AUGMENTED SPACE LIBRARY: HYBRID P2P LIBRARY FOR REMOTE CROSS REALITY INVESTIGATIONS

Gregory Smith, Kelvin Sung

Computing and Software Systems, University of Washington (UNITED STATES)

Abstract

The maturing of Augmented Reality (AR) and Virtual Reality (VR) related technologies has enabled increasingly sophisticated investigations into their applications. An exciting recent development is in the investigations into remote collaborations where geographically distributed participants can be equipped with distinct hardware configurations, such as handheld AR, immersive VR, or typical PCs, while interacting in an integrated and homogeneous environment, e.g., [1], [2]. These are cross reality collaborations because the participants operate in a spectrum of technologically enabled realities across the augmented physical world and digital virtual spaces [3].

Cross reality collaboration is a new and stimulating field with many enthusiasts, especially among undergraduate students who grew up with the technology. However, even with increasingly accessible hardware, due to a lack of dedicated infrastructural support, the barrier for developing a simple remote cross reality investigative application remains significant. For this reason, explorations of related ideas are off limit to many undergraduate students who can typically dedicate only one or two academic terms for such activities.

This paper first examines the requirements of and articulates a novel hybrid peer-to-peer (P2P) infrastructure specifically for supporting the rapid prototyping of remote cross reality ideas. These requirements are used as the design guideline for the Augmented Space Library (ASL). Following the design discussion, the paper details an implementation of the ASL where the frontend peer-to-peer Application Programming Interface (API) model is presented as an extension to Unity, an accessible and free-of-charge technology for undergraduate students, with a hidden backend server based on the GameLift service of Amazon Web Services (AWS).

The paper then describes and analyzes the results of six undergraduate student projects based on the ASL. These projects, each with two to three team members, were brainstormed, proposed, prototyped, tested, and refined in the period of one academic term. Interestingly, and coincidentally, with the current COVID-19 situation, all collaborations of the projects were carried out remotely. Though with areas for future improvements, the results from these projects demonstrated the validity of the articulated considerations and design decisions, and that ASL is indeed an effective tool for supporting undergraduate students in investigating cross reality collaboration applications.

Keywords: Augmented Reality, Virtual Reality, Cross Reality, Hybrid P2P, Network API, Unity, Rapid Prototype.

1 INTRODUCTION & RELATED WORK

The recent resurgence of Augmented and Virtual Reality (AR/VR) related technologies, especially applications in non-entertainment related industries, indicates that AR/VR technologies are maturing [4]. As the field continues to develop, AR/VR technologies are increasingly being relied upon as foundations for investigations across reality spaces where solutions span the continuum between virtual and physical spaces [5]. For example, the support for multiple remote interior designers to collaborate with an on-site owner in furnishing her currently empty laboratory space; where, some remote designers are equipped with immersive VR while others are interacting on a PC or via sensing interfaces such as the Microsoft Kinect, and the on-site owner is equipped with immersive AR [3].

Remote cross reality collaborations involve technologically enabled communal environments where participants can cooperate to manipulate shared objects to accomplish joint tasks. Investigation of ideas in this area requires specific and non-trivial infrastructures including network communication, object state synchronization, and hardware reality device interfaces. For time constrained enthusiasts with simple ideas, including hobbyists and typical undergraduate students, the infrastructure requirements can be a formidable barrier prohibiting the exploration of simple and creative ideas. In

particular, for undergraduate students the time available for examining ideas is usually limited to one or two academic terms.

Existing research in remote cross reality collaboration focuses on many important issues, including specialized hardware, e.g., [6], system configurations, e.g., [7], design efforts, e.g., [8], the details of many creative applications, e.g., [1]–[3], [9], [10] and the approaches to analyze the results e.g., [11]. While exciting and enlightening, these efforts and their corresponding results do not elaborate on the requirements of the infrastructure to support such investigations. Although general support for remote information sharing is relatively mature, well-established solutions are typically based on the traditional client-server [12] or peer-to-peer [13] models, and as will be discussed, may not align well with the needs of time-constrained remote cross reality investigations. For these reasons, even with the rapidly decreasing costs in hardware and increasing qualities in software development kits (SDK), the barrier for starting a remote cross reality collaboration investigation remains high.

