FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Remote Collaboration over 3D Content in DeskVR

Nuno Miguel da Silva Alves

WORKING VERSION

Mestrado em Engenharia Informática e Computação

Supervisor: Daniel Filipe Martins Tavares Mendes

Second Supervisor: Rui Pedro Amaral Rodrigues

June 21, 2024

Remote Collaboration over 3D Content in DeskVR

Nuno Miguel da Silva Alves

Mestrado em Engenharia Informática e Computação

June 21, 2024

Abstract

The transition of real-world collaborative and teamwork tasks to the digital world has proven challenging, often resulting in the loss of the inherent social aspects of real-world collaboration. Research has addressed this gap by exploring social translucence – an approach to designing systems for social processes by implementing visibility, awareness, and accountability.

Virtual Reality (VR) is an excellent vector for incorporating the principles of social translucence. Due to its high sense of presence and immersion, VR inherently possesses enhanced social capabilities. A subset of VR of particular interest in this domain is DeskVR. DeskVR allows users to be fully immersed in a virtual environment while remaining seated at their desks. It offers a practical solution for extended working hours by reducing physical fatigue associated with standing and mid-air gestures.

Nevertheless, close interaction collaboration, such as the shared manipulation of an object, is complex due to the nature of the system. One fundamental challenge is the lack of awareness regarding other users' intentions. This absence contributes to the unpredictability of concurrent object manipulation, exacerbated by the lack of a physical link between users and the object. Furthermore, devising interaction techniques for DeskVR is difficult due to limited physical mobility and space.

This work proposes a method for implementing social awareness in the context of shared object manipulation in DeskVR. The proposed method involves designing a specialized multi-user interaction technique suited to the constraints of DeskVR. By designing with social translucence in mind, this technique helps users understand each other, fostering harmonious communication and collaboration. The research will also comprise an empirical user study to validate the effectiveness of this approach, evaluating the solution's usability and effectiveness.

Keywords: virtual environments, virtual reality, shared object manipulation, collaboration, social translucence, awareness, DeskVR

ACM Classification:

- Human-centered computing → Human computer interaction (HCI) → Interaction paradigms → Virtual reality
- Human-centered computing → Human computer interaction (HCI) → Interaction paradigms → Collaborative interaction

Acknowledgements

Nuno Alves

"It's a meaningless slab of iron you can't even lift,
for killin' dragons and monsters that ain't even real."

Godo, Berserk "He Who Hunts Dragons" by Kentaro Miura

Contents

List of Figures

List of Tables

Listings

Abbreviations

SA	Situation Awareness
CSCW	Computer-Supported Cooperative Work
VR	Virtual Reality
XR	Extended Reality
VE	Virtual Environment
CVE	Collaborative Virtual Environment
DOF	Degrees of Freedom
CAVE	Cave Automatic Virtual Environment
HMD	Head Mounted Display
WIM	World-in-Miniature
RPC	Remote Procedure Call

Chapter 1

Introduction

Throughout human history, collaboration has been an integral part of our pursuit of knowledge. Effective communication and a deep comprehension of each other have propelled us to advance our understanding and technological progress. In the digital age, it is only natural to harness the power of technology to extend collaboration to the digital realm, connecting people around the world. This is particularly pertinent in the aftermath of the COVID-19 pandemic, which has underscored the importance of digital tools in facilitating collaboration.

Virtual Reality (VR) emerges as a compelling platform for collaboration, offering a heightened sense of presence and immersion that inherently enhances social capabilities. A noteworthy subset of VR is DeskVR, immersing users in a virtual environment using a stereoscopic head-mounted display (HMD), all while comfortably seated at an office desk. DeskVR presents unique opportunities for interactions with the virtual environment, focusing on enhancing comfort, reducing physical fatigue, and improving accessibility to VR.

1.1 Context and Motivation

The transition of real-world collaborative tasks to the digital realm demands careful consideration and a thorough understanding of how we communicate. Often, certain tasks become more challenging in the digital space, leading users to prefer and be more efficient in performing those tasks in the real world. During this transition, social information tends to become opaque, limiting users' understanding of each other [16].

VR is an excellent medium for collaboration since users can coexist in a virtual space, visually observing each other's actions and communicating through voice. Still, designing for awareness in VR presents some challenges, particularly with interfaces and information presentation. For example, reading text can be difficult in VR [43, 26]. Nonetheless, advancements in resolution and HMD technology, such as eye gaze tracking, are addressing this issue.

