EyeMRTK: A Toolkit for Developing Eye Gaze Interactive Applications in Virtual and Augmented Reality

Diako Mardanbegi
Lancaster University, UK
d.mardanbegi@lancaster.ac.uk

Thies Pfeiffer
CITEC, Bielefeld University, Germany
thies.pfeiffer@uni-bielefeld.de

ABSTRACT

For head mounted displays, like they are used in mixed reality applications, eye gaze seems to be a natural interaction modality. EyeMRTK provides building blocks for eye gaze interaction in virtual and augmented reality. Based on a hardware abstraction layer, it allows interaction researchers and developers to focus on their interaction concepts, while enabling them to evaluate their ideas on all supported systems. In addition to that, the toolkit provides a simulation layer for debugging purposes, which speeds up prototyping during development on the desktop.

CCS CONCEPTS

• **Human-centered computing → User interface toolkits**;

KEYWORDS

eye tracking, gaze interaction, virtual reality, Unity

ACM Reference format:
Diako Mardanbegi and Thies Pfeiffer. 2019. EyeMRTK: A Toolkit for Developing Eye Gaze Interactive Applications in Virtual and Augmented Reality. In *Proceedings of Communication by Gaze Interaction, Denver , CO, USA, June 25–28, 2019 (COGAIN @ ETRA'19),* 5 pages.
https://doi.org/10.1145/3317956.3318155

1 INTRODUCTION

The motivation for creating EyeMRTK was to provide the students and developers with the main elements needed for implementing eye gaze interactive applications in mixed reality (augmented and virtual reality). These main elements consist of: 1) the filtered version of the gaze ray in 3D as well as the rays defined by head direction and wand laser, 2) a set of confirmation methods that can be used for selection, 3) and finally, a component that can be assigned to any object to make them interactable.

Eye tracking and (eye-)gaze-based interaction is a long researched subject (e.g. see [Majaranta and Bulling 2014]) and has been recognized as a relevant interaction modality for augmented reality [Höllerer and Feiner 2004]. Since then, its applicability in this domain has been demonstrated by a range of applications [Lee et al. 2011; Nilsson et al. 2009; Park et al. 2008; Toyama et al. 2015]. However, all of these examples are using proprietary implementations of gaze-based interaction patterns. This is understandable, as in early research, almost all devices integrating eye tracking and mixed reality were proprietary scientific prototypes. This has changed since then, and today established eye tracking system builders offer to integrate eye tracking into consumer HMDs for immersive VR. One of the forerunners of this trend has been SMI [Sensomotoric Instruments 2018], yet others, such as Tobii [Tobii VR 2019] have quickly followed. In addition to that, start-up companies offer VR headsets with build-in eye tracking, such as FOVE [Inc. 2017] or Looxid Labs [LooxidLabs 2017]. In 2018, the first AR headset with built-in eye tracking has been shipped [Magic Leap, Inc. 2019].

A toolkit for gaze-based interaction in mixed reality should provide solutions for the basic interaction mechanisms:

- Selection
- Navigation
- Manipulation

Beyond that, there are system-level interaction techniques, some operating on a very technical level, such as foveated rendering [Guenter et al. 2012; Levoy and Whitaker 1990], in which the gaze orientation is used to optimize the pipeline for generating the digital images. Others are used to optimize aspects such as information clutter, e.g. by making visualizations responsive to gaze, realizing a gaze-over mechanism similar to mouse-over mechanisms known from HTML/JavaScript.

In the first iteration of the EyeMRTK toolkit, the focus lies on direct interaction mechanisms. The rationale behind this is that it is important to provide key advantages of gaze-based interaction to developers and users, so that employing the technology, which still requires efforts and financial resources, can be justified. More subtle uses of gaze-based interaction might have their niche, but may only come second once the general concept of gaze-based interaction has been accepted.

