896.9771	672.5638	48	501.994	54	1	6	4	4
710	4	2	4	2	5	2	6	5	4	3	65	921.0404	736.5908	54	488.499	33	1	5	6	6
851	3	3	3	5	3	5	3	4	3	3	41.7	540.8738	473.8369	43	319.917	25	2	4	3	3
639	6	1	6	1	6	2	5	1	6	2	86.7	245.3138	225.4598	22	248.148	18	3	6	6	5
293	4	2	6	2	6	2	6	2	6	1	81.7	445.7046	399.3685	41	316.12	21	2	5	4	4
977	5	3	5	1	5	4	4	3	6	1	71.7	465.8221	440.6935	51	287.965	28	3	5	6	4

Harder w/o zoom	Zoom factor	Zoom speed	Sickness	Ranking	Finds w/ MagRect	Raised hands time	Item-finding distance
5	2	2	1	2	3	522.7365	1.693844
6	3	3	1	3	4	441.9958	1.025952
7	2	2	1	1	5	440.127279	2.380598
5	3	3	4	1	5	70.33446	1.324085
2	3	3	1	3	3	273.69735	0.877189
6	3	3	1	3	5	256.2037	3.658267
2	2	3	1	1	3	231.11698	1.74624
7	2	2	1	1	4	307.3533	1.013033
7	2	2	2	3	5	411.503711	1.23106
4	1	3	3	3	5	302.6371	1.86456
1	3	3	1	3	5	170.012069	1.233395
7	2	2	2	1	5	777.26995	1.547589
7	2	3	1	2	4	803.3025	3.256748
7	2	2	1	3	4	353.7752	1.768919
3	3	2	1	2	4	169.323812	1.262675
6	2	2	1	1	4	82.6876	1.796908
4	2	2	1	3	5	306.88614	5.367459
4	3	3	2	2	5	405.52394	2.796729

Combined

Subject	SUS1	SUS2	SUS3	SUS4	SUS5	SUS6	SUS7	SUS8	SUS9	SUS10	Score	Task time	Mag time	Teleports	Distance	Item misses	Order	Focus	Shape/size	Location
613	5	3	4	2	5	3	5	3	4	1	68.3	482.253	380.2654	36	376.023	24	2	4	5	5
445	5	2	6	2	5	2	6	2	6	2	80	400.6801	351.7368	27	212.59	9	3	5	5	4
378	7	2	6	3	6	2	7	4	5	1	81.7	862.8753	728.8542	88	515.451	35	1	7	5	4
584	2	2	6	1	4	1	7	2	7	1	81.7	779.9322	610.5558	26	339.351	27	1	5	2	3
850	5	4	4	2	5	3	6	3	5	3	66.7	235.1663	224.4624	27	167.347	19	3	5	5	5
916	3	4	6	2	5	4	5	5	5	3	60	808.4764	718.6566	64	474.569	42	2	6	4	4
669	4	3	5	2	5	2	3	3	4	3	63.3	636.4603	529.4183	65	419.853	38	2	4	2	2
510	4	5	5	1	5	3	6	5	6	3	65	535.4777	494.4637	49	365.082	29	3	4	3	4
762	2	1	6	1	7	1	7	1	6	1	88.3	754.9878	551.8459	62	469.28	35	1	6	6	4
224	3	2	6	2	6	2	6	4	7	2	76.7	926.9343	613.1859	62	534.888	28	1	5	6	5
521	4	2	4	1	2	1	5	4	6	1	70	392.0916	353.4343	53	340.282	31	3	6	1	1
291	6	1	7	2	6	1	7	1	7	1	95	752.8933	621.6418	35	352.648	40	2	7	7	7
366	6	1	6	1	6	1	7	1	6	1	93.3	363.8818	293.249	31	351.3	15	2	6	5	6
710	5	2	6	2	5	2	5	5	5	2	71.7	376.6244	332.5342	30	305.038	14	3	6	6	6
851	6	2	6	2	6	2	5	1	6	1	85	791.9917	639.959	52	444.208	37	1	6	6	4
639	6	2	5	1	6	2	5	2	6	1	83.3	387.526	278.1769	34	406.576	18	1	6	5	5
293	4	1	6	1	6	2	7	2	6	2	85	431.2907	379.4075	49	388.293	19	3	7	5	5
977	4	5	6	1	6	3	3	3	5	2	66.7	760.4149	607.0182	64	369.677	29	2	4	6	3

Harder w/o zoom	Zoom factor	Zoom speed	Sickness	Ranking	Finds w/ MagRect	Raised hands time	Item-finding distance
5	2	2	1	1	5	291.630562	1.373184
7	2	2	1	1	5	186.231453	2.181944
7	3	2	1	2	4	799.7178	1.519499
7	2	2	2	3	4	335.0334	1.481295
4	2	1	1	2	1	84.063894	0.840662
4	3	3	1	1	5	587.81384	2.684975
3	2	2	1	2	5	215.86751	1.363525
3	3	3	1	3	5	191.123937	1.443478
6	2	2	2	2	4	672.5938	1.532914
6	2	2	1	1	4	620.7735	3.136849
1	3	2	1	2	5	195.987054	1.288242
7	2	1	2	2	5	635.19875	1.299017
7	2	2	1	1	5	300.985877	1.450283
4	2	2	1	2	5	104.63961	1.287047
3	2	2	1	1	3	291.89261	1.100216
6	2	2	1	2	4	268.8042	1.476578
4	2	2	1	1	5	283.10742	2.388919
4	3	2	2	1	5	616.1309	3.369582

None

Subject	SUS1	SUS2	SUS3	SUS4	SUS5	SUS6	SUS7	SUS8	SUS9	SUS10	Score	Task time	Mag time	Teleports	Distance	Item misses	Order	Focus	Shape/size	Location
613	3	3	3	4	3	4	4	5	3	2	46.7	466.9849	408.2154	35	302.69	31	3	2	4	5
445	5	3	6	3	4	3	6	3	5	3	68.3	281.5503	196.4713	25	152.533	10	2	3	5	5
378	5	1	7	1	6	2	7	4	6	1	86.7	556.235	524.033	72	294.877	26	3	4	6	6
584	2	3	6	1	5	2	7	2	7	1	80	472.7592	383.0903	22	301.576	26	2	3	3	3
850	6	2	6	1	6	2	7	2	7	2	88.3	548.3857	432.7283	46	323.471	28	1	5	6	7
916	4	6	4	1	4	3	4	6	5	3	53.3	723.8165	523.4225	47	420.99	37	1	5	4	4
669	2	3	3	2	5	1	6	5	4	2	61.7	446.965	412.4377	47	376.262	24	3	4	1	1
510	4	3	6	1	5	3	5	3	5	2	71.7	604.6741	507.422	56	371.286	36	2	3	5	5
762	2	1	4	1	7	1	7	3	7	1	83.3	307.5265	291.5768	36	213.115	15	3	3	1	1
224	3	4	4	1	5	6	6	4	4	1	60	299.3196	264.8898	28	252.152	19	2	2	3	3
521	4	3	5	2	6	1	6	3	5	4	71.7	483.2831	362.8912	51	312.041	31	1	4	7	7
291	7	1	7	1	7	1	7	2	7	1	98.3	1090.106	740.8189	51	396.919	67	1	5	7	7
366	5	1	7	1	6	1	7	2	7	1	93.3	308.9119	279.2843	28	225.493	26	3	6	3	4
710	5	2	5	3	6	2	6	6	5	2	70	373.715	318.4175	21	215.561	16	2	6	6	6
851	2	4	2	3	4	5	5	5	4	2	46.7	329.9204	286.2734	31	252.957	19	3	5	2	3
639	3	3	4	2	6	4	4	3	5	3	61.7	522.8983	503.9964	58	462.35	20	2	3	4	3
293	4	2	6	2	6	2	6	2	5	1	80	527.1633	437.5148	46	301.953	46	1	5	4	5
977	6	2	7	1	6	2	6	1	7	1	91.7	949.7912	747.0691	69	507.49	31	1	5	6	6