Using VR while standing and relying on mid-air interaction for extended periods can induce fatigue and may be less accessible for individuals with mobility impairments. DeskVR addresses this concern, allowing users to interact with the environment while seated. The use of an office

desk offers additional benefits, serving as a surface for touch-based approaches, providing passive haptic feedback, and serving as a virtual representation for presenting social information or operational instructions [65, 56]. However, constraining users to a seated position limits their movement, requiring special attention while designing for DeskVR.

For instance, one of the most common interactions in VR is object manipulation, which often relies on techniques designed for users who are standing and engaging in physically demanding mid-air movements. Given that DeskVR users are confined to a desk, their mobility and range for mid-air movements are limited. Therefore, it is important to design object interaction techniques considering range, physical demand, and ergonomics for seated users [2].

Collaboration can be used to help object manipulation. For example, handling large objects in VR is difficult as they may obscure the user's view during positioning tasks, often requiring users to place and inspect the object from alternative angles [7]. A collaboration involving multiple users can be advantageous in these situations, as diverse viewpoints contribute to a better understanding of the environment, assisting the primary user in successfully manipulating the object [42]. The challenge lies in seamlessly integrating efficient collaboration and social awareness within the limitations of DeskVR.

1.2 Objectives

The goal of this work is to design and implement a collaborative framework for VR that seamlessly integrates efficient collaboration and awareness within DeskVR. This will involve a comprehensive examination of the current state-of-the-art techniques in collaboration, computer-supported cooperative work (CSCW), and DeskVR. The objective was to discern the difficulties and challenges inherent in these areas, hoping to identify a promising avenue for exploring collaboration. Subsequently, this framework will undergo user evaluation to analyze its effectiveness as a tool for communication and cooperation.

1.3 Document Structure

Chapter 2, *Related Work*, explores the current state-of-the-art relevant to this research. It starts with an investigation into computer-supported cooperative work, then examines techniques employed to maintain consistency in shared virtual environments, and subsequently delves into DeskVR interaction techniques. The chapter ends with a discussion of the examined work, relating the concepts between various topics and offering insights into the design of the proposed solution.

Chapter 3, *Design of the Proposed Method*, presents a high-level overview of the design of the proposed solution, offering detailed specifications and descriptions for each component necessary for the solution's implementation.

Chapter 4, *Implementation of a Prototype*, showcases the concrete implementation following the design standards established in Chapter 3. It details the technologies and hardware used, as well as delving into more technical aspects.

Chapter 5, *Evaluation*, outlines the testing methodology and the subsequent evaluation of the obtained results.

Finally, Chapter 6, *Conclusions*, brings this work to a close by revisiting key points, suggesting potential future research directions, and providing concluding remarks.

Chapter 2

Literature Review

This chapter explores the literature on virtual reality and collaboration. Section 2.1 delves into computer-supported cooperative work and describes social translucence, concepts of workspace awareness, and group spatial interactions. Section 2.2 explores methods to ensure consistency in shared virtual environments across multiple users. Section 2.3 explores various interaction techniques used in DeskVR research. Lastly, section 2.4 assesses the techniques' applicability for a DeskVR environment, providing insights that inform the design of the proposed solution.

2.1 Computer-Supported Cooperative Work

The transition of real-world collaborative tasks to the digital world has rendered social information invisible, leading to challenges in effective communication and collaboration [16]. This section explores approaches to designing digital systems to enhance collaboration and information visibility by addressing these challenges.

2.1.1 Social Translucence

Erickson and Kellogg [16] define *socially translucent systems* as those that facilitate coherent behavior in human-to-human communication by making participants and their activities visible to one another. They believe that social information provides the basis for inferences, planning, and coordination of activities. They describe three properties of these systems: *visibility* of socially significant information, *awareness* of what is happening, and *accountability* for users' actions. Interestingly, they denote that while accountability and awareness are typically correlated in the physical world, they may not necessarily coincide in the digital realm.