Beyond interaction techniques, the use of eye tracking as means for interaction comes along with some requirements and procedures, that have to be followed to maintain a high-level of usability. First of all, the devices typically have to be calibrated to the user. There are many aspects in which our visual systems differ and some are relevant either on the basic level of computer vision, the estimation of eye orientation or the interpretation of gaze direction. Especially users not familiar with gaze-based interaction need to be supported in setting up optimal conditions. Thus means for measuring and reporting the currently achieved accuracy and contrasting it to the optimal accuracy or monitoring for drift of the eye tracking

COGAIN @ ETRA'19, June 25–28, 2019, Denver , CO, USA

ACM ISBN 978-1-4503-6728-8/19/06…$15.00
https://doi.org/10.1145/3317956.3318155

gear are important, yet often neglected by developers. The toolkit aims to provide standard wizards and means to communicate these aspects to the users and guide them towards an optimal calibration in the future.

2 RELATED WORK

The design of the present toolkit for eye-gaze-based interaction is heavily inspired by the available VR toolkits. On the one hand, it provides a layer of abstraction over different eye tracking systems. On the other hand, it offers a modular approach of interaction modelling, which abstracts away from particular hardware specific implementations. In the discussion of related work, we want to focus on the interaction layer, to underline the diversity of concepts and approaches that have been suggested for gaze-based interaction.

2.1 Toolkits for Interaction in AR/VR

The toolkit targets Unity, which is as of today the most universal engine to realized mixed reality applications on almost all relevant platforms. Unity has introduced its own support for mixed reality projects (called XR support). However, there are several toolkits available that provide extended possibilities and examples. Those toolkits are either provided by system developers, such as the SteamVR SDK [ValveSoftware 2019] or the Oculus Integration Package [Oculus 2019], or they are grown by the VR/AR enthusiasts in community efforts, such like the Virtual Reality ToolKit [TheStonefox 2019]. We acknowledge that there are toolkits for other platforms and frameworks (e.g. [Takala 2014]), which we will leave out to remain focused. The main contribution of these SDKs is that they provide an abstraction layer for the available hardware devices (head mounted displays, controllers) based on components, such as abstract user models (representing head and hands) and cameras. Applications built on the basis of these components will in the best case support multiple VR platforms out of the box or can at least be easily ported to different platforms. Switching from one desktop VR setup, e.g. to HTC Vive, to another, such as Oculus Rift, is for example handled completely by the abstraction layer (e.g. OpenVR which is the basis of SteamVR) as long as no platform specific elements have been used, such as e.g. the thumbsticks on the Windows Mixed Reality Controllers. Switching from desktop setups, e.g. Oculus Rift, to a mobile setup, such as the Oculus Go, however, will require a retargeting of the project to the different binary platform (here e.g. from Windows/x86 to Android/arm), but the application model itself may stay untouched as long as the interaction design did not require two controllers or room-scale tracking, which is not available on the mobile platform.

2.2 Gaze-Based Interaction

Most gaze-based interaction methods have focused on selection in desktop scenarios. The application of eye-gaze in the desktop scenario has been described by Jacob [1990]. Many of the techniques that have been invented for the interaction with 2D user interfaces via gaze may be translated to mixed reality and spatial interfaces.

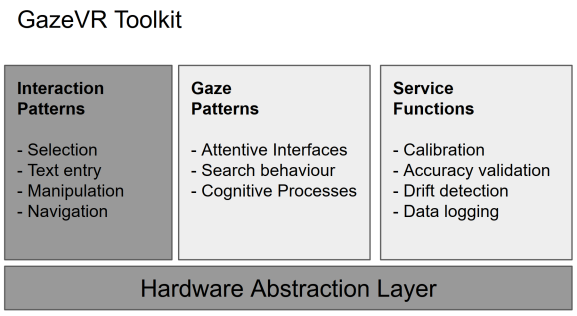

Figure 1: The basic architecture of the EyeMRTK. Focus of this paper are the HAL and the interaction patterns for selection.