Harder w/o zoom	Zoom factor	Zoom speed	Sickness	Ranking	Finds w/ MagRect	Raised hands time	Item-finding distance
0	0	0	1	3	5	283.242305	1.546892
0	0	0	1	2	4	123.884559	0.951278
0	0	0	1	3	5	526.548293	1.281957
0	0	0	4	2	5	157.886904	1.589839
0	0	0	1	1	2	243.2337	0.89648
0	0	0	1	2	4	519.193	1.690902
0	0	0	1	3	5	198.684731	1.727734
0	0	0	1	2	5	196.230424	0.73582
0	0	0	2	1	5	217.514223	1.211213
0	0	0	1	2	5	186.99366	1.162663
0	0	0	2	1	4	303.0649	1.071385
0	0	0	2	3	4	662.8251	1.166879
0	0	0	1	3	5	170.837674	1.252971
0	0	0	1	1	5	121.619126	1.802291
0	0	0	1	3	5	102.86251	1.873542
0	0	0	1	3	5	380.92894	1.323349
0	0	0	1	2	4	363.0228	1.240294
0	0	0	2	3	3	877.9933	1.43428

Longform responses

Subject 613

Gaze: "I mainly used the magnifying glass for transporting as opposed to magnifying."

Combined: "- Sometimes focussing using the magniftying glass was slightly delayed. - It was sometimes difficult to identify the correct position I wanted to transport to."

None: "I did not find the magnifying glass suitable for zooming or focusing, as it was difficult to correctly [mark?] the site to move to. This was based more on luck."

Preferred: Combined

Why: "I felt that this technique was very intuitive and allowed for the greatest amount of usefulness. It felt structured properly, and I was able to act on the surroundings easily."

Suggestions: "For long distances, sometimes focussing took a while. As moving through zooming was the most common act I did, this aspect might be good to improve."

Subject 445

Gaze: "I thought the magnifying glass a bit disturbing at times -> could not see the objects properly, as too big -> the movements were too quick."

Combined: "Definitely the better option out of the 3!"

None: "I believe the glass was better distance-wise, closer would be better, but not as close as the fast one."

Preferred: Combined

Why: "Because it gave me the option of seeing the image how I wanted to see it -> I could see a bigger chunk of objects or the one I wanted to -> more efficient"

Suggestions: "Maybe make it more steady as it moves easily with the movement of the hands. Sometimes it does not reach the wanted effect with the hands movement."

Subject 378

Gaze: "The second time was much easier."

None: "Without zooming in it was much harder to be transported to the right place."

Preferred: Gaze

Why: "I felt the most comfortable using it. The zoom was very useful in finding the right spot. And I believe that it is much more easy to use only one magnification (like in the second technique)."

Suggestions: "Maybe if you wanted to change magnification it could be done in a different way but not moving hands, so the view is more still."

Subject 584

Combined: "The resolutions can be better. The longer I used the headset the more dizziness I felt."

None: "The visuals became very blur by the end. Probably relates to the duration of the usage."

Preferred: Combined

Why: "It magnifies the object & I can modify it."

Subject 850

Combined: "I prefer the lack of zoom. However, the magnification helped me see objects from a distance that I was trying to find."

None: "The glass was an intuitive method of movement."

Preferred: None

Why: "I had more control. I found it easier to navigate when I had more control."

Suggestions: "It would help if the height was larger. I could alter the width with my hands, but the height was smaller than I wanted, and I could not change this."

Subject 916

None: "While the option to 'teleport' was useful, I found the glass a bit annoying, & sometimes I forgot it was there. I would prefer to use something else that has the same fuction (e.g. ability to point somewhere with the controller, rather than doing it via the glass.)

Preferred: Gaze

Why: "It felt more natural and intuitive."

Suggestions: "I think the zoom happens too fast and it's a bit too much of a zoom. I found that useful once, but most of the time it was a bit annoying and I felt like I wanted to 'put it away'. I'd suggest to make the zoom a bit more progressive & somehow make me feel like I'm in more control of the zoom level."

Subject 669

Gaze: "The magnifying glass made teleportation easier."

None: "The accuracy of the eye tracking made it hard to teleport where I wanted."

Preferred: Gaze

Why: "The glass looked where I wanted it to and teleportation was easy."

Suggestions: "I am not sure if I consciously used the eye tracking, but teleporting was easy even with far distances."

Subject 510

Preferred: Gaze

Why: "Sufficient magnification to give an overall pleasure [sic] user experience."

Suggestions: "Explore into 3-dimension magnification / teleportation to identify objects at higher shelves - Calibrate speed with default settings rather than user-dependent."

Subject 762

Preferred: None

Why: "Acurately showed me the region of the image I was interested in."

Suggestions: "Keep the image more stable. At times the image jumped around when I was not moving or looking around very much."

Subject 224

Preferred: Combined

Why: "As I had full control over it."

Suggestions: N/A

Subject 521

None: "Sometimes it was hard to have the circle move to the place I wanted it to be by moving my eyes. Sometimes only half of the circle was visible -> this was confusing, not sure if bug or feature."

Gaze: "I would have liked to be able to control the amount of zoom. No nausea compared to the circle on the ground. I found that using the glass I teleported too close to objects."

Combined: "The usefulness of the glass was little to none, because:

1) I could see stuff anyway

2) If it's far away at a small angle, I can't see it anyway even with the glass. It did, however, focus better on teleporting me where I wanted to than the previous ones - this might be because I learned to use it or because the headset was well calibrated and was not slipping."

Preferred: None

Why: "Allowed me to move fast, which was all I needed."

Suggestions: "Give button control to zoom function of glass. For glass, I would have liked to be able to see from top -> e.g. see into a bin from far away."

Subject 291

None: "Zone map."

Combined: "Combine similar zones."

Gaze: "Zoom glass feels better without hand distance operations."

Preferred: Gaze

Why: " 1) short-sighted (~400 deg.), best one facilitates operations

2) easier to operare than 2nd"

Suggestions: N/A

Subject 366

Preferred: Combined

Why: "Because I could control the degree of magnification."

Suggestions: "Use of buttons on the hand remotes."

Subject 710

Preferred: None

Why: "More straightforward + easier on the eyes."

Suggestions: "No suggestions to add."

Subject 851

Combined: "One time the magnifying glass was angled weirdly - possibly to do with my hands not being level but was a very minor inconvenience."

Gaze: "Magnifying glass in this round was less intuitive to use. Usage improved as I was able to use it more and more but still did not reach the proficiency I wanted like with round 1. Perhaps doing the tutorial in round 1 may have had an impact as it allowed me to get familiar with the magnifying glass as opposed to round 2."

None: "Glass in round 3 felt more intuitive compared to round 2 but less than round 1. However, teleporting to far away distances was the hardest in this round because a slight movement of the eyes results in a large distance in the virtual world space, making it harder to teleport long distances. Also, without the zoom, a slight offset of the eye tracking marker makes it harder to place the virtual marker to where you want to go."

Preferred: Combined

Why: "Most intuitive to use, did not feel like I was learning how to use it for long. Easy to designate where I wanted to move to. (Note: not sure if tutorial had an impact on this.)

Suggestions: "No major improvements come to mind."

Subject 639

None: "I felt the lack of the zoom made it difficult look at where I wanted to teleport."

Preferred: Gaze

Why: "It felt natural to just gaze at the object you wanted zoomed in."

