A Unity Framework for Multi-User VR Experiences

Alexander Novotny, Rowan Gudmundsson, and Frederick C Harris, Jr.

Computer Science and Engineering
University of Nevada, Reno
Reno, NV USA 89557
anovotny@nevada.unr.edu,rgudmundsson@nevada.unr.edu, fred.harris@cse.unr.edu

Abstract

We have developed a framework for multi-user Virtual Reality experiences aimed at video games played over a network. Features include tracked avatars, interactable physics objects, peer-to-peer with a user matching system, and voice chat, as well as options to customize these modules for a wide range of support. We go into detail on how several implementation details, such as networking, voice chat, and interaction, work. We also go into detail on how to use the library in Unity for your own projects. We also talk about avatar representation in VR, and how this tool can be used to facilitate many different types of avatar representation.

Keywords: graphics, user-interface, virtual reality, multiplayer, networking

1 Introduction

Multi-Headset Virtual reality experiences are few and far between which brings about exciting new opportunities when looking at solidifying standards for interacting in this environment. Current multi-player frameworks aren't built to handle the intricacies of Virtual Reality support, and current Virtual Reality frameworks aren't built with the intention of having multiple headsets in the same virtual environment at the same time. As the popularity of virtual reality increases, the need for more diverse experiences will increase as well, and this will lead to a need for multi-player experiences, which have been previously under-explored. In this paper, we introduce a framework and several techniques for multi-user Virtual Reality experiences. This framework builds a foundation for a multitude of multi-headset experiences to be built on top of it. It provides developers with a networking connection layer, voice chat system, and networked physics system.

The rest of this paper is structured as follows: In Section 2 we cover target platforms, other tools, avatars, and social interaction. In Section 3, we discuss the design and implementation of our framework, which includes how we set up our networking stack and multi-user matchmaking in Section 3.1, multi-user interaction and object ownership in Section 3.2, avatar representation in Section 3.3, and multi-user voice chat in Section 3.4. We finish the paper with Conclusions and Future Work in Section 4.

2 Background Review

2.1 Target Platforms

Our framework is built for the Unity[9] game engine, but the techniques discussed can easily be extended to any other game engine. Unity was chosen due to its current popularity in individual game development as well as the availability of already established Virtual Reality and networking frameworks, such as Mirror[12], which was chosen for our framework. Mirror allows easy setup for the simple interactions that commonly occur in multiplayer virtual environments while allowing for the possibility of more complex networking interactions. Mirror also allows for simple peer-to-peer communication needed in simple 2-4 player games and server-client communication needed in massively multiplayer experiences. As well, our framework uses OpenVR [10] due to its hardware-agnosticism, however it can easily be extended to other Virtual Reality frameworks. OpenVR also requires Steam[11] to be running, so we targeted Steam users and took advantage of several features of the Steamworks SDK. However none of the methods discussed in this paper require the use of any of these pre-existing frameworks.

2.2 Other Tools

Networking libraries for Unity, such as Mirror [12], Photon [3], and SteamWorks, are common, but come with many downsides to the developer, especially when concerning VR. Photon, for instance, uses a client-server model, where players are matched on Photon's own servers, but Photon expectes developers to pay for this service. Steamworks, meanwhile, provides this service for free, but doesn't provide any networking layer for syncing objects in Unity itself. Mirror provides a good peer-to-peer system which syncs objects in Unity, but isn't built to accommodate VR headsets, and by default will not work with them at all. As well, as a peer-to-peer tool, Mirror does not provide a system of matching users, nor a voice chat system, which are necessary in many modern multi-user experiences.

VR libraries are also common - Unity has a VR library built in, for instance. OpenVR, one of the most popular cross-platform libraries that supports many headsets, also has support for Unity. However, none of these libraries are built to work with multiple headsets, let alone multiple headsets over a network.

2.3 Avatar Representation

When a user is immersed in a virtual environment, there are many choices when it comes to how to represent that user's self/body in the environment. There is good research about the implications/advantages of using different levels of representations of a user's own body in the environment when it comes to immersion, virtual awareness, and computation cost. In single-player experiences, increased complexity of player avatars doesn't gain any significant advantage in terms of immersion, while coming at the cost of framerate - something very important in virtual immersion [4, 5, 6, 8]. But in multiplayer experiences, there is an unexplored question of how other users' bodies should be represented and what needs to be tracked/networked to make that level of detail possible.