2.3 Gaze-Based Interaction in AR/VR

Sibert and Jacob [2000] as well as Tanriverdi and Jacob [2000] demonstrated at about the same time that eye-gaze-based selection techniques could outperform other techniques, such as hand-based pointing or mouse-based interaction for selecting objects in virtual environments. This was replicated in several following studies with modified designs of the selection process (e.g. [Cournia et al. 2003], [Hülsmann et al. 2011]), however, depending on the combination of aiming and triggering mechanism, the performances differed in terms of time and accuracy. Since then, many alternative techniques for gaze-based interaction have been proposed (e.g. [Pfeuffer et al. 2017; Piumsomboon et al. 2017])

An area of human-computer interaction in which eye tracking has a prominent appearance is assistive computing and in particular eye-gaze-based typing (see e.g. [Hansen et al. 2004; Majaranta and Bates 2009]). Gaze-based typing for AR or VR has only been targeted quite recently (e.g. see [Blattgerste et al. 2018]) and will be integrated in EyeMRTK in the near future.

3 EYEMRTK

The main building blocks of the architecture of EyeMRTK are sketched in Figure 1. The toolkit will be incrementally extended according to the following roadmap:

(1) Hardware abstraction layer (realized, but not focus of this paper)
(2) Selection techniques (focus of this paper)
(3) Extended interaction techniques, such as text entry
(4) Gaze-pattern detection to classify user behavior (e.g. detecting reading or searching)
(5) Service functions, such as calibration procedures, drift detection and correction, accuracy analysis

The majority of the interaction techniques mentioned in the related work are based on a combination of three basic operations: 1) user aiming at a target, 2) user confirming the selection, and 3) system providing feedback during 1) and 2). In the following we describe how these three main components are implemented in the toolkit and later how they can be used.

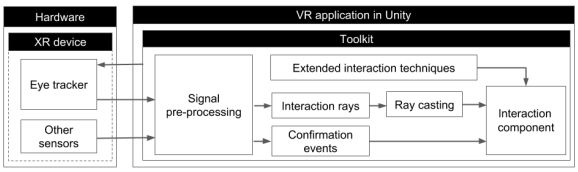

Figure 2: Data flow for gaze interaction in our toolkit.

3.1 Aiming / Pointing

Aiming in a spatial environment typically requires at least 2-3 degrees of freedom (e.g. rotation in a panoramic VR environment) and up to 6 degrees of freedom in room-scale VR. Typically, the aiming direction is represented as a ray, cast into the scene. The toolkit provides easy access to three main sources of rays in Unity: the laser beam from the wand controller, head direction ray, and three variants of gaze rays (left eye, right eye, cyclopean gaze ray) (Figure 3). Each ray can be set to be displayed in the Scene View to facilitate debugging. Both the raw and the filtered versions of the gaze rays are available to use and can be utilized for different purposes.

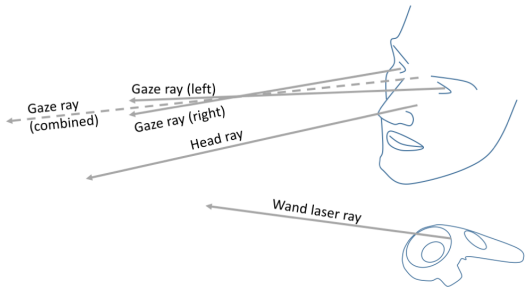

Figure 3: The toolkit provides an easy access to various sources of rays.

3.1.1 Gaze Ray. The main aim of this toolkit was to make the application development separate from the type of the eye tracker and having a common interface for various eye trackers. The current version of the toolkit supports Tobii [Tobii VR 2019], SMI [Sensomotoric Instruments 2018], and Pupil [Kassner et al. 2014] eye tracking modules made for VR. The interface controls the calibration evens and provides the gaze as a ray in 3D using the gaze data received from the tracker. It further processes the gaze data as follows.