Subject 293

Preferred: Combined

Why: "Customisability"

Suggestions: "Not really"

Subject 977

Preferred: Combined

Why: "Didn't zoom until needed. Other two were too zoomed in."

Suggestions: "Using the joystick when using glass to zoom in/out."

Appendix D

Project plan and interim report

COMP0138 Project plan

Project title: Exploring gaze-assisted magnified interfaces in virtual reality

Student: Sondre Agledahl

Supervisor: Anthony Steed

Aim

To assess the extent to which allowing users in a virtual environment to magnify their sense of vision based on their eye gaze is a viable interaction that enhances task performance and/or immersion.

Objectives

1. Explore existing literature on gaze-based user interaction in VR
2. Develop several (no more than three) candidate implementations of magnified gaze-assisted vision in Unity for use on the HTC Vive Pro Eye head-mounted display
3. Personally evaluate the viability of each implementation, and optimise them for use by inexperienced users
4. Design and create Unity scenes for evaluation containing logical puzzles that require fine-grained close-up vision
5. Prepare user test questionnaires and data gathering methods
6. Conduct user experiments with test subjects of each implementation, where subjects can optionally use the new technique while solving a puzzle
7. Evaluate the effectiveness of the new techniques based on experiment data

Deliverables

1. An implementation of a novel gaze-assisted magnification technique for use in virtual environments, as well as its design specification
2. An evaluation of the new technique based on experiment results
3. A literature survey of existing work on gaze-assisted interaction in VR

Work plan

Project start to end of December:

Literature review. Iterate on different possible Unity implementations. Assess the viability of these implementations personally and within the Virtual Environments group at UCL and decide on a (or limited set of) preferred solutions for experimentation. Document all design decisions during development.

January to mid-February:

Design and conduct user studies to evaluate final implementation.

Mid-February to end of March:

Analyse results of user studies and write final report.

COMP0138 Interim report

Project title: Exploring gaze-assisted magnified interfaces in virtual reality

Student: Sondre Agledahl

Supervisor: Anthony Steed

Progress to date

- Explored existing literature on gaze-based user interfaces in VR
- Developed and iterated on many possible gaze-magnified UI designs
- Received feedback on initial implementations from researchers in the UCL VE group
- Settled on a pair of slightly different gaze-magnification solutions for testing
- Designed and implemented a puzzle game for test subjects to try out the solution in
- Created a logging system for recording behaviour of test subjects in the game

Remaining work

- Perform user studies, with subjects playing the game while given the option of using the novel UI design
- Parse the data gathered from experiments
- Analyse effectiveness of solution based on gathered data
- Write final report

Appendix E

Code listing

This section contains a listing of the source code behind the systems described throughout this report.

Due to space constraints, code files that are not essential to magnification, teleportation, item detection or data collection have been omitted (notably all code files responsible only for aesthetic effects).

With the exception of *logParser.py* (written in Python 3), all code is in C#.

MagnificationManager.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using Tobii.XR;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;
using UnityEngine.XR;
using Valve.VR;

public class MagnificationManager : MonoBehaviour
{
    public bool IsMagnifying { get { return _isActive; } }

    public enum MagnificationMode { NATURAL, GAZE, COMBINED, NONE }

    public MagnificationMode Mode { get { return _mode; } }

    [SerializeField]
    private MagnificationMode _mode;

    [SerializeField]
    private float _rectHeight = 0.2f;

    [SerializeField]
    private float _offsetFromHands = 0.1f;

    private Transform _magRect;

    private Camera _magCamera;

    private IMagnifier _magnifier;

    private Transform _player;

    private WorldGazeTracker _gazeTracker;

    private HandTeleporter _handTeleporter;

    private GazeTeleport _gazeTeleport;

    private Checklist _checklist;

    private Transform _leftHand;

    private Transform _rightHand;

    private LerpAlpha[] _rectFadeEffects;

    private bool _isActive = false;

    private float _standardFov;

    private float _handDistance;

    private Vector3 _planeNormal = Vector3.zero;

    private RaycastHit? _gazeRectIntersection;

    private float _totalTimeActive = 0f;

    private float _curTimeActive = 0f;

    private BufferedLogger _log = new BufferedLogger("MagRect");

    private void Awake()
    {
        _magRect =
GameObject.FindGameObjectWithTag("MagRect").transform;
        _player = Camera.main.transform;
        _magCamera = GetComponentInChildren<Camera>();
        _standardFov = _magCamera.fieldOfView;
        _rectFadeEffects =
_magRect.GetComponentsInChildren<LerpAlpha>();
        _gazeTeleport = FindObjectOfType<GazeTeleport>();
        _handTeleporter = FindObjectOfType<HandTeleporter>();
        _checklist = FindObjectOfType<Checklist>();
        _gazeTracker = FindObjectOfType<WorldGazeTracker>();
```

```
        AssignMagMode();
    }

    private void Start()
    {
        ToggleMagnification(false);
    }

    private void AssignMagMode()
    {
        switch (_mode)
        {
            case MagnificationMode.NATURAL:
                _magnifier = GetComponent<NaturalMagnifier>();
                break;
            case MagnificationMode.GAZE:
            case MagnificationMode.NONE:
                _magnifier = GetComponent<GazeMagnifier>();
                break;
            case MagnificationMode.COMBINED:
                _magnifier = GetComponent<CombinedMagnifier>();
                break;
        }
    }

    private bool AreHandsAlive()
    {
        return _leftHand && _rightHand;
    }

    private void FindGazeRectIntersection()
    {
        if (Physics.Raycast(_gazeTracker.LastGazeRay, out RaycastHit
hit, 10f))
        {
            if (hit.collider.transform == _magRect)
            {
                _gazeRectIntersection = hit;
            }
        }
        else
        {
            _gazeRectIntersection = null;
        }
    }

    private void ToggleMagnification(bool isEnabled)
    {
        foreach (LerpAlpha la in _rectFadeEffects)
        {
            la.Fade(isEnabled);
        }
        _isActive = isEnabled;
        if (isEnabled)
        {
            _curTimeActive += Time.deltaTime;
            _totalTimeActive += Time.deltaTime;
        }
        else
        {
            _curTimeActive = 0f;
        }
    }

    public void OnHandConnectionChange(SteamVR_Behaviour_Pose pose,
SteamVR_Input_Sources changedSource, bool isConnected)
    {
        if (changedSource == SteamVR_Input_Sources.LeftHand)
        {
            _leftHand = isConnected ? pose.transform : null;
        }
        else if (changedSource == SteamVR_Input_Sources.RightHand)
        {
            _rightHand = isConnected ? pose.transform : null;
        }

        pose.GetComponentInChildren<Renderer>().enabled = isConnected;
        if (!isConnected)
        {
            ToggleMagnification(false);
```

```
        }
    }

    private void Update()
    {
        if (AreHandsAlive())
        {
            UpdateRectDimensions();
            FindGazeRectIntersection();
            if (_gazeRectIntersection.HasValue && !_isActive &&
!_handTeleporter.IsArcActive && !_checklist.IsVisible && _handDistance
<= 0.7f)
            {
                ToggleMagnification(true);
            }
            else if (_isActive && (!_gazeRectIntersection.HasValue ||
_handTeleporter.IsArcActive || _checklist.IsVisible || _handDistance >
0.7f))
            {
                ToggleMagnification(false);
            }
        }

        UpdateCameraTransform();
        if (_isActive && !_gazeTeleport.IsTeleportPending)
        {
            float magnification;
            if (_mode == MagnificationMode.NONE)
            {