2.4 Social Interactions

With multiple users in a virtual experience, it becomes important for them to be able to interact in an expected way. This includes having perfect replication of environment, similar interaction

schemes between users, and feedback to let players know that they are interacting with another player [7, 13]. A multiplayer Virtual Reality framework must seek to efficiently implement these goals, and allow for other types of social interactions easily.

3 Framework Design

Our framework seeks to fill in the "holes" left by other libraries made for multi-user and VR experiences by providing a peer-to-peer networking stack with user matching which is free to the developer. Our framework is made for Unity, supports a wide variety of user interactions and avatar representations, and has voice chat built in.

3.1 Networking Setup

The networking API that our framework is built on top of is Mirror. Mirror uses a type of client-server communication where the server can also be a user of the software as well. To make connecting to other users easy, Steamworks was used. Opening of the software requires Steam to be open, and will load a list of friends who are currently running the game, as can be seen in Fig. 1. Selecting one of these friends will invite them to join a lobby, thereby starting a server on the user's local machine and marking the two player as a Steam "lobby". Further users will be able to see this lobby and instead of starting another server when inviting those friends, will instead join the already made server as an observer or other user. Steam allows connecting between users with their Steamworks API, allowing easy connections through firewalls without having to know the other users' IP address(es).

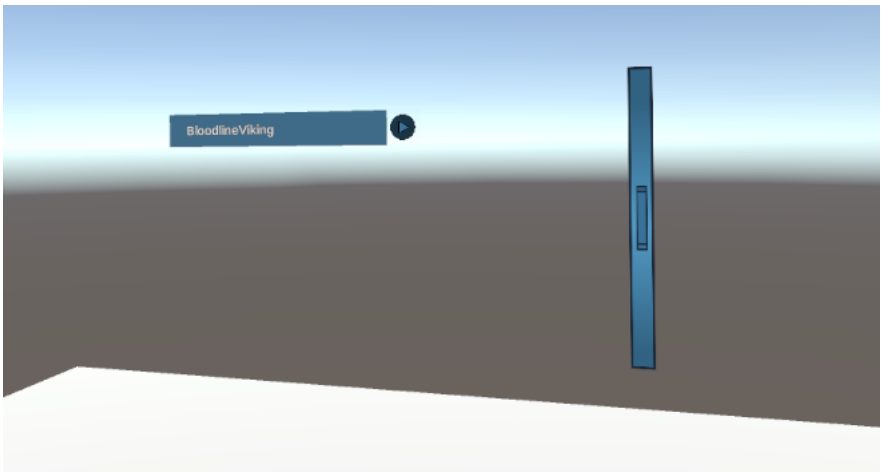

Figure 1: A menu displaying active Steam friends with which to join into a lobby

Starting a server opens a scene normally, but when other users connect to the server, the scene is then cloned to the connecting user. From then on, the server pushes updates to all connected clients syncing the scenes on their computers to the one on the server, so only changes made on the server will be represented to other users. Users can attempt to modify objects in their scene, but these changes will eventually be overwritten by the connection. As well, they can modify objects which are not synced over the network, however these changes will not be represented to other clients.

To override this behaviour, a mechanic called "authority" is used to determine which client has the authority to modify certain objects at any time. Each networked object has a single "authority figure" at any given time, and if this authority figure is not the server, the syncing behaviour changes to one where authority figure will push its changes on an object to the server, and the server will re-distribute these changes to all other clients. Using this "authority" method keeps networking costs down to a minimum, and ensures smooth physics if needed.

3.2 Multi-User Interaction

3.2.1 Tracking Users

The first step to creating a multi-user environment is tracking those users throughout the environment. This is done natively by many Virtual Reality frameworks, but typically not in a multi-user fashion. Our first attempt to track users in the environment was to simply network the objects tied to the tracked pieces of the user. However, this didn't work as using multiple player objects in a single scene caused each connected player to influence the motions of each player in the scene. OpenVR picks up all player objects in the scene as a controllable entity and so would change the position of the tracked points in each model simultaneously. In order to rectify this, we disabled all the components in the scene tracked by OpenVR which were not controlled by the local player (the player the current client is supposed to control) (Fig. 2). This method seemed like the most reasonable solution to the problem without delving too far into Steam's OpenVR implementation.