The rationale for using the term *translucent*, rather than *transparent*, is that in the real world, social information is not entirely transparent due to various constraints. For example, physical distances between conversing groups prevent each group from hearing the other's conversation. Erickson and Kellog argue that there is power in these constraints, shaping our words and actions based on the audience present or its size. In digital environments, it is hard to understand the constraints or whether they are even shared, such as muting someone on a chat room without

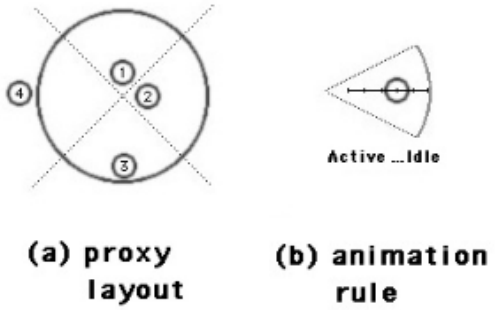


Figure 2.1: The illustration on the left shows the layout of the proxy in Babble [16], where dots 1, 2, and 3 are part of the conversation, and 4 is in another. The illustration on the right shows how the dots animate and drift further away depending on their activity level.

their knowledge or instances of shadow-banning. They say that the shared awareness of these constraints is critical in structuring our interactions, so they highlight the importance of supporting this in the digital realm.

Moreover, in the same article [16], they discuss how social translucence may be implemented in practice. First, they explore ways to make individuals' activity visible. For this purpose, they describe three designs: a realistic approach – projecting real-world social information to the virtual world through means such as video conferencing; a mimetic approach – creating a representation of social information in the digital world as closely as possible, such as avatars; and an abstract approach – portraying social information in the digital domain through abstract representations.

As a result of this discussion, they [16] chose the abstract method for further analysis. This decision stems from the apparent characteristics of these representations using text and simple graphics: they are easy to create, persist over time, and can be easily found using search and visualization engines.

Consequently, Erickson and Kellog describe a prototype called Babble that uses abstract representations to portray social information. Babble illustrates social communication using two approaches: a persistent textual representation that displays comments, participants' names, and timestamps; and a synchronous visual representation titled *social proxy*, which shows conversations as a circle, where participants are dots inside that circle, and their closeness to the center represents their activity level, as shown in Figure 2.1. With this, they show the efficacy and the importance of abstract representations in portraying social information.

2.1.2 Workspace Awareness

Endsley [15] defines *awareness* as "knowing what is going on." Several traits have emerged in previous studies on awareness [1, 38, 15, 22]: Awareness is the knowledge about the state of the environment in both space and time. Since environments change over time, maintaining and updating this knowledge is part of awareness. Individuals can achieve this by interacting with and

exploring the environment. Finally, awareness is not the primary goal of a task-oriented system but rather a secondary goal, the primary objective being to complete the task.

Situation Awareness (SA) is a concept that arose in research about more dynamic and complex environments with high information load, variable workload, and risk [19]. Adams et al. [1] define SA as "the up-to-the-minute cognizance required to operate or maintain a system." Endsley [15] describes a three-stage definition of SA that focuses more on the process. The first stage is the perception of relevant elements in the environment. The second stage is the comprehension of the current situation, or the ability to relate perceptual information retrieved in the first stage with past knowledge to understand their meaning. The third and last stage is the forecast of the status of those elements, which is invaluable for decision-making.

Gutwin and Greenberg [22] define the concept of workspace awareness as "the up-to-the-moment understanding of another person's interaction with the shared workspace" [22]. It is a specialized kind of SA due to the shared workspace context – it is about being aware not only of the environment but also of other individuals and their interactions with that environment. Similarly to Erickson and Kellog [16], they believe the difficulty of maintaining awareness in collaborative tasks in digital systems lies in the technological constraints of the medium, the lack of information due to the limited perception of others, and the misrepresentation of that information.

To aid the understanding of concepts and purposes of designing workspace awareness in groupware systems, Gutwin and Greenberg [22] developed a three-part conceptual framework. The first part focuses on what information is used and what is necessary for workspace awareness. The second part is about how that information is gathered. Lastly, the third part concerns the application of workspace awareness in collaboration.

Previous research [14, 55, 34, 22] explore a set of workspace information that people keep track of during collaborative work, which is the elements that answer the questions "who, what, where, when, and how." Gutwin and Greenberg [22] present specific elements and questions answered by those elements according to what they believe is the core of workspace awareness, seen summarized on tables 2.1 and 2.2.

Earlier findings [51, 38, 12, 25, 22] propose three primary sources and mechanisms for gathering workspace information: through people's bodies and consequential communication, through workspace artifacts and feedthrough, and through conversation, gestures, and intentional communication. People's bodies provide extensive details about their current work, making watching and hearing other people a mechanism for obtaining workspace information [22, 51]. This mechanism for gathering information is called *consequential communication* [51].