_magnifier.GetMagnification(_gazeRectIntersection.Value,
_planeNormal);
                magnification = 1f;
            }
            else
            {
                magnification =
_magnifier.GetMagnification(_gazeRectIntersection.Value,
_planeNormal);
            }
            _log.Append("magFactor", magnification);
            _magCamera.fieldOfView = _standardFov / magnification;
        }

        if (_isActive)
        {
            _log.Append("curActiveTime", _curTimeActive);
            _log.Append("activeTimeTotal", _totalTimeActive);
        }

        _log.Append("active", _isActive);
        _log.CommitLine();
    }

    private void UpdateRectDimensions()
    {
        _handDistance = Vector3.Distance(_leftHand.position,
_rightHand.position);
        float width = _handDistance - _offsetFromHands;

        _magRect.localScale = new Vector3(width, _rectHeight, 1f);
        _magCamera.aspect = width / _rectHeight;
        _magRect.position = (_leftHand.position + _rightHand.position)
/ 2f;

        Vector3 upDir = _leftHand.forward + _rightHand.forward;
        Vector3 rightDir = _rightHand.position - _leftHand.position;

        _planeNormal = Vector3.Cross(rightDir, upDir);
        _magRect.rotation = Quaternion.LookRotation(_planeNormal,
upDir);

        _log.Append("width", width);
        _log.Append("pos", _magRect.position);
        _log.Append("rot", _magRect.rotation.eulerAngles);
        _log.Append("headDist", Vector3.Distance(_player.position,
_magRect.position));
    }
```

```
    private void UpdateCameraTransform()
    {
        _magCamera.transform.position = _magRect.position;
        _magCamera.transform.rotation =
Quaternion.LookRotation(_player.forward);
    }

    private void OnValidate()
    {
        if (Application.isPlaying && _player != null)
        {
            AssignMagMode();
        }
    }

}
```

GazeMagnifier.cs

```
using System.Collections;
using System.Collections.Generic;
using Tobii.XR;
using UnityEngine;
using UnityEngine.UI;
using Valve.VR;

public class GazeMagnifier : MonoBehaviour, IMagnifier
{
    // World space pos. of gaze dot last frame
    public Vector3 LastGazePos { get; private set; }

    public float LastGazeDistance { get; private set; }

    // How much to weigh the most recently sampled gaze distance
    [SerializeField]
    private float _sampleAlpha = 0.05f;

    // Max. distance of gaze ray
    [SerializeField]
    private float _gazeRange = 500f;

    [SerializeField]
    private float _distMultiplier = 0.8f;

    // How long after looking away to wait before reset magnification
    [SerializeField]
    private float _idleResetTime = 1f;

    // How many frames (n) position of gaze dot is determined from
    [SerializeField]
    private int _numFramesToSample = 30;

    private Transform _player;

    private Transform _magRect;

    private float _timeAtLastSample;

    // How many frames to weigh average gaze distance from; defined in
awake as 3n
    private int _numInertialFramesToSample;

    // Sampled world space gaze pos. for last n frames (stored as ring
buffer)
    private Vector3[] _sampledPoints;

    // Index in sampled points of most recent sample (always mod n)
    private int _frameIndex = 0;

    // Sampled distances between player pos. and gaze pos. (stored as
ring buffer)
    private float[] _sampledDistances;

    // Index in sampled distances of most recent sample (mod 3n)
    private int _inertialFrameIndex = 0;

    private bool _isMagActive = false;

    private WorldGazeTracker _gazeTracker;

    private Transform _gazeDot;

    private Camera _magCamera;

    // Ignore teleport marker
    private int _layerMask = ~(1 << 13);

    // Avg. gaze distance calculated last frame
    private float _oldAverageDist;

    private bool _isTeleporting = false;

    private float _lastMagnification = 1f;

    private BufferedLogger _log = new BufferedLogger("GazeMag");

    private void OnEnable()
    {
```

```
        GazeTeleport.OnGazeTeleport += OnGazeTeleport;
        Teleporter.OnTeleportComplete += OnTeleportComplete;
    }

    private void OnDisable()
    {
        GazeTeleport.OnGazeTeleport -= OnGazeTeleport;
        Teleporter.OnTeleportComplete -= OnTeleportComplete;
    }

    private void Awake()
    {
        _magRect =
GameObject.FindGameObjectWithTag("MagRect").transform;
        _gazeTracker = FindObjectOfType<WorldGazeTracker>();
        _player = Camera.main.transform;

        _timeAtLastSample = Time.time;

        _gazeDot =
GameObject.FindGameObjectWithTag("GazeDot")?.transform;

        _magCamera = _magRect.GetComponentInChildren<Camera>();

        _numInertialFramesToSample = _numFramesToSample * 4;

        _sampledPoints = new Vector3[_numFramesToSample];
        _sampledDistances = new float[_numInertialFramesToSample];
    }

    private void Update()
    {
        if (Time.time - _timeAtLastSample > _idleResetTime &&
_isMagActive)
        {
            _isMagActive = false;
        }
    }

    private void ResetZoom(bool toWorldGaze)
    {
        _frameIndex = 0;
        _inertialFrameIndex = 0;
        LastGazePos = _player.position;
        _oldAverageDist = 0f;

        for (int i = 0; i < _numFramesToSample; i++)
        {
            _sampledPoints[i] = toWorldGaze ? _gazeTracker.GazePos :
_gazeDot.position;
        }

        Vector3 eyeBallsPos = TobiiXR.EyeTrackingData.GazeRay.Origin;
        float gazeDistEstimate = Vector3.Distance(eyeBallsPos,
_gazeTracker.GazePos) / 2f;
        for (int i = 0; i < _numInertialFramesToSample; i++)
        {
            _sampledDistances[i] = toWorldGaze ? gazeDistEstimate :
0f;
        }

        _oldAverageDist = toWorldGaze ? gazeDistEstimate : 0f;

        _log.Append("didReset", true);
    }

    private void OnGazeTeleport(Vector3 destination)
    {
        _isTeleporting = true;
        // Extra zoom effect
        //float dist = Vector3.Distance(_player.position, destination)
* 6f;
        //for (int j = 0; j < _numInertialFramesToSample; j++)
        //{
        //    _sampledDistances[_inertialFrameIndex] = dist;
        //    _inertialFrameIndex = (_inertialFrameIndex + 1) %
_numInertialFramesToSample;
        //}
    }
```

```
    private void OnTeleportComplete()
    {
        _isTeleporting = false;
        ResetZoom(false);
    }


    // Called every frame when magnification active
    public float GetMagnification(RaycastHit gazePoint, Vector3
planeNormal)
    {
        if (!_isMagActive)
        {
            _isMagActive = true;
            ResetZoom(true);
        }
        if (_isTeleporting)
        {
            _log.CommitLine();
            return _lastMagnification;
        }

        _log.Append("active", true);
        Vector3 gazeSceenPos = gazePoint.textureCoord;
        Ray magRay = _magCamera.ViewportPointToRay(gazeSceenPos);

        Vector3 hitPos;
        if (Physics.Raycast(magRay, out RaycastHit hit, _gazeRange,
_layerMask))
        {
            hitPos = hit.point;
            _log.Append("curFrameGazePos", hitPos);
        }
        else
        {
            Debug.Log("Looking outside range");
            hitPos = gazePoint.point + (magRay.direction *
_gazeRange);
            _log.Append("outsideRange", true);
        }

            _sampledPoints[_frameIndex] = hitPos;

            Vector3 dotPos = Vector3.zero;
            _frameIndex = (_frameIndex + 1) % _numFramesToSample;
            int i = _frameIndex;
            for (int s = 0; s < _numFramesToSample; s++)
            {
                dotPos += _sampledPoints[i];
                i = (i + 1) % _numFramesToSample;
            }
            dotPos /= _numFramesToSample;
            _log.Append("dotPos", dotPos);