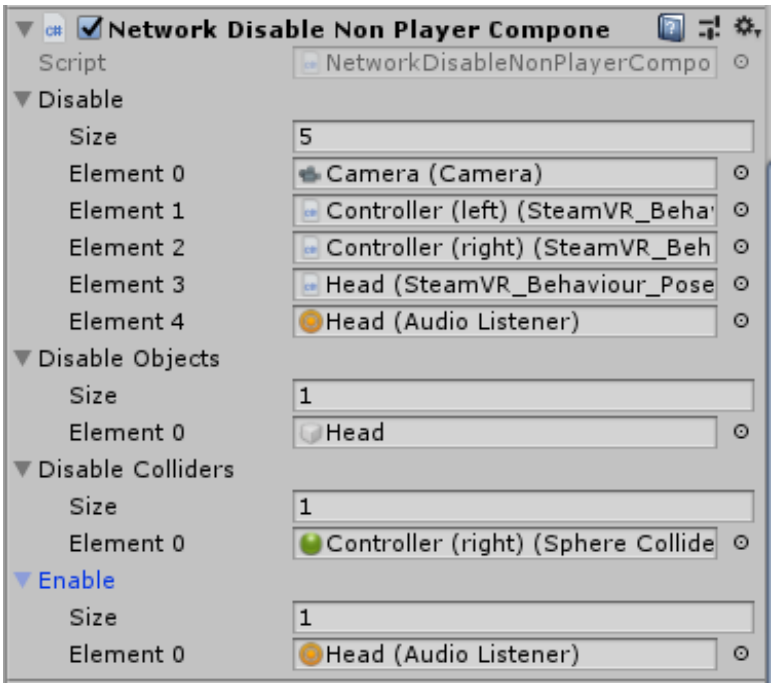

Figure 2: This figure shows the components which are being disabled for non-local players in the scene.

Additionally, in order to create a more immersive environment, we decreased the sync rate between the server and the clients to allow for more smooth movement in the players and interactable objects in the scene. We changed the sync rate from it's default of 100 milliseconds to 10 milliseconds. This gave the players almost seamless movement and made interacting with objects with multiple players very fluid and life-like.

3.2.2 Tracking objects

The next step in creating a multi-user environment is to allow players to interact with objects in the scene together. Doing this on a single machine in Virtual Reality is trivial since we only need to worry about keeping the physics updated on the local machine. In a multi-user setting however, we need to worry about how the physics of a given object is tracked across all clients. Some challenges we faced were figuring out which client should have "authority" over an object at a given time and how physics should be tracked over the network. We settled on only keeping track of physics on the machine which has authority over the tracked object and then just updating the position over all clients (Fig. 3). This method seemed to allow for the best performance since no information about the physics is transferred over the network only information about position, rotation, etc.

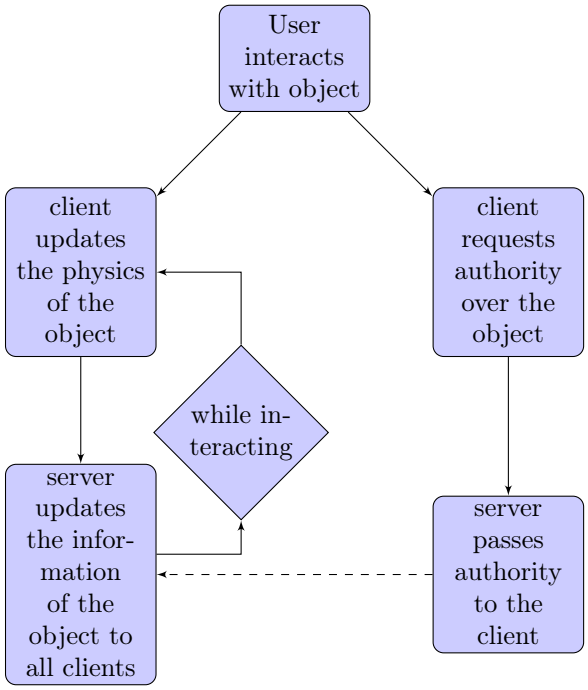

Figure 3: The flow of operation for interacting with an object on a client

3.3 Avatar Representation

In single-player virtual experiences, it has been shown that there is no notable increase in immersion or self-presence with the addition of more-realistic player avatars. However, in a multi-player environment, this can change. Not only does the user have to keep track of their

own avatar, but they now also must be able to keep track of other avatars as well. There is also now potentially a need for a user to be able to see the same avatar that everyone else is seeing.