This paper articulates the infrastructural considerations for supporting rapid prototyping of remote cross reality collaborative investigations by identifying relevant disciplines and analyzing their characteristics. These considerations fuelled the design decisions of the Augmented Space Library (ASL) [14], a SDK extension to the Unity game engine [15] where the logistics of connecting remote participants are hidden and investigators can proceed with examining their ideas of collaboration across realities without concerns of network connection and object sharing solutions.

The results from six ASL based one-academic-term undergraduate projects indicated that, though with areas for improvement, the articulated considerations and the associated solutions do address and lower the hurdle for, and, that ASL is an effective tool supporting novice investigators.

2 INFRASTRUSTURE CONSIDERATIONS

An infrastructure that supports the development of investigative remote cross reality collaboration applications must present a coherent application programming interface (API) that integrates utilities across multiple disciplines. A coherent and effective API would allow the investigators to focus on idea explorations and not become distracted by the remote sharing aspect. The first step to understand the necessary requirements for such an API is to analyze the characteristics of needs in the involved fields.

2.1 Remote

For an application to facilitate the collaboration of geographically distributed participants, it must support straightforward connections of the participants and sharing of objects. The fields of remote access and object sharing are relatively mature with many well-established solutions. In the context of cross reality collaboration, the following are essential considerations.

- **Participant connection**: The participants are likely to be trusted, distributed geographically, and may not know the technical setup and specifics locations of each other. Such participants must be able to locate each other and establish a collaborative session effortlessly.
- **Application state synchronization**: The presentation of different realities mainly involves graphical geometric objects in 3D spaces. The investigative nature of these applications implies that the actual application states may vary widely. System synchronization mechanisms must provide elaborate support for graphical objects and be flexible to support arbitrary state information.

2.2 Cross Reality

Cross reality refers to the fact that, depending on the technological setup of each participant, they can experience different versions of reality across the continuum between actual physical world and virtual digital space. There are two essential considerations.

- **Device independence**: The different realities are delivered via various hardware configurations; thus, the distinct hardware devices must be supported by the underlying infrastructure.
- **Common referencing coordinate system**: The referencing coordinate system of an augmented world is typically and conveniently defined to be the AR device's camera position and orientation when the application is first started. With multiple participants on different AR

and VR devices collaborating across distances, the supporting system must allow a clear and consistent way of defining a common coordinate system.

2.3 Collaboration

The infrastructure should be optimized to support collaborative behavior patterns including mutual learning and coordination [16]. In a collaborative session, participants are likely to observe one another, take turns, and work alongside one another. Under such a model, contention over shared objects is likely to be infrequent. Though not a technical requirement, this observation is an important guide in system design.

2.4 The Investigative Applications

The last category of consideration is the functionality of the actual investigative application. Upon analyzing recent efforts, e.g., [1]–[3] and based on our own experiences, e.g., [17], the following observations can be drawn.

- **Object behavior**: Objects in these applications are typically either static, e.g., a part of the common environment and do not change once created, or dynamic where they can be manipulated by the participants. Though there may be objects with pre-defined autonomous behaviors, e.g., driven by a physics simulation engine, unlike typical interactive games, such objects may not be present and when they do exist, the number of such objects are likely to be low.
- **Participant-Object interaction**: Participants typically select an object and then perform intuitive manipulations such as moving, re-orientating, or re-sizing. This observation leads to two implications. First, only the transforms of 3D objects need be synchronized where the potentially complicated geometry can be built-in as part of the application and thus need not be synchronized during runtime. For example, 3D models of geometrically complicated cars can be distributed as part of the application and during runtime only the transform of the car object needs to be synchronized. Second, to properly support mutual learning and coordination, participants must be able to clearly identify object ownership so that they can effortlessly transition between being an observer and manipulator. Simple visual feedback, such as highlight color or texture, is an excellent way to indicate ownership of objects.
- **Straightforward functionality**: Investigative applications are typically focused in scope, designed to explore specific ideas in relative isolation. For example, in the collaboration with remote interior designers to furnish the empty laboratory space example, the focus is on understanding effective distant object manipulation [3], the investigation is not concerned with many other aspects such as graphical user interface design, support for physics simulations, or detailed high-quality rendering. In this way, the corresponding infrastructure should also be of limited scope and ensure completeness of support within those scopes, e.g., elaborate, and efficient support for object transform synchronization without specific support for rendering quality control.