            _gazeDot.position = dotPos - (0.1f * magRay.direction);
            _gazeDot.rotation = Quaternion.LookRotation(_player.forward,
_player.up);

            LastGazePos = dotPos;

            Vector3 eyeBallPos = TobiiXR.EyeTrackingData.GazeRay.Origin;
            LastGazeDistance = Vector3.Distance(eyeBallPos,
_gazeDot.position);

            float magnification = 1f +
(GetWeightedAverageDist(LastGazeDistance) * _distMultiplier);

            _timeAtLastSample = Time.time;
            _lastMagnification = magnification;

            _log.CommitLine();
            return magnification;
        }

        // Take weighted average of weighted average distances
        private float GetWeightedAverageDist(float currentDist)
        {
            Debug.Assert(_inertialFrameIndex >= 0 && _inertialFrameIndex <
_numInertialFramesToSample);
            _log.Append("curDotDist", currentDist);
```

```
        _sampledDistances[_inertialFrameIndex] = currentDist;
        _inertialFrameIndex = (_inertialFrameIndex + 1) %
_numInertialFramesToSample;

        float averageDist = 0f;
        int i = _inertialFrameIndex;

        for (int s = 0; s < _numInertialFramesToSample; s++)
        {
            averageDist += _sampledDistances[i];
            i = (i + 1) % _numInertialFramesToSample;
        }
        averageDist /= _numInertialFramesToSample;

        // exponential moving average
        averageDist = (averageDist * _sampleAlpha) + ((1 -
_sampleAlpha) * _oldAverageDist);
        _oldAverageDist = averageDist;

        _log.Append("avgDotDist", averageDist);
        return averageDist;
    }

}
```

NaturalMagnifier.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using Valve.VR;

public class NaturalMagnifier : MonoBehaviour, IMagnifier
{
    // Virtual image distance of mag. glass
    [SerializeField]
    private float _imageDistance = 1.4f;

    [SerializeField]
    private float _focalLength = 0.23f;

    private Transform _player;

    private Transform _magRect;

    private void Awake()
    {
        _player = Camera.main.transform;
        _magRect =
GameObject.FindGameObjectWithTag("MagRect").transform;
    }


    public float GetMagnification(RaycastHit gazePoint, Vector3
planeNormal)
    {
        float eyeDistance = Vector3.Distance(_player.position,
_magRect.transform.position);
        float realImageDistance = Mathf.Abs(eyeDistance -
_imageDistance);

        float magnification = (0.25f / eyeDistance) * (1 +
((realImageDistance - eyeDistance) / _focalLength));

        return magnification;
    }
}
```

CombinedMagnifier.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using Valve.VR;

public class CombinedMagnifier : MonoBehaviour, IMagnifier
{
    // How much weight [0, 1] to give gaze magnification (rest
provided by natural-mag)
    [SerializeField]
    private float _gazeWeight = 0.5f;

    private GazeMagnifier _gazeMag;

    private NaturalMagnifier _naturalMag;

    private void Awake()
    {
        _gazeMag = GetComponent<GazeMagnifier>();
        _naturalMag = GetComponent<NaturalMagnifier>();
    }

    public float GetMagnification(RaycastHit gazePoint, Vector3
planeNormal)
    {
        float naturalMag = _naturalMag.GetMagnification(gazePoint,
planeNormal);
        float gazeMag = _gazeMag.GetMagnification(gazePoint,
planeNormal);

        return (gazeMag * _gazeWeight) + ((1 - _gazeWeight) *
naturalMag);

    }
}
```

IMagnifier.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public interface IMagnifier
{
    float GetMagnification(RaycastHit gazePoint, Vector3 planeNormal);
}
```

GazeTeleport.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using Valve.VR;
using Valve.VR.InteractionSystem;

public class GazeTeleport : MonoBehaviour
{
    public delegate void GazeTeleportEvent(Vector3 destination);
    public static event GazeTeleportEvent OnGazeTeleport;

    public bool IsTeleportPending { get { return
_teleportCandidate.HasValue; } }

    [SerializeField]
    private float _activationTime = 1f;

    private GazeDotIndicator _dotImage;

    private MagnificationManager _magManager;

    private GazeMagnifier _gazeMag;

    private TeleportPoint _teleportMarker;

    private TeleportMarkerCollider _markerCollider;

    private Teleporter _teleporter;

    private Vector3? _teleportCandidate;

    private Transform _player;

    private Transform _playspace;

    private float _holdDownTime = 0f;

    private bool _isHoldingTrigger = false;

    private BufferedLogger _log = new BufferedLogger("GazeTeleport");

    private void Awake()
    {
        _teleporter = FindObjectOfType<Teleporter>();
        _gazeMag = FindObjectOfType<GazeMagnifier>();
        _magManager = FindObjectOfType<MagnificationManager>();
        _teleportMarker = FindObjectOfType<TeleportPoint>();
        _markerCollider = FindObjectOfType<TeleportMarkerCollider>();
        _dotImage = GetComponent<GazeDotIndicator>();

        _player = Camera.main.transform;
        _playspace = _player.parent;
    }

    private void Update()
    {
        if (!_magManager.IsMagnifying)
        {
            return;
        }
        // Set + adjust position once per frame
        SetMarkerPosition();
        // If marker still collides after adjustment, target cannot be
teleported to
        bool isTargetValid = IsTeleportTargetValid();
        _dotImage.SetValid(isTargetValid);
        _log.Append("targetValid", isTargetValid);

        SteamVR_Action_Boolean_Source triggerDown =
SteamVR_Actions.default_GrabPinch[SteamVR_Input_Sources.RightHand];
        if (triggerDown.stateDown && !_teleporter.IsTeleporting)
        {
            if (isTargetValid)
            {
                _isHoldingTrigger = true;
                _holdDownTime = 0f;
```

```
            SetTeleportTarget();
            _teleportMarker.SetAlpha(1f, 1f);
        }
        else
        {
            _teleportCandidate = null;
            _teleportMarker.SetAlpha(0f, 0f);
            _log.CommitLine();
            return;
        }
    }
    if (triggerDown.stateUp)
    {
        _isHoldingTrigger = false;
        _teleportCandidate = null;
        _teleportMarker.SetAlpha(0f, 0f);
    }
    if (_isHoldingTrigger)
    {
        _holdDownTime += Time.deltaTime;
        _dotImage.SetProgress(_holdDownTime / _activationTime);
        if (_holdDownTime >= _activationTime)
        {
            _isHoldingTrigger = false;
            _log.Append("isTeleporting", true);

            // Start teleport
            _teleporter.Teleport(_teleportCandidate.Value);
            OnGazeTeleport(_teleportCandidate.Value);
            _teleportCandidate = null;
        }
    }
    else
    {
        _teleportMarker.SetAlpha(0f, 0f);
        _dotImage.SetProgress(1f);
    }
    _log.CommitLine();
}
```