To keep networking costs low, only three positions are sent over the network: head position, and hand positions. After these positions are sent over the network, then each client separately updates the avatars of each player with respect to these positions. This allows for a large amount of freedom with player avatars, including more complex player avatars through the use of techniques such as inverse kinematics. Players can choose which avatars they would like to represent themselves and others as without impacting the other players. Some examples of dynamic avatar representation can be seen in Figs. 4 to 6

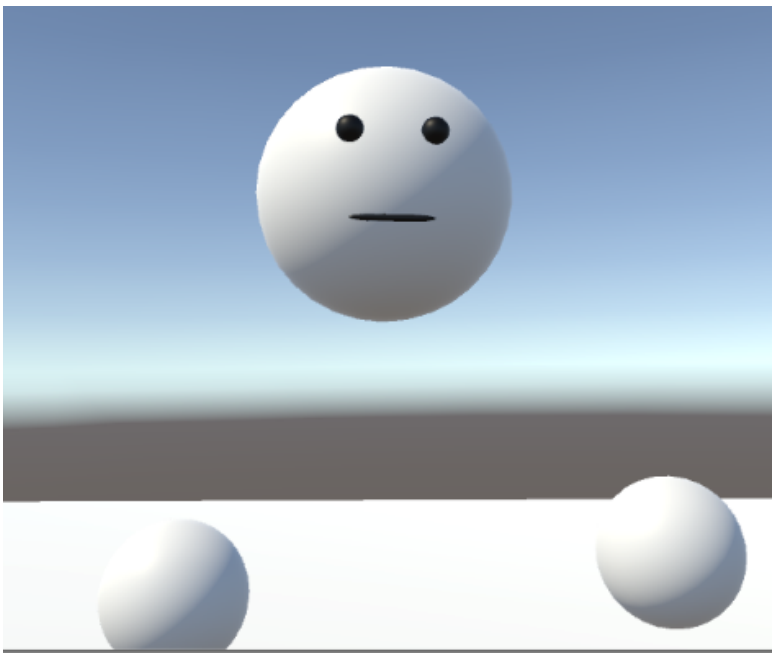

Figure 4: A simple player model which allows for good social interactions.

3.4 Multi-User Voice Chat

Another form of interaction one might wish to have in a multi-user virtual environment is speech. Indeed, every modern Head-Mounted Display made for Virtual Reality has a microphone array built-in with this purpose in mind, meaning voice chat is accessible to everyone.

Steamworks makes using these microphones easy - the library will automatically pick up on and compress any audio from the microphone on the headset. This is stored in a buffer until the appropriate retrieval function is called, upon which time it gives access to 16-bit compressed Pulse-Code Modulation (PCM) audio. This is ideal for sending over the network, and we implemented a custom network package to deliver the audio containing a buffer for the audio, a player ID to keep track of the origin of the audio, and a channel ID for special purposes. This is sent to the server, which then re-sends it to every other client. When a client receive this package, it finds the audio source associated to the player ID, decodes the audio, and stores it in a buffer waiting to play. This is a bit tricky, as the C# version of Steamworks returns a