Lastly and very importantly, the infrastructure must be simple, with a flat learning curve that supports rapid prototyping. An important goal of this work is to support time-constrained undergraduate research projects.

3 DESIGN DECISIONS

The discussed considerations point to an API that amalgamates three areas: network architecture, information sharing, and the rapid prototyping of interactive graphical applications that support multiple AR/VR devices. The essential design decisions for each are discussed next.

3.1 Network Architecture

The underlying infrastructure must allow straightforward establishment of collaboration sessions and support efficient and flexible application state management among all the participants.

With a client-server model, a participating client can connect to a collaborative session with the sole knowledge of the well-known server and does not need to be concerned with the details of any other participants. For this reason, the application state would be maintained by the server where

communications with the clients are essentially state update requests. Because of the ease of establishing connections and centralized application state control, the well-established network infrastructure solutions for supporting remote interactive graphical applications are all based on the client-server model e.g., [18], [19], [20]. While a centralized state management solution facilitates consistency and efficiency, it is also true that a client-server-based application consists of essentially two separate solutions—one each for the client and the server, and therefore can be challenging for novices to rapidly prototype simple ideas.

In stark contrast, a peer-to-peer (P2P) model requires all participants to explicitly connect to each other to establish a session, and each peer is responsible for ensuring a coherent and properly synchronized application state. While there are investigations of collaborative applications based on the P2P model including interactive 3D applications, e.g., [21], [22], distributed virtual environment simulations e.g., [23], and multimedia applications, e.g., [24], the focus of these projects are on their respective applications and do not address general infrastructural needs. Although joining a network session can be logistically challenging, once established and with proper synchronization support, application state management for each peer can be similar to that of a single participant application.

An ideal solution would be a hybrid network model with the client-server approach for establishing collaborative sessions, and the P2P solution for managing the application state by each participant. With such a model, an investigator can begin by exploring ideas and developing simple applications for one participant, and generalize their solution to increase the number of participants by simply allowing more P2P users. The investigator would never have to worry about the application state consistency, the communication requirements, or the geographic location of other participants.

3.2 Information Sharing

With the application state being maintained by individual participants, updates of shared objects must be synchronized. A straightforward approach is object ownership locks where only the owners of object locks can send update information for the corresponding object [25]. The control of this locking mechanism can either be distributed, e.g., the current owner must release a lock voluntarily [26], or centralized, e.g., a server grants and reclaims locks [25]. The former respects each owner but risks starvation, the latter allows a centralized policy but can significantly increase the server complexity.

While the length of time that a participant may hold on to an object can vary, the collaborative behavior patterns suggest that the transfer of ownership is likely to be an orderly sequence [16]. For this reason, the locking mechanism can assume a relaxed and systematic ownership transfer and focus on simplicity in both implementation and usage. A hybrid approach where locks can be voluntarily released by current owners, and when necessary, forcefully reclaimed by the server can be straightforward to implement and easy to work with in non-contentious settings.

A trivial implementation of such a system would allow the server to mediate object locks with a simple system of per-object states. By default, objects are owned by the server where object ownership arbitration is as follows.

- **Objects not owned**: Claims of unowned objects will always be granted with the per-object state recording the current owner.
- **Objects owned**: Claims of currently owned objects will result in a wait object state and the server initiating the reclaiming of ownership lock. Clients always honor the server reclaims and send acknowledgement of ownership release. When server regains ownership, the currently waiting participant will be granted ownership.
- **Objects in wait state**: Claims of objects currently in wait state will always be denied.
- **Object contention**: At any instance, if the server should receive more than one ownership claim requests, only one will be honored and the rest denied. There are two important reasons behind this strategy. The first is to guarantee definitive responses to the interactive client applications. Without an ownership wait queue in the server, the client application is guaranteed a success or failure response within the expected network delay and is therefore never in a state of owning a lock sometime in the unknown future. The second reason is simplicity—without explicit queues, the server state management can be kept simple and elegant.