3.1.2 Processing the Gaze Ray. The gaze data are processed using a smoothing filter that allows the user to smooth the jittery gaze data during fixations. This is done using filter that average the last N samples of the gaze data where the value of N can be set by the user. To prevent the filter from smoothing the saccadic eye movements and adding a delay after the saccades, the filter buffer is cleared after each saccade (similar to the method used by Kumar et al. [2008]). The saccade detection is based on the velocity where a saccade is identified as parts of the gaze signal where the velocity goes above a threshold (also defined by the user). The gaze angular velocity along each direction is calculated using a weighted moving average over five data samples (to suppress noise) applied on the gaze angular displacement. The magnitude of the angular gaze velocity will be taken as the gaze velocity used in the saccade detector.

3.1.3 Ray Casting. Each ray is provided by a class with a list containing the name of all intersected objects which will be used later to indicate whether each ray hits a given object. The Ray-enter and Ray-exit events are defined as the moment where each ray hits an object or leaves an object respectively which are monitored continuously for each object in relation to each ray. This is done inside the *EyeGazeInteractable* component which can be assigned to each individual object. We later further explain the use of this component.

3.1.4 Ray Pointer. The ray pointer is defined as a point along the ray which can be shown in the Game View (seen by the user). The point can be set to be either the hit point where the ray hits an object (cast trail) or a point always at a fixed distance along the ray (relative to origin of the ray). Showing the pointer at a fixed distance insures that the pointer size remains fixed in the view regardless of the object distance, yet it may introduce stereo conflicts.

3.2 Confirmation methods

Three main methods for confirmation that are provided by the toolkit are:

(1) Dwell time: The dwell duration can be set differently for each ray and for each object. Several algorithms have been suggested to improve the robustness and responsiveness of dwell time-based triggers (e.g. [Räihä 2015]). For example, the dwell timer may be set to ignore a few frames of gaze hit-loss to increase robustness. This prevents the dwell counter from being reset when gaze leaves the object for a few frames which may be due to jittery or inaccurate gaze data.
(2) Button press: A set of key status of the active controller used by the user are detected and they can be used for confirmation.
(3) Basic head gestures [Mardanbegi et al. 2012; Špakov and Majaranta 2012]: The head gestures are detected based on the head velocity along the main four directions providing a set of simple one-stroke head gestures: Head left, head right, head up, and head down. Head rotations that bring the head back to its natural orientation are discarded for 1 second after the first head rotation is made.

3.3 EyeGazeInteractable Component

In order to make the development of gaze interactive applications easier, EyeMRTK provides a *EyeGazeInteractable* component that can be assigned to any object in the scene. This component monitors the status of the object in relation to any of the rays introduced previously (e.g., GazeEnter event, GazeExit event, and fixation duration). It also monitors the confirmation events in relation to each object.

To make an object interactive, the EyeGazeInteractable component has to be assigned to the object. The pointing and confirmation status monitored by this component can then be accessed from another asset.

4 SIMULATION MODE

Another feature in the EyeMRTK is the simulation mode which permits the developers to implement their gaze-based interaction concepts without having their VR device or any eye tracker connected to their computer. The gaze point is emulated by moving the mouse in the screen on top of the Game window in Unity. The camera movements, which are controlled by head rotations in VR, are emulated using four arrow keys on the keyboard and the controller key press events are replaced by the left/right mouse clicks.

5 EXAMPLES

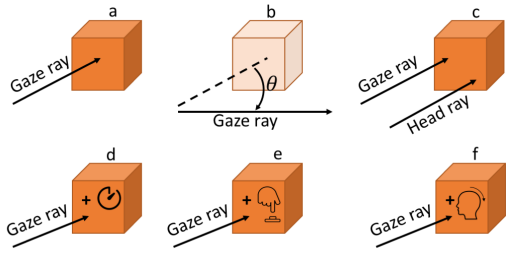

Figure 4: Six different examples demonstrating how different methods for pointing and selecting can be combined to interact with an object using the toolkit.