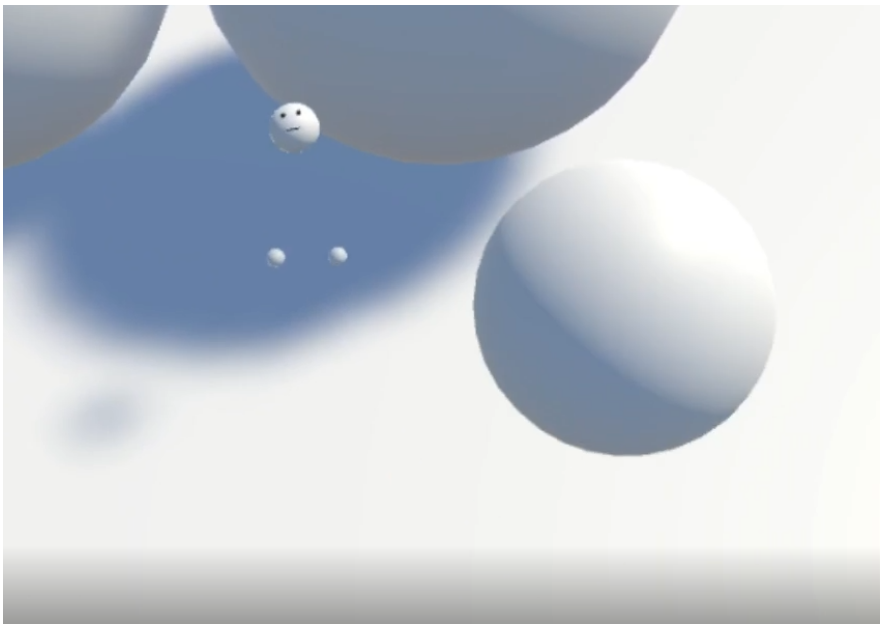

Figure 5: A much smaller player avatar - player avatars can be as flexible as you want!

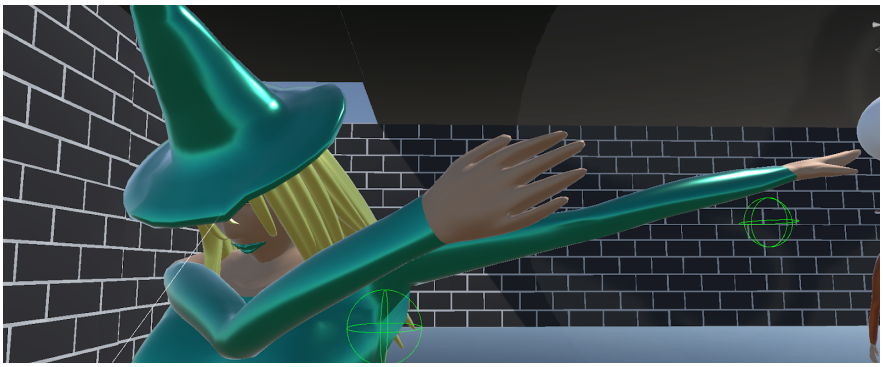

Figure 6: A complex player avatar rigged over the network [2].

buffer of 8-bit integer values to represent 16-bit audio and Unity requires 32-bit floating point values between -1 and 1. As well, C# is little-endian (i.e. high order bytes are stored after low order bytes), so there is a bit of finesse required to turn this audio into something useable. Once this is done, however, the audio source is set to stream from the buffer of incoming voice audio, making the audio sound like it is coming from that player.

As well, one can use the aforementioned channel ID to change how this works. Channels can be used to filter out certain players from hearing other players, joining certain voice chat channels, and changing which audio source to play the incoming audio from. For instance, a mining evacuation simulator used voice channels to make player audio come out of walkie-talkies instead being played directly from the other player, as players were often on other sides of the mine [1].

By Using SteamVR actions, this allows players and developers flexibility in how they want to be able to talk to other players. By default, the framework is set up to use a push-to-talk schema where as user will push a button to start talking and then push it again to stop while an indicator lets them know that they are broadcasting Fig. 7. This can be easily configured by both developers and users to become a hold-to-talk scheme or an always-on scheme where users are always chatting. The framework doesn't send voice packages over the network unless noticeable audio is detected, so silent users of an always-on scheme won't cause network stress. As well, action sets can be configured to make a more dynamic voice chat experience. For instance, the aforementioned mining simulator uses walkie-talkies whose push-to-talk button doesn't become available until the player picks up the walkie-talkie into their hand thereby switching action sets [1].