With this ownership arbitration scheme, a contentious object will constantly be in a wait state impacting system productivity. However, it is assumed that collaborative participants will seldom

compete for object ownership. In all cases, contentious or otherwise, because of nondeterministic network delays, object ownership starvation is unlikely.

3.3 Rapid Prototyping

The goal of the API is to support the quick prototyping of focused investigative interactive graphical applications that support different AR/VR devices and allow participants from different geographical locations to work together. It is fortunate that the goal of supporting the building of interactive graphical applications aligns well with that of typical game engines [27]. Additionally, many well-established game engines support multiple popular AR/VR devices [28].

For these reasons, a straightforward approach would be to develop the API as an extension to an existing game engine. In all cases, the support of rapid prototyping with a flat learning curve requires that the API be simple, straightforward to install, and well-documented with ample and useful tutorials.

4 THE AUGMENTED SPACE LIBRARY (ASL)

Based on the considerations and design decisions discussed, the Augmented Space Library (ASL) was built to support the rapid prototyping of investigative remote collaborative applications where the participants can be equipped with heterogeneous devices [14]. This section overviews the API implementation and the SDK delivered.

To the investigators who are building the investigative applications, ASL is a collection of functions similar to any API. ASL developers need not be aware of the existence of a separate server.

4.1 Choice of Unity

Unity is a popular commercial game engine with a friendly graphical user interface and is relatively straightforward to learn [15]. The Unity developer ecosystem includes elaborate API documentations [29], a vibrant user community with many extension tools and tutorials [30], sensible educational licenses [28], and support and plans to continue to improve the support for all existing and future major AR and VR devices [31]. Because of these factors, and due to our own experiences with using the system in academic settings [32], Unity was chosen to be the hosting platform for ASL.

It should be noted that while Unity is a game engine, Unity applications built with ASL may not be games. Though investigations into cross reality often include game-like elements, including real-time interactions with virtual 3D objects. These elements are meant to assist the exploration of collaboration across the different hardware mediums.

4.2 The Frontend API

The two key factors governing the API design and implementation are the support for 1) rapid prototyping with a flat learning curve and 2) interactive graphical applications with focused scope. The API must be simple with in-depth support of functionality within that scope. In this context, the scope of the API is to support synchronization of the graphical application state. These considerations resulted in three general categories of functions.

- **3D Object Support**: As discussed, the need is to synchronize the results of interactive manipulations: the transforms and not the geometry of the 3D objects. ASL strives to provide the complete spectrum of functions supporting objects and their transforms including creation, deletion, ownership control, and incremental and absolute updates of transforms. Besides the need to acquire an object ownership before manipulation, the sharing and synchronization aspects of these objects are completely transparent to the developers. Synchronization of object color and texture are also supported for straightforward feedback of object ownership.
- **General State Support**: Due to the investigative nature of these applications, it is impossible to predict the needs for synchronizing the state of any application. A generic approach would be to support the synchronization of a byte data block. Based on our experience, application states are frequently encoded as a collection of numbers. To avoid excessive data type conversions, ASL chooses to support the synchronization of a floating-point data bock. Participants can communicate custom arrays of floating-point numbers they encoded for any arbitrary purpose.
- **AR Specific Support**: ASL wraps Google's ARCore library [33] to support the synchronization of ARCore's cloud anchors to share a common reference coordinate system amongst all

collaborators. To avoid confusions over different coordinate systems, ASL offers distinct and explicit supports for the selection of AR scanned reference planes and regular 3D objects.

It is important to note the emphasis on simplicity of over support for general utility. For example, the AR selection utility in different coordinate system is the only network-independent utility function defined in the ASL API. This philosophy is true even for synchronization support—there is no explicit support for the sharing of 3D object geometries, e.g., the vertices and material properties of a car. The assumption is that geometric objects, including ones in the common environment, will be defined as part of the application where during runtime, only the transformation of these objects are of interest. When necessary, e.g., to communicate an AR application scanned geometry, at runtime applications can communicate object geometry information via the custom floating-point data block.

Notice that the there are no explicit functions supporting the establishment of collaborative sessions. The ASL startup process guides participants via an intuitive GUI interface to identify their unique session identifier and automatically establish connections.