```
    private void SetMarkerPosition()
    {
        Vector3 target = _gazeMag.LastGazePos;
        target.y = 0f;
        _teleportMarker.transform.position = target;

        if (_markerCollider.HasCollided())
        {
            _teleportMarker.transform.position =
_markerCollider.GetAdjustedPosition();
        }
    }

    private bool IsTeleportTargetValid()
    {
        Vector3 target = _gazeMag.LastGazePos;
        if (target.y > _player.position.y * 2f ||
_markerCollider.HasCollided())
        {
            return false;
        }
        return true;
    }

    private void SetTeleportTarget()
    {
        Vector3 target = _gazeMag.LastGazePos;
        _log.Append("origTarget", target);
        target.y = 0f;
        _teleportMarker.transform.position = target;

        if (_markerCollider.HasCollided())
        {
            target = _markerCollider.GetAdjustedPosition();
        }
        _teleportMarker.transform.position = target;
        _log.Append("shiftedTarget", target);
        _teleportCandidate = target;
    }
}
```

TeleportMarkerCollider.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class TeleportMarkerCollider : MonoBehaviour
{
    // Ignore floor, teleport marker and mag. rect
    private int _layerMask = ~((1 << 12) | (1 << 13) | (1 << 9));

    private CapsuleCollider _myCollider;

    private void Awake()
    {
        _myCollider = GetComponent<CapsuleCollider>();
    }

    public bool HasCollided()
    {
        Vector3 capsuleTop = transform.position + (transform.up *
_myCollider.height);
        return Physics.CheckCapsule(transform.position, capsuleTop,
_myCollider.radius, _layerMask);
    }

    public Vector3 GetAdjustedPosition()
    {
        Vector3 capsuleTop = transform.position + (transform.up *
_myCollider.height);

        Collider[] touchingColliders =
Physics.OverlapCapsule(transform.position, capsuleTop,
_myCollider.radius, _layerMask);

        float largestDist = 0f;
        Vector3 shiftDir = Vector3.zero;
        foreach (Collider col in touchingColliders)
        {
            if (!Physics.ComputePenetration(_myCollider,
transform.position, transform.rotation,
                col, col.transform.position, col.transform.rotation,
out Vector3 awayDir, out float distance))
            {
                continue;
            }
            // Don't want to shift downwards
            if (distance > largestDist && Vector3.Angle(-transform.up,
awayDir) > 45f)
            {
                largestDist = distance;
                shiftDir = awayDir;
            }

        }

        largestDist += _myCollider.radius;
        return transform.position + (largestDist * shiftDir);
    }

}
```

WorldGazeTracker.cs

```
using System.Collections;
using System.Collections.Generic;
using Tobii.XR;
using UnityEngine;

public class WorldGazeTracker : MonoBehaviour
{
    public Ray LastGazeRay { get; private set; }

    public Vector3 GazePos { get; private set; }

    [SerializeField]
    private int _numFramesToBuffer = 30;

    private int _gazeBufferIndex = 0;

    private Vector3[] _bufferedGazePositions;

    private void Awake()
    {
        _bufferedGazePositions = new Vector3[_numFramesToBuffer];
    }

    private void Update()
    {
        TobiiXR_GazeRay gazeRay = TobiiXR.EyeTrackingData.GazeRay;

        if (gazeRay.IsValid && Physics.Raycast(gazeRay.Origin,
gazeRay.Direction, out RaycastHit hit, 10f))
        {
            LastGazeRay = new Ray(gazeRay.Origin, gazeRay.Direction);

            _bufferedGazePositions[_gazeBufferIndex] = hit.point;
            int i = _gazeBufferIndex;
            Vector3 averageGazePos = Vector3.zero;
            for (int s = 0; s < _numFramesToBuffer; s++)
            {
                averageGazePos += _bufferedGazePositions[i];
                i = (i + 1) % _numFramesToBuffer;
            }
            averageGazePos /= _numFramesToBuffer;

            _gazeBufferIndex = (_gazeBufferIndex + 1) %
_numFramesToBuffer;
            GazePos = averageGazePos;
        }
    }

}
```

WorldConfig.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class WorldConfig : MonoBehaviour
{
    public Permutation WorldConfiguration { get { return _permutation;
} }

    public static WorldConfig Instance { get; private set; }

    // Item placements
    public enum Permutation { PERM_1, PERM_2, PERM_3 }

    [SerializeField]
    private Permutation _permutation;

    private void Awake()
    {
        if (Instance != null)
        {
            Debug.LogError("More than one WorldConfig!!!");
        }
        Instance = this;
    }

    public void SetPosition(Transform item, HiddenItem.Type type)
    {
        switch (_permutation)
        {
            case Permutation.PERM_1:
                switch (type)
                {
                    case HiddenItem.Type.BROWN_CHEESE:
                        item.position = new Vector3(1.767f,
0.4242516f, -21.158f);
                        item.rotation = Quaternion.Euler(2.316f,
536.898f, 0f);
                        break;
                    case HiddenItem.Type.BROKEN_BOTTLE:
                        item.position = new Vector3(3.935f, 0.807f, -
9.871f);
                        item.rotation = Quaternion.Euler(-3.476f, -
20.452f, 1.295f);
                        break;
                    case HiddenItem.Type.HEADLESS_FISH:
                        item.position = new Vector3(7.363f, 0.809f, -
40.293f);
                        item.rotation = Quaternion.Euler(0f, -82.833f,
0f);
                        break;
                    case HiddenItem.Type.MARGARINE:
                        item.position = new Vector3(3.2713f, 0.459f, -
35.316f);
                        item.rotation = Quaternion.Euler(0f, 0f, 0f);
                        break;
                    case HiddenItem.Type.PINEAPPLE_PIZZA:
                        item.position = new Vector3(-0.498f, 0.538f, -
27.656f);
                        item.rotation = Quaternion.Euler(0f, 110.59f,
0f);
                        break;
                }
                break;
            case Permutation.PERM_2:
                switch (type)
                {
                    case HiddenItem.Type.PINEAPPLE_PIZZA:
                        item.position = new Vector3(-8.399718f, 0.54f,
-44.4046f);
                        item.rotation = Quaternion.Euler(0f, 266.413f,
0f);
                        break;
                    case HiddenItem.Type.HEADLESS_FISH:
                        item.position = new Vector3(6.901f, 0.409f, -
36.793f);
                        item.rotation = Quaternion.Euler(0f, -0.108f,
0f);
                        break;
```

```
                    case HiddenItem.Type.BROKEN_BOTTLE:
                        item.position = new Vector3(-5.976f, 1.19f, -
26.372f);
                        item.rotation = Quaternion.Euler(23.433f, -
220.234f, -16.476f);
                        break;
                    case HiddenItem.Type.BROWN_CHEESE:
                        item.position = new Vector3(7.058f, 0.797f, -
15.881f);
                        item.rotation = Quaternion.Euler(0f, 543.318f,
0f);
                        break;
                    case HiddenItem.Type.MARGARINE:
                        item.position = new Vector3(1.587f, 0.275f, -
37.903f);
                        item.rotation = Quaternion.Euler(0f, 184.748f,
0f);
                        break;
                }
                break;
            case Permutation.PERM_3:
                switch (type)
                {
                    case HiddenItem.Type.PINEAPPLE_PIZZA:
                        item.position = new Vector3(-4.396f, 1.074f, -
8.886f);
                        item.rotation = Quaternion.Euler(-180f,
308.005f, 180f);
                        break;
                    case HiddenItem.Type.BROWN_CHEESE:
                        item.position = new Vector3(6.732431f,
0.4201571f, -32.80882f);
                        item.rotation = Quaternion.Euler(-1.83f,
543.318f, 0f);
                        break;
                    case HiddenItem.Type.HEADLESS_FISH:
                        item.position = new Vector3(8.204f, 0.409f, -
16.535f);
                        item.rotation = Quaternion.Euler(0f, 180f,
0f);
                        break;
                    case HiddenItem.Type.MARGARINE:
                        item.position = new Vector3(1.618f, 0.282f, -
19.763f);
                        item.rotation = Quaternion.Euler(0f, 184.748f,
0f);
                        break;
                    case HiddenItem.Type.BROKEN_BOTTLE:
                        item.position = new Vector3(-1.489f, 0.157f, -
31.217f);
                        item.rotation = Quaternion.Euler(30.855f, -
84.075f, 15.287f);
                        break;
                }
                break;
        }

    }

    private void OnValidate()
    {
        if (Application.isPlaying)
        {
            return;
        }