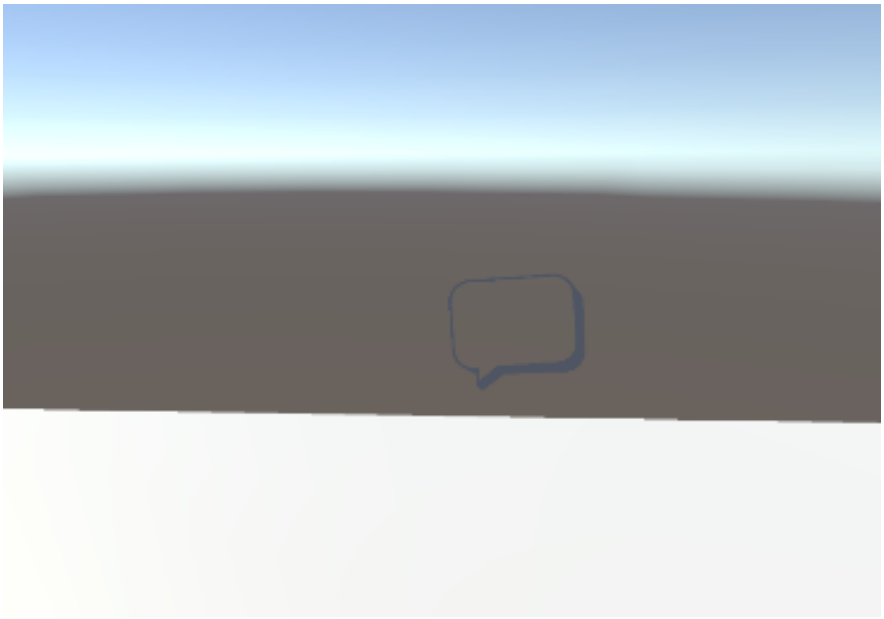

Figure 7: An indicator lets a user know that they are broadcasting to other players

4 Conclusions and Future Work

We believe this library will provide to be a useful framework for other multi-user virtual reality experiences, which are becoming more and more common. With integration with Steam, the world's largest game distribution and VR platform, and Unity, one of the largest game engines in use today, together with being free to the developer, we believe this framework will be a great boon to developers looking to get into this new market.

We are planning to include IBM Watson support with the voice-chat feature, which would allow for written transcripts of audio sessions during the experience. Planned use cases include 'replays' of scenarios in the experience, as well as voice-to-text chat in the experience. We also plan to use the feature in our own virtual reality game to trigger certain features of the game off of certain key phrases.

Other projects which already use this framework include [1] and [2]. We also are planning on releasing our own multi-user virtual reality game, which was the biggest reason for making the framework. We are planning on having asymmetric game-play, which gave rise to our emphasis on avatar representation.

Acknowledgment

This material is based in part upon work supported by the National Science Foundation under grant numbers IIA-1301726. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] K. Andersen, S. J. Gaab, J. Sattarvand, and F.C. Harris, Jr. METS VR: Mining evacuation training simulator in virtual reality for underground mines. In *Proceedings of the 17th International Conference on Information Technology : New Generations (ITNG 2020)*, 2020.

[2] L. Calabrese, A. Flangas, and F.C. Harris, Jr. Multi-user VR cooperative puzzle game. In *Proceedings of the 17th International Conference on Information Technology : New Generations (ITNG 2020)*, 2020.

[3] Exit Games. Photon. https://www.photonengine.com/en/pun, Last Accessed: 11/27/2019.

[4] L. Kruse, E. Langbehn, and F. Stelnlcke. I can see on my feet while walking: Sensitivity to translation gains with visible feet. In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 305–312, 3 2018.

[5] J. Lugrin, M. Ertl, P. Krop, R. Klüpfel, S. Stierstorfer, B. Weisz, M. Rück, J. Schmitt, N. Schmidt, and M. E. Latoschik. Any "body" there? avatar visibility effects in a virtual reality game. In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 17–24, 3 2018.

[6] J. Lugrin, M. Wiedemann, D. Bieberstein, and M. E. Latoschik. Influence of avatar realism on stressful situation in vr. In *2015 IEEE Virtual Reality (VR)*, pages 227–228, 3 2015.

[7] D. Roth, C. Klelnbeck, T. Feigl, C. Mutschler, and M. E. Latoschik. Beyond replication: Augmenting social behaviors in multi-user virtual realities. In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 215–222, 3 2018.

[8] D. Roth, J. Lugrin, D. Galakhov, A. Hofmann, G. Bente, M. E. Latoschik, and A. Fuhrmann. Avatar realism and social interaction quality in virtual reality. In *2016 IEEE Virtual Reality (VR)*, pages 277–278, 3 2016.

[9] Unity Technologies ApS. Unity. https://unity.com/ Last Accessed: 11/27/2019.

[10] Valve Corporation. OpenVR. https://github.com/ValveSoftware/openvr, Last Accessed: 11/27/2019.

[11] Valve Corporation. Steam. https://steampowered.com, Last Accessed: 11/27/2019.

[12] vis2k. Mirror. https://github.com/vis2k/Mirror, Last Accessed: 11/27/2019.

[13] C. Wienrich, K. Schindler, N. Döllinqer, S. Kock, and O. Traupe. Social presence and cooperation in large-scale multi-user virtual reality - the relevance of social interdependence for location-based environments. In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 207–214, 3 2018.