Various examples are included in the toolkit that show different combinations of the pointing and selecting methods for interaction with objects which we describe each as follows:

Point and Highlight: This example uses a piece of code that simply checks the RayEnter event from the EyeGazeInteractable component and highlights the object when the ray intersects with the object (Figure 4.a). The ray used for aiming in the example can be set to be any of the main three rays.

Visible in Periphery:An object could react differently depending how far the gaze ray is from the object (θ angle in Figure 4.b). This example shows an object that is only visible in the periphery and it disappears as soon as the angle θ goes smaller than a threshold. The angle θ between the ray and the object can be utilized differently either as a continuous parameter (e.g., gradually changing the transparency of the object) or a discreet parameter (e.g., hiding or showing the object).

Gaze and Head intersection: This example illustrates a simultaneous use of multiple rays similar to some of the methods proposed by [Zeleznik et al. 2005] where the pointing task is done by intersecting a hand-held pointing ray with the gaze or head ray. The object in this example gets selected as soon as multiple rays (e.g., gaze and head) intersect with the object (Figure 4.c).

Gaze and Dwell confirmation: This example shows a simple form of interaction that uses a dwell time method to confirm the selection when looking at a target. The object gets selected when the gaze ray fixates on the target for a certain amount of time (Figure 4.d).

Gaze and Button press: In this example, the confirmation is done by pressing the touchpad of the controller while the gaze ray intersects with the object (Figure 4.e).

Gaze and Head gesture: In this example the pointing can be done by various types of rays and the confirmation is done by a head gesture. Head gestures can also be used when the pointing is done by the head ray even though the head ray may go outside the object during the head movements. This is possible because the gesture confirmation takes the pointing direction when the gesture has not started yet.

6 CONCLUSION

We have presented EyeMRTK, a toolkit for eye-gaze-based interaction in virtual and augmented reality for the Unity engine. Following the concepts of VR toolkits, such as VRTK and SteamVR, EyeMRTK provides a hardware abstraction layer for common eye tracking systems ready for virtual and augmented reality. On top of that, it implements several eye-gaze-based interaction patterns that have been published in the research community.

The toolkit has been made publicly available at the GitHub repository of the COGAIN association. It can be accessed using https://github.com/The-COGAIN-Association/EyeMRTK.

Natural future contributions are an extension to cover more hardware platforms, such as Microsoft HoloLens 2, FOVE, Magic Leap, as well as desktop-based systems. Main future contributions on the functional level have already been sketched in the beginning of the paper, such as gaze pattern detection and extensive support for maintaining high quality operation of gaze-based interaction (accuracy, drift, calibration).

6.1 Future Vision: Interaction Benchmarks

Providing a hardware abstraction layer and a library of tested and evaluated interaction patterns is an important contribution to facilitate the adoption and dissemination of eye gaze as a viable interaction modality. Besides applications in games, edutainment or assistance systems, such patterns could also be used to allow people with special needs to participate in mixed reality environments at all.

One interesting extensions of the toolkit would be the definition and implementation of interaction challenges, which could be used to provide interesting benchmarks for the scientific community (e.g. standardized gaze-based Fitt's law-like testing batteries). Similar to image databases in computer vision/machine learning, such interaction challenges could help to make contributions to the field of gaze-based interaction more comparable.

6.2 Sustainability

The toolkit is currently supported by academic institutions and the progress of work on the toolkit will be ensured by a sequence of student projects. This work is reflected in a public repository at GitHub, with the aim to attract contributors from academia and industry alike. The vision is to provide something similar as VRTK for the mixed reality eye tracking community.

ACKNOWLEDGMENTS

This work was partly supported by the Cluster of Excellence Cognitive Interaction Technology 'CITEC' (EXC 277) at Bielefeld University, which is funded by the German Research Foundation (DFG). Diako is supported by the funding by the EPSRC project MODEM Grant No. EP/M006255/1.