4.3 The Backend Server

There are many excellent network solutions, e.g., GameSparks [19], or GameLift [20], even ones specifically designed for Unity developers, e.g., Photo Engine [18]. However, these are all based on the client-server architecture. ASL's server is based on the AWS GameLift services [20] where participant connection requests are serviced by server-less AWS Lambda functions and object ownership control and object state broadcasting are supported by a lightweight server [34]. In this way, the lightweight server acts more as a relay with minimal state information, e.g., which participants own which objects, and not as a typical server in the client-server model. The correctness of the per-object state ownership control requires the system to guarantee packet orders and delivery. For this reason, all communications are based on TCP sockets.

It is important to re-emphasize that the existence of the backend server is completely transparent to ASL application developers. An investigator can simply focus on refining their ideas, developing the application based on their ideas, and at no time needs to be concerned that there is a backend server.

4.4 Documentation and Tutorials

ASL is released with detailed documentation, including guides on installation and configuration, explanations on API functions, tutorials for programming with individual API functions, and more elaborate tutorials on approaches to combine the functions to build simple functionality [35].

5 RESULTS

ASL has gone through two stages of testing: basic quality assurance (QA) and programmability. Towards the end of the second phase, a survey was conducted with response from nine (9) of the developers.

5.1 Quality Assurance and SDK Tutorials

QA testing includes basic unit and correctness stress testing. Unit tests are trivial iterations over parameters of individual or groups of related functions. These test cases are created to cover every function and are maintained throughout the development process. In the end, these sets of test cases serve as excellent secondary documentation and simple tutorials for the investigators to experiment on and learn from.

The stress tests are designed to verify the ownership control scheme and assess system performance capacity. In the case of ownership verification, test cases with continuous automated creation, manipulation, and deletion of shared objects among multiple participants were built and ran over 48 hours to verify complete and correct synchronized states. Performance test results, though dependent on actual network bandwidth and traffic, showed that ASL can easily support a small number of participants, e.g., 5, sharing and manipulating several hundred shared objects simultaneously. Similar to the unit test cases, these tests serve as excellent secondary tutorials on how ASL functions can be combined to accomplish simple tasks.

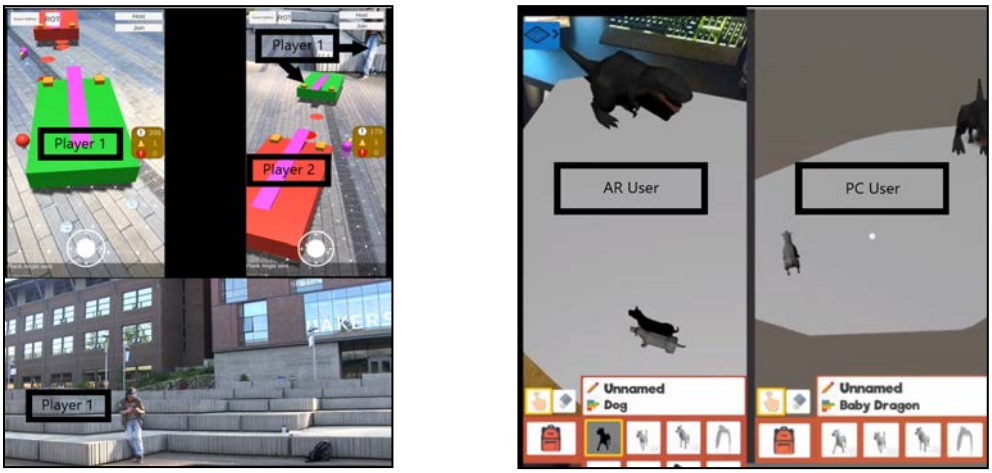

Figure 1. Left: AR Ballista Challenge credit: Saiful Salim. Right: AR Pet Farm credit: Sean Miles and Marc Skaarup

5.2 Programmability Testing

Driven by their personal interests, eleven senior-level computer science undergraduate students volunteered, working individually or in teams, on six independent investigative projects based on ASL. Each project, from ideation, implementation, end-user testing, to final refinements lasted exactly one academic quarter, or 10 weeks. All project ideas were from the students, where some had end-user VR experience, but none had prior AR/VR programming exposure or previous experience working with ASL. When the projects began, the students only had each other and the existing ASL documentation for reference. Of the six projects, one was based on AR-only devices, three were based on AR/PC collaboration, and two were VR devices with potential PC collaboration. Please refer to the Cross Reality Collaboration Sandbox (CRCS) Research Group website for details of these projects [36].