        HiddenItem[] items = FindObjectsOfType<HiddenItem>();
        foreach (HiddenItem item in items)
        {
            SetPosition(item.transform, item.ItemType);
        }

    }


}
```

HiddenItem.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class HiddenItem : MonoBehaviour
{
    public delegate void ItemEvent(Type type);
    public static event ItemEvent OnItemFound;

    public enum Type { BROWN_CHEESE, HEADLESS_FISH, BROKEN_BOTTLE,
PINEAPPLE_PIZZA, MARGARINE }

    public bool Found { get { return _hasBeenDiscovered; } }

    public Type ItemType { get { return _type; } }

    [SerializeField]
    private Type _type;

    private LerpAlpha _lerpAlpha;

    private ParticleSystem _particles;

    private bool _hasBeenDiscovered = false;

    private void Awake()
    {
        _lerpAlpha = GetComponent<LerpAlpha>();
        _particles = GetComponentInChildren<ParticleSystem>();
    }

    private void Start()
    {
        WorldConfig.Instance.SetPosition(transform, _type);
    }

    public void Discover()
    {
        if (_hasBeenDiscovered)
        {
            return;
        }
        // TODO: Sound effect
        _hasBeenDiscovered = true;
        OnItemFound(_type);
        StartCoroutine(FadeEffect());
    }

    private IEnumerator FadeEffect()
    {
        _lerpAlpha.FadeWithEmission();
        yield return null;
        Destroy(gameObject, 4f);
        _particles?.Play();
    }


}
```

GazeItemDetector.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class GazeItemDetector : MonoBehaviour
{
    [SerializeField]
    private float _detectionTime = 2f;

    private MagnificationManager _magManager;

    private WorldGazeTracker _worldGaze;

    private GazeMagnifier _gazeMagnifier;

    private HiddenItem[] _allItems;

    private HiddenItem _gazedAtItem;

    private Camera _playerCam;

    private Camera _magRectCam;

    private bool _isDiscoveryPending = false;

    private float _timeGazed = 0f;

    private BufferedLogger _log = new BufferedLogger("Items");

    private void Awake()
    {
        _magManager = FindObjectOfType<MagnificationManager>();
        _gazeMagnifier = FindObjectOfType<GazeMagnifier>();
        _worldGaze = FindObjectOfType<WorldGazeTracker>();
        _allItems = FindObjectsOfType<HiddenItem>();

        _playerCam = Camera.main;
        _magRectCam = _magManager.GetComponentInChildren<Camera>();
    }

    // Or null
    private HiddenItem GetItemAtPoint(Vector3 pos)
    {
        // Check all objects in 0.2m radius of gaze point
        Collider[] collidersSeen = Physics.OverlapSphere(pos, 0.2f);
        foreach (Collider col in collidersSeen)
        {
            HiddenItem item = col.GetComponent<HiddenItem>();
            if (item != null)
            {
                return item;
            }
        }

        return null;
    }

    // Or null
    private HiddenItem GetCurrentGazedAtItem()
    {
        // Check regular gaze ray
        HiddenItem item =  GetItemAtPoint(_worldGaze.GazePos);
        if (item == null && _magManager.IsMagnifying)
        {
            // Check gaze ray through MagRect
            item = GetItemAtPoint(_gazeMagnifier.LastGazePos);
        }
        return item;
    }

    private bool InViewOf(Vector3 point, Camera cam)
    {
        const int layerMask = ~((1 << 9) | (1 << 10) | (1 << 11) | (1
<< 13));

        // Check if inside camera's viewport
        Vector3 vp = cam.WorldToViewportPoint(point);
        if (vp.x < 1 && vp.x > 0 && vp.y < 1 && vp.y > 0 && vp.y < 1
&& vp.z > 0)
```

```
        {
            // Check that nothing is in the way
            if (!Physics.Linecast(cam.transform.position, point,
layerMask))
            {
                return true;
            }
        }
        return false;
    }

    private void CheckFov()
    {
        foreach (HiddenItem item in _allItems)
        {
            if (item.Found)
            {
                continue;
            }

            if (InViewOf(item.transform.position, _playerCam))
            {
                Debug.Log(item.ItemType + " visible to player");
                _log.Append(item.ItemType + "_inPlayerFov", true);
            }
            if (_magManager.IsMagnifying &&
InViewOf(item.transform.position, _magRectCam))
            {
                Debug.Log(item.ItemType + " visible through rect");
                _log.Append(item.ItemType + "_inRectFov", true);
            }
        }
    }


    private void Update()
    {
        CheckFov();
        HiddenItem item = GetCurrentGazedAtItem();
        if (_isDiscoveryPending)
        {
            // Player looked away -- cancel discovery
            if (item == null)
            {
                _isDiscoveryPending = false;
            }
            else
            {
                _timeGazed += Time.deltaTime;
                if (_timeGazed >= _detectionTime)
                {
                    _gazedAtItem.Discover();
                    _isDiscoveryPending = false;

                    _log.Append(item.ItemType + "_discovered", true);
                }
            }
        }
        if (item != null && !_isDiscoveryPending)
        {
            _gazedAtItem = item;
            _timeGazed = 0f;
            _isDiscoveryPending = true;

            _log.Append(item.ItemType + "_firstSpotted", true);
        }

        _log.CommitLine();
    }

}
```

BufferedLogger.cs

```
using System.Collections;
using System.Collections.Generic;
using System.IO;
using System.Text;
using UnityEngine;

public class BufferedLogger
{
    private readonly string _className;

    private StringBuilder _sb = new StringBuilder();

    private bool _isNewLine = true;

    public BufferedLogger(string className)
    {
        _className = className;
    }

    public void Append(string label, object data)
    {
        if (_isNewLine)
        {
            _sb.Clear();

_sb.Append('"').Append(_className).Append('"').Append(":{");
            _isNewLine = false;
        }
        _sb.Append('"').Append(label).Append('"').Append(':');

        if (data is Vector3 v3)
        {
            _sb.Append("[").Append(v3.x + ",").Append(v3.y +
",").Append(v3.z + "]");
        }
        else if (data is Vector2 v2)
        {
            _sb.Append("[").Append(v2.x +
",").Append(v2.y).Append("]");
        }
        else if (data is float || data is int)
        {
            _sb.Append(data);
        }
        else
        {
            _sb.Append('"').Append(data).Append('"');
        }
        _sb.Append(',');
    }

    public void CommitLine()
    {
        if (_isNewLine)
        {
            return;
        }

        // remove trailing comma
        _sb.Remove(_sb.Length - 1, 1);
        _sb.Append("}");

        FileLogger.Instance.AppendLine(_sb.ToString());
        _isNewLine = true;
    }


}
```

logParser.py

```
import sys
import json
import os
import numpy as np

glob_item_positions = [
    { # permutation 1
        "BROWN_CHEESE": (1.767, 0.4242516, -21.158),
        "BROKEN_BOTTLE": (3.935, 0.807, -9.871),
        "HEADLESS_FISH": (7.363, 0.809, -40.293),
        "MARGARINE": (3.2713, 0.459, -35.316),
        "PINEAPPLE_PIZZA": (-0.498, 0.538, -27.656)
    },
    { # permutation 2
        "PINEAPPLE_PIZZA": (-8.399718, 0.54, -44.4046),
        "HEADLESS_FISH": (6.901, 0.409, -36.793),
        "BROKEN_BOTTLE": (-5.976, 1.19, -26.372),
        "BROWN_CHEESE": (7.058, 0.797, -15.881),
        "MARGARINE": (1.587, 0.275, -37.903)
    },
    { # permutation 3
        "PINEAPPLE_PIZZA": (-4.396, 1.074, -8.886),
        "BROWN_CHEESE": (6.732431, 0.4201571, -32.80882),
        "HEADLESS_FISH": (8.204, 0.409, -16.535),
        "MARGARINE": (1.618, 0.282, -19.763),
        "BROKEN_BOTTLE": (-1.489, 0.157, -31.217)
    }
]

def get_max_frame(log_json):
    max_frame = 0
    while True:
        if not log_json.get("frame-" + str(max_frame + 1)):
            break
        max_frame += 1
    return max_frame

def parse_log(log_json, iteration):
    max_frame = get_max_frame(log_json)
    start_time = get_start_time(log_json)
    out_str = ""
    #out_str += "Highest frame is %d\n" % max_frame
    #print("Completion time is", get_completion_time(log_json,
max_frame, start_time))
    #out_str += "Mag time: %f\n" % get_mag_time(log_json, start_time,
max_frame)
    #out_str += "Number of teleports: %d\n" %
get_num_teleports(log_json, start_time)