REFERENCES

Jonas Blattgerste, Patrick Renner, and Thies Pfeiffer. 2018. Advantages of Eye-Gaze over Head-Gaze-Based Selection in Virtual and Augmented Reality under Varying Field of Views. In *COGAIN '18. Proceedings of the Symposium on Communication by Gaze Interaction*. ACM. https://doi.org/10.1145/3206343.3206349

Nathan Cournia, John D. Smith, and Andrew T. Duchowski. 2003. Gaze- vs. Hand-based Pointing in Virtual Environments. In *CHI '03 Extended Abstracts on Human Factors in Computing Systems (CHI EA '03)*. ACM, New York, NY, USA, 772–773. https://doi.org/10.1145/765891.765982

Brian Guenter, Mark Finch, Steven Drucker, Desney Tan, and John Snyder. 2012. Foveated 3D Graphics. *ACM Trans. Graph.* 31, 6 (Nov. 2012), 164:1–164:10. https://doi.org/10.1145/2366145.2366183

John Paulin Hansen, Kristian Tørning, Anders Sewerin Johansen, Kenji Itoh, and Hirotaka Aoki. 2004. Gaze Typing Compared with Input by Head and Hand. In *Proceedings of the 2004 Symposium on Eye Tracking Research & Applications (ETRA '04)*. ACM, New York, NY, USA, 131–138. https://doi.org/10.1145/968363.968389

Tobias Höllerer and Steve Feiner. 2004. Mobile augmented reality. *Telegeoinformatics: Location-Based Computing and Services. Taylor and Francis Books Ltd., London, UK* 21 (2004). 00533.

Felix Hülsmann, Timo Dankert, and Thies Pfeiffer. 2011. Comparing gaze-based and manual interaction in a fast-paced gaming task in Virtual Reality. In *Proceedings of the Workshop Virtuelle & Erweiterte Realität 2011*. https://pub.uni-bielefeld.de/publication/2308550

FOVE Inc. 2017. FOVE. (2017). https://www.getfove.com/ Accessed 2018-04-14.

Robert J. K. Jacob. 1990. 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 (CHI '90)*. ACM, New York, NY, USA, 11–18. https://doi.org/10.1145/97243.97246

Moritz Kassner, William Patera, and Andreas Bulling. 2014. Pupil: An Open Source Platform for Pervasive Eye Tracking and Mobile Gaze-based Interaction. *CoRR* abs/1405.0006 (2014). https://doi.org/10.1145/2638728.2641695 00096.

Manu Kumar, Jeff Klingner, Rohan Puranik, Terry Winograd, and Andreas Paepcke. 2008. Improving the accuracy of gaze input for interaction. In *Proceedings of the 2008 symposium on Eye tracking research & applications*. ACM, 65–68.

Jae-Young Lee, Hyung-Min Park, Seok-Han Lee, Soon-Ho Shin, Tae-Eun Kim, and Jong-Soo Choi. 2011. Design and implementation of an augmented reality system using gaze interaction. *Multimedia Tools and Applications* 68, 2 (Dec. 2011), 265–280. https://doi.org/10.1007/s11042-011-0944-5

Marc Levoy and Ross Whitaker. 1990. Gaze-directed Volume Rendering. In *Proceedings of the 1990 Symposium on Interactive 3D Graphics (I3D '90)*. ACM, New York, NY, USA, 217–223. https://doi.org/10.1145/91385.91449 00137.

LooxidLabs. 2017. LooxidLabs HomePage. (2017). http://looxidlabs.com/ Accessed 2018-04-14.

Magic Leap, Inc. 2019. Magic Leap One. (2019). https://www.magicleap.com/ Accessed 2019-01-25.

Päivi Majaranta and Richard Bates. 2009. Special issue: Communication by gaze interaction. *Universal Access in the Information Society* 8, 4 (March 2009), 239–240. https://doi.org/10.1007/s10209-009-0150-7 00000.