Fig. 1 shows the screenshots from the AR-only application (left), and an example of AR/PC collaboration (right). On the left of Fig. 1, the top two screenshots are views from the two AR devices and the bottom photograph shows Player 1 interacting with his AR device. This application challenges the two players to control their ballista through their AR devices to hit the opponent's ballista.

The two screenshots on the right of Fig. 1 are from the AR Pet Farm project, where the AR participants can scan and create farm areas in the physical world, share the area with PC participants, and raise and nurture pets in these areas. In this case, the left screenshot shows an AR user created a farm area on the desk in front of the keyboard and shared that area with the PC player on the right. The two players are collaborating in raising and nurturing a dog, and what appears to be a T-Rex dinosaur.

Fig. 2 shows an investigation into a collaboration application with VR devices where participants must work together to solve multiple puzzles and escape the virtual room. The top two screenshots are views of what the player sees and the bottom photos are the respective players. The top-left screenshot shows Player 1 observing Player 2's attempt at solving the pipe-puzzle, and the top-right screenshot shows Player 2 interacting and manipulating the pipes.

The other projects include two collaborative applications between AR participants and distant PC players. The first project challenges the participants built a bridge over a living-room floor turned into a lava field, and the second was focused on competing to collect coins on surfaces scanned from the AR participants' environment. The last project was a VR collaboration application in a Minecraft-like environment where the participants must mine and build to reach a destination castle.

5.3 Survey Results

A survey questionnaire was instituted at the end of these projects to assess students' experience. The results indicated general agreement on ease in establishing network sessions and success in synchronization support. Also discovered was the difficulty in integrating with Unity's physics engine and the lack of VR device specific support, especially in UI and debugging [14].

Figure 2. VR Escape room credit: Cody Thayer, Isaiah Snow, and Yuto Akutsu.

5.4 Discussion

It is ironic that with the current COVID-19 situation, these projects proceeded with all participants collaborated remotely in the entire development process. Within the first week each project team was able to successfully prototype simple PC only collaborative applications where they can manipulate shared objects remotely. The first two to three weeks were dedicated to idea evaluation and prototyping with corresponding hardware. The first draft version of applications began to appear by the mid-quarter, at around fifth weeks' time frame. In all cases, the first draft version was single-user and remote participants were included only in later releases when the core functionalities were verified.

The drastically different applications, the different combinations of realities involved, the rapidly ramped up prototypes, the remote and collaborative natures of both the development process and end-results, the success of the unpolished investigative applications, and the positive feedbacks from the developers all attest to the success of ASL meeting the goal of supporting the rapid prototyping of remote cross reality collaboration investigations.

6 CONCLUSION

This paper identified that, to support rapid prototyping of investigations into cross reality collaboration, the underlying infrastructure should support 1) effortless establishments of collaborative sessions; 2) efficient synchronization for graphical objects and generic data blocks; 3) different AR/VR devices; 4) explicit establishment of common referencing coordinate system. Additionally, it is reasonable to assume friendly and trustworthy participants where shared objects are manipulated in coordinated fashions.

These observations lead to a hybrid client-server and P2P network architecture that offers easy participant connections and facilitates per-peer application state management, supporting straightforward prototyping without special handling of shared objects. The assumption on collaborative shared object manipulation allows object ownership control mechanism to be optimized, focusing on simplicity. A custom object ownership arbitration system is proposed and implemented to ensure trivial server implementation and guarantees deterministic response to the client.

The derived solution is implemented in the ASL system as a SDK extension to the Unity game engine. The extensive testing by six groups of undergraduate students with drastically different projects verifies the validity of the identified considerations, the design of the solutions, and the effectiveness of the ASL to be an infrastructure for novice investigators.