    #out_str += "Distance covered: %f\n" %
get_distance_covered(log_json, start_time)
    #out_str += "Number of item misses: %d\n" %
get_missed_items(log_json, max_frame)
    #out_str += "Num. finds through MagRect: %d\n" %
get_mag_rect_finds(log_json)
    #out_str += "Time with raised hands: %f\n" %
get_hands_raised_time(log_json, start_time, max_frame)
    #out_str += "Mean item-finding distance: %f\n" %
get_mean_item_find_distance(log_json, iteration)

    return out_str

def get_distance_covered(log_json, start_time):
    dist = 0.0
    buffered_position = []
    last_buffered = []
    mod = 0
    for key in log_json:
        frame = log_json[key]
        if frame == 1 or frame["time"] <= start_time:
            continue
        pos = frame["Player"]["GlobalHeadPos"]
        pos[1] = 0.0 # ignore y-movement

        if len(buffered_position) > 0:
            buffered_position = np.divide(np.add(pos,
buffered_position), 2.0)
        else:
            buffered_position = pos
```

```
        mod += 1
        if mod % 90 == 0:
            if len(last_buffered) > 0:
                dist += np.linalg.norm(np.subtract(last_buffered,
buffered_position))
                last_buffered = buffered_position
                buffered_position = []
            else:
                last_buffered = buffered_position
    return dist

def get_start_time(log_json):
    # Check whether tutorial was done; if so start from there
    start_time = 0.0
    for key in log_json:
        frame = log_json[key]
        gm = (frame != 1) and frame.get("GameManager")
        if gm and gm.get("tutorialComplete") == "True":
            start_time = frame["time"]
            break
    return start_time

def get_completion_time(log_json, max_frame, start_time):
    # Find GameManager:gameOver (searching backwards)
    for i in range(max_frame, 0, -1):
        frame = log_json["frame-%d" % i]
        gm = frame.get("GameManager")
        game_over = gm and gm.get("gameOver")
        if game_over == "True":
            return frame["time"] - start_time
    print("ERROR - couldn't find completion time")

def get_hands_raised_time(log_json, start_time, max_frame):
    raised_time = 0.0
    last_lowered_timestamp = 0.0
    hands_raised = False

    # Need to go in chronological order
    for i in range(1, max_frame):
        frame = log_json["frame-%d" % i]
        controller = frame.get("Controller")
        if frame["time"] <= start_time or not controller:
            continue
        height = frame["Player"]["GlobalHeadPos"][1]

        l_pos = controller.get("LPos")
        r_pos = controller.get("RPos")

        if l_pos is None or r_pos is None:
            continue

        right_hand_height = r_pos[1]
        left_hand_height = l_pos[1]

        threshold = (2.0 / 3.0) * height
        if left_hand_height > threshold and right_hand_height >
threshold:
            hands_raised = True
        else:
            if hands_raised:
                hands_raised = False
                if last_lowered_timestamp > 0:
                    raised_time += frame["time"] -
last_lowered_timestamp
                last_lowered_timestamp = frame["time"]
    return raised_time

def get_mean_item_find_distance(log_json, iteration):
    mean_dist = 0.0

    for key in log_json:
        frame = log_json[key]
        items = (frame != 1) and frame.get("Items")
        if not items:
            continue

        for item_key in items:
            if "_discovered" in item_key and items[item_key] ==
"True":
```

```
                item_name = item_key[0:item_key.index("_discovered")]
                item_pos = glob_item_positions[iteration -
1][item_name]
                player_pos = frame["Player"]["GlobalHeadPos"]
                to_item = np.subtract(item_pos, player_pos)
                mean_dist += np.linalg.norm(to_item)
                break
    return mean_dist / 5.0

def get_mag_time(log_json, start_time, max_frame):
    prev_frame = None
    mag_time = 0.0
    # Need to go in chronological order
    for i in range(1, max_frame):
        frame = log_json["frame-%d" % i]
        mag_rect = frame.get("MagRect")
        active = mag_rect and mag_rect.get("active")
        if active == "True" and prev_frame:
            mag_time += frame["time"] - prev_frame["time"]
        prev_frame = frame
    return mag_time - start_time

def get_num_teleports(log_json, start_time):
    num_teleports = 0
    for key in log_json:
        frame = log_json[key]
        gt = (frame != 1) and frame.get("GazeTeleport")
        teleport = gt and gt.get("isTeleporting")
        if teleport == "True" and frame["time"] > start_time:
            num_teleports += 1
    return num_teleports

def get_missed_items(log_json, max_frame):
    num_misses = 0
    item_visible = False
    last_vis_item_key = None

    # Make sure we go in chronological order
    for i in range(1, max_frame):
        frame = log_json["frame-%d" % i]
        items = frame.get("Items")
        if items:
            # Check if item found this frame
            for item_key in items:
                if "_discovered" in item_key and items[item_key] ==
"True":
                    item_visible = False
                    break

            # If not, check whether item still visible
            if item_visible:
                if items.get(last_vis_item_key) != "True": # no longer
visible
                    num_misses += 1

                for item_key in items:
                    if "Fov" in item_key and items[item_key] == "True":
                        item_visible = True
                        last_vis_item_key = item_key
                        break
        else:
            if item_visible:
                num_misses += 1
            item_visible = False
    return num_misses

def get_iteration_no(filename):
    l_paren = filename.index('(') + 1
    r_paren = filename.index(')')

    iteration_no = int(filename[l_paren:r_paren])
    assert iteration_no > 0 and iteration_no < 4
    return iteration_no

# i.e. num. item finds through MagRect
def get_mag_rect_finds(log_json):
    num_finds = 0
    for key in log_json:
        frame = log_json[key]
        items = (frame != 1) and frame.get("Items")
```

```
        if items:
            for item_key in items:
                item_name = None
                if "_inRectFov" in item_key and items[item_key] ==
"True":
                    item_name =
item_key[0:item_key.index("_inRectFov")]
                    if items.get(item_name + "_discovered") == "True":
                        num_finds += 1
                    break
    return num_finds

def main():
    if "-r" in sys.argv:
        assert len(sys.argv) == 3

        with open("./logout.txt", 'w') as out_file:
            folder = sys.argv[2]
            for root, _, files in os.walk(folder):
                for file in files:
                    filename = root + '/' + file
                    print("Parsing", filename)
                    with open(filename) as log_file:
                        log_json = json.load(log_file)
                        out_file.write("\n" + filename + "\n")

                        iteration = get_iteration_no(filename)
                        out_file.write(parse_log(log_json, iteration))
    else:
        assert len(sys.argv) == 2

        with open(sys.argv[1]) as log_file:
            log_json = json.load(log_file)
            iteration = get_iteration_no(sys.argv[1])
            print(parse_log(log_json, iteration))
    print("DONE!")

if __name__ == "__main__":
    main()
```