Päivi Majaranta and Andreas Bulling. 2014. Eye Tracking and Eye-Based Human–Computer Interaction. In *Advances in Physiological Computing*. Springer, London, 39–65. https://link.springer.com/chapter/10.1007/978-1-4471-6392-3_3

Diako Mardanbegi, Dan Witzner Hansen, and Thomas Pederson. 2012. Eye-based head gestures. In *Proceedings of the symposium on eye tracking research and applications*. ACM, 139–146.

Susanna Nilsson, Torbjörn Gustafsson, and Per Carleberg. 2009. Hands Free Interaction with Virtual Information in a Real Environment: Eye Gaze as an Interaction Tool in an Augmented Reality System. *PsychNology Journal* 7, 2 (Aug. 2009), 175–196.

Oculus. 2019. Oculus Integration for Unity. (2019). https://developer.oculus.com/ Accessed 2019-01-25.

Hyung Min Park, Seok Han Lee, and Jong Soo Choi. 2008. Wearable Augmented Reality System Using Gaze Interaction. In *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality (ISMAR '08)*. IEEE Computer Society, Washington, DC, USA, 175–176. https://doi.org/10.1109/ISMAR.2008.4637353

Ken Pfeuffer, Benedikt Mayer, Diako Mardanbegi, and Hans Gellersen. 2017. Gaze + Pinch Interaction in Virtual Reality. In *Proceedings of the 5th Symposium on Spatial User Interaction (SUI '17)*. ACM, New York, NY, USA, 99–108. https://doi.org/10.1145/3131277.3132180

Thammathip Piumsomboon, Gun Lee, Robert W Lindeman, and Mark Billinghurst. 2017. Exploring natural eye-gaze-based interaction for immersive virtual reality. In *3D User Interfaces (3DUI), 2017 IEEE Symposium on*. IEEE, 36–39.

Kari-Jouko Räihä. 2015. Life in the fast lane: Effect of language and calibration accuracy on the speed of text entry by gaze. In *Human-Computer Interaction*. Springer, 402–417.

Sensomotoric Instruments. 2018. Eye Tracking Solutions by SMI. (2018). https://www.smivision.com/ Accessed 2018-04-14.

Linda E. Sibert and Robert J. K. Jacob. 2000. Evaluation of Eye Gaze Interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '00)*. ACM, New York, NY, USA, 281–288. https://doi.org/10.1145/332040.332445

Oleg Špakov and Päivi Majaranta. 2012. Enhanced gaze interaction using simple head gestures. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*. ACM, 705–710.

Tuukka M Takala. 2014. RUIS: A toolkit for developing virtual reality applications with spatial interaction. In *Proceedings of the 2nd ACM symposium on Spatial user interaction*. ACM, 94–103.

Vildan Tanriverdi and Robert J. K. Jacob. 2000. Interacting with Eye Movements in Virtual Environments. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '00)*. ACM, New York, NY, USA, 265–272. https://doi.org/10.1145/332040.332443

TheStonefox. 2019. VRTK. (2019). https://vrtoolkit.readme.io/ Accessed 2019-01-25.

Tobii VR. 2019. Tobii Pro VR Integration based on HTC Vive HMD. (Oct. 2019). https://vr.tobii.com/ Accessed 2019-01-25.

Takumi Toyama, Daniel Sonntag, Jason Orlosky, and Kiyoshi Kiyokawa. 2015. Attention Engagement and Cognitive State Analysis for Augmented Reality Text Display Functions. In *Proceedings of the 20th International Conference on Intelligent User Interfaces (IUI '15)*. ACM, New York, NY, USA, 322–332. https://doi.org/10.1145/2678025.2701384

ValveSoftware. 2019. SteamVR Unity Plugin. (2019). https://github.com/ValveSoftware/steamvr_unity_plugin Accessed 2019-01-25.

Robert C Zeleznik, Andrew S Forsberg, and Jürgen P Schulze. 2005. Look-that-there: Exploiting gaze in virtual reality interactions. *Technical Report CS-05, Tech. Rep.* (2005).