Future improvements for ASL are in areas that were aggressively optimized in favor of API simplicity. For example, the support for autonomous behaviors of shared objects where five of the six projects implemented crude solutions for, e.g., wandering of pets in the farm. Another functionality would be the synchronization according to time such that real time events can occur at the exact same moment. It is important to note that while these functionalities do not explicitly exist, they can be implemented based on the generic floating-point data block. It will be an on-going optimization decision to balance between functionality and API simplicity. Finally, as the SDK for the AR/VR devices continue to improve, it is critical to constantly re-evaluate the integration and support for these devices.

ACKNOWLEDGEMENTS

The Microsoft Mixed Reality Academic Seeding Program donated all of the HTC Windows Mixed Reality headsets. Thanks to the adventurous and courageous undergraduate students who volunteered and participated so enthusiastically in this investigation, even during the very difficult pandemic lockdown. The first author and the AR equipment involved in these projects are supported by generous grants from the Computing and Software System Division, at the University of Washington Bothell.

REFERENCES

[1] S. A. Alharthi, K. Spiel, W. A. Hamilton, E. Bonsignore, and Z. O. Toups, "Collaborative Mixed Reality Games," in *Companion of the ACM CSCW and Social Comp.*, Oct. 2018, pp. 447–454.

[2] J. Müller, J. Zagermann, J. Wieland, U. Pfeil, and H. Reiterer, "A Qualitative Comparison Between Augmented and Virtual Reality Collaboration with Handheld Devices," in *Proceedings of Mensch und Computer 2019*, Hamburg Germany, Sep. 2019, pp. 399–410.

[3] M. Tanaya *et al.*, "A Framework for analyzing AR/VR Collaborations: An initial result," in *2017 IEEE International Conference on CIVEMSA*, Jun. 2017, pp. 111–116.

[4] B. Marr, "The 5 Biggest Virtual And Augmented Reality Trends In 2020 Everyone Should Know About," *Forbes*. https://www.forbes.com/sites/bernardmarr/2020/01/24/the-5-biggest-virtual-and-augmented-reality-trends-in-2020-everyone-should-know-about/ (accessed Jan. 04, 2021).

[5] L. Avila and M. Bailey, "Augment Your Reality," *IEEE CG&A*, vol. 36, no. 1, pp. 6–7, Jan. 2016.

[6] S. Wong, S. Singhal, and C. Neustaedter, "Smart Crew: A Smart Watch Design for Collaboration Amongst Flight Attendants," in *Companion of the 2017 ACM CSCW and Social Computing*, Feb. 2017, pp. 41–44.

[7] E. Peters *et al.*, "Design for Collaboration in Mixed Reality Technical Challenges and Solutions," Nov. 2016, doi: 10.1109/VS-GAMES.2016.7590343.

[8] K. M. Everitt, S. R. Klemmer, R. Lee, and J. A. Landay, "Two Worlds Apart: Bridging the Gap Between Physical and Virtual Media for Distributed Design Collaboration," *NEW Horiz.*, no. 5, p. 8, 2003.

[9] T. Teo, L. Lawrence, G. A. Lee, M. Billinghurst, and M. Adcock, "Mixed Reality Remote Collaboration Combining 360 Video and 3D Reconstruction," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, May 2019, pp. 1–14.

[10] C. Elvezio, M. Sukan, O. Oda, S. Feiner, and B. Tversky, "Remote collaboration in AR and VR using virtual replicas," in *ACM SIGGRAPH 2017 VR Village*, Jul. 2017, pp. 1–2.

[11] C. Y. Wang, L. Drumm, C. Troup, Y. Ding, and A. S. Won, "VR-Replay: Capturing and Replaying Avatars in VR for Asynchronous 3D Collaborative Design," in *2019 IEEE Conference on VR and 3D User Interfaces*, Mar. 2019, pp. 1215–1216.

[12] "Client–server model," *Wikipedia*. Dec. 25, 2020, Accessed: Jan. 04, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Client%E2%80%93server_model&oldid=996262483.

[13] "Peer-to-peer," *Wikipedia*. Dec. 13, 2020, Accessed: Jan. 04, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Peer-to-peer&oldid=993999508.

[14] G. Smith, "Augmented Space Library 2: A Network Infrastructure for Collaborative Cross Reality Applications," Master's, University of Washington, United States -- Washington, 2020.

[15] "Unity (game engine)," *Wikipedia*. Jan. 04, 2021, Accessed: Jan. 04, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Unity_(game_engine)&oldid=998256828.

[16] B. Wasson and A. Mørch, "Identifying collaboration patterns in collaborative telelearning scenarios," *Educ. Technol. Soc.*, vol. 3, Jan. 2000.

[17] A. Hitchcock and K. Sung, "Multi-view augmented reality with a drone," in *Proceedings of the 24th ACM Symposium on VRST*, Tokyo Japan, Nov. 2018, pp. 1–2.

[18] "Introduction | Photon Engine." https://doc.photonengine.com/en-us/pun/v1/demos-and-tutorials/pun-basics-tutorial/intro (accessed Jan. 04, 2021).

[19] “GameSparks.” https://www.gamesparks.com/ (accessed Jan. 04, 2021).

[20] “Dedicated Game Server Hosting - Amazon GameLift - Amazon Web Services,” *Amazon Web Services, Inc.* https://aws.amazon.com/gamelift/ (accessed Jan. 04, 2021).

[21] A. G. Martínez, A. H. Orozco, C. F. Ramos, and M. Siller, “A Peer-to-Peer Architecture for Real-Time Distributed Visualization of 3D Collaborative Virtual Environments,” in *2009 13th IEEE/ACM International Sym. on Distributed Simulation and Real Time App.*, Oct. 2009, pp. 251–254.

[22] H. Suzuki and R. Huang, “Virtual real-time 3D object sharing for supporting distance education and training,” in *18th International Conference on AINA 2004.*, Mar. 2004, vol. 1, pp. 445-450.

[23] S. Rueda, P. Morillo, and J. M. Orduna, “A Peer-To-Peer platform for simulating distributed virtual environments,” in *2007 Intl. Conf. on Parallel and Distributed Systems*, Dec. 2007, pp. 1–8.

[24] J. Li, “Peer-to-peer multimedia applications,” in *Proceedings of the 14th ACM international conference on Multimedia*, Oct. 2006, pp. 3–6.

[25] Chen-Chi Kuo, J. Carter, and R. Kuramkote, “MP-LOCKs: replacing H/W synchronization primitives with message passing,” in *Proceedings Fifth International Symposium on High-Performance Computer Architecture*, Jan. 1999, pp. 284–288.

[26] H. Fan, H. Zhu, Q. Liu, Y. Shi, and C. Sun, “Shared-locking for semantic conflict prevention in real-time collaborative programming,” in *2017 IEEE 21st CSCWD*, Apr. 2017, pp. 174–179.

[27] “Game engine,” *Wikipedia*. Dec. 18, 2020, Accessed: Jan. 04, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Game_engine&oldid=994975794.

[28] “Best VR Game Engine Software 2020: Compare Reviews on 10+ Software,” *G2*. https://www.g2.com/categories/vr-game-engine (accessed Jan. 04, 2021).

[29] U. Technologies, “Unity - Manual: Unity User Manual.” https://docs.unity3d.com/Manual/index.html (accessed Jan. 04, 2021).

[30] U. Technologies, “Online and in-person courses & training in 2D, 3D, AR, & VR development | E-Learning.” https://unity.com/learn (accessed Jan. 04, 2021).

[31] “Unity XR platform updates - Unity Technologies Blog,” Jan. 24, 2020. https://blogs.unity3d.com/2020/01/24/unity-xr-platform-updates/ (accessed Jan. 04, 2021).

[32] G. Smith and K. Sung, “Teaching Computer Graphics Based on a Commercial Product,” 2019, doi: 10.2312/eged.20191031.

[33] “ARCore,” *Google Developers*. https://developers.google.com/ar (accessed Jan. 04, 2021).

[34] “AWS Lambda – Serverless Compute - Amazon Web Services,” *Amazon Web Services, Inc.* https://aws.amazon.com/lambda/ (accessed Jan. 04, 2021).

[35] Gregory Smith, *UWB-ARSandbox/ASL_NoMPsTutorials*. https://github.com/UWB-ARSandbox/ASL_NoMPsTutorials, 2020.

[36] “Projects | Cross Reality Collaboration Sandbox Research Group.” https://sites.uw.edu/crcs/projects/ (accessed Jan. 04, 2021).