The second source of information is the artifacts produced in the workspace [12, 20]. The information that can be collected from these artifacts can be obtained either through visible characteristics [22] or through the sound that is made when manipulating them [20]. The mechanism used for gathering this information is called *feedthrough* [12], meaning that when manipulating an artifact, information is received by not only the person doing the action as a form of feedback but also by others who are watching.

Table 2.1: Elements of workspace awareness of the present [22]

Category	Element	Questions
Who	Presence	Is anyone in the workspace?
	Identity	Who is participating? Who is that?
	Authorship	Who is doing that?
What	Action	What are they doing?
	Intention	What goal is that action part of?
	Artifact	What object are they working on?
Where	Location	Where are they working?
	Gaze	Where are they looking?
	View	Where can they see?
	Reach	Where can they reach?

Table 2.2: Elements of workspace awareness of the past [22]

Category	Element	Questions
How	Action history	How did that operation happen?
	Artifact history	How did this artifact come to be in this state?
When	Event history	When did that event happen?
Who	Presence history	Who was here, and when?
Where	Location history	Where has a person been?
What	Action history	What has a person been doing?

The third source of workspace information is conversation and gesture through intentional communication [9, 24, 6]. People can gather information from verbal communication in different ways: through explicit conversational exchange of awareness elements [22], by picking up relevant information from other peoples' conversations [22], or through *verbal shadowing* or "outlouds" [24] which are comments made by individuals while working that are addressed to no one in particular.

Gutwin and Greenberg [22] suggest five types of activities that benefit from workspace awareness. The identified types include: the management of coupling – the degree to which people are working together [50] or the amount of work a person can do before requiring the help of another [22]; simplification of communication – allowing using deictic references, demonstrations, and visual evidence; coordination of actions; anticipation; and assistance.

2.1.3 Spatial Group Interaction

Unlike conventional CSCW settings, real-time collaboration in virtual environments introduces novel interaction possibilities, particularly when multiple users simultaneously engage with one or more objects [8]. Therefore, it becomes essential to acknowledge the importance of enhancing awareness in this specific context.

A spatial approach to collaborative virtual environments consists of spaces or rooms allowing individuals to navigate and engage with one another and the objects, or artifacts, within these designated areas [4]. With the goal of enhancing collaboration and scalability within large virtual environments, Benford and Falhén [4] propose the spatial model of interaction.

The spatial interaction model introduces the concepts of medium, aura, awareness, focus, nimbus, and adapters. The medium is the means through which interactions with objects occur, for example, through text, audio, or visuals. The aura represents an area in which objects can interact within a given medium. When two auras collide, the interaction between the objects in the medium becomes possible. Objects can have multiple auras, such as their size, shape, and color.

Awareness is the basis on which objects control their potential interactions, as determined by their auras. This awareness is not necessarily symmetrical. For example, between object A and object B, object A can be more aware of B than B is aware of A. The awareness of an object towards another is calculated based on the combination of its focus and nimbus. Focus and nimbus are, like auras, subspaces of an object and describe its attention or presence, respectively. Benford and Falhén [4] describe this as "the more an object is within your focus, the more aware you are of it and the more an object is within your nimbus, the more aware it is of you".

Furthermore, Benford and Falhén [4] denote that a person does not need to be aware of their aura, focus, and nimbus, as these may be manipulated using natural interactions. Notably, they indicate three main ways of managing these subspaces. The first is through implicit interaction derived from, for example, the movement, orientation, or eye gaze of individuals. The second is through explicit adjustment of parameters, for instance, with the help of a user interface. The third is through adapters which modify a user's aura, focus, and nimbus. An example of an adapter would be a tool that a person can pick up or a table in which a user can sit, altering the user's aura, focus, and nimbus for that context.

Domingues et al. [13] develop a workflow-based approach to collaborative 3D interaction based on the spatial model of interactions. They describe the workflow concept, which manages tasks and ensures coordination in collaborative environments. The workflow comprises two components: the shared component, encompassing the data and state information of all users and sources within the environment; and the motor component, which uses the data from the shared component to execute actions on particular sources through assistance functions. Here, a *source* is an object that users can perceive, *particular sources* are objects that change during 3D interaction through assistance functions, serving as support tools, and *assistance functions* help manage coordination in 3D interaction.

Next, they specify five particular sources, dedicated to interaction in the virtual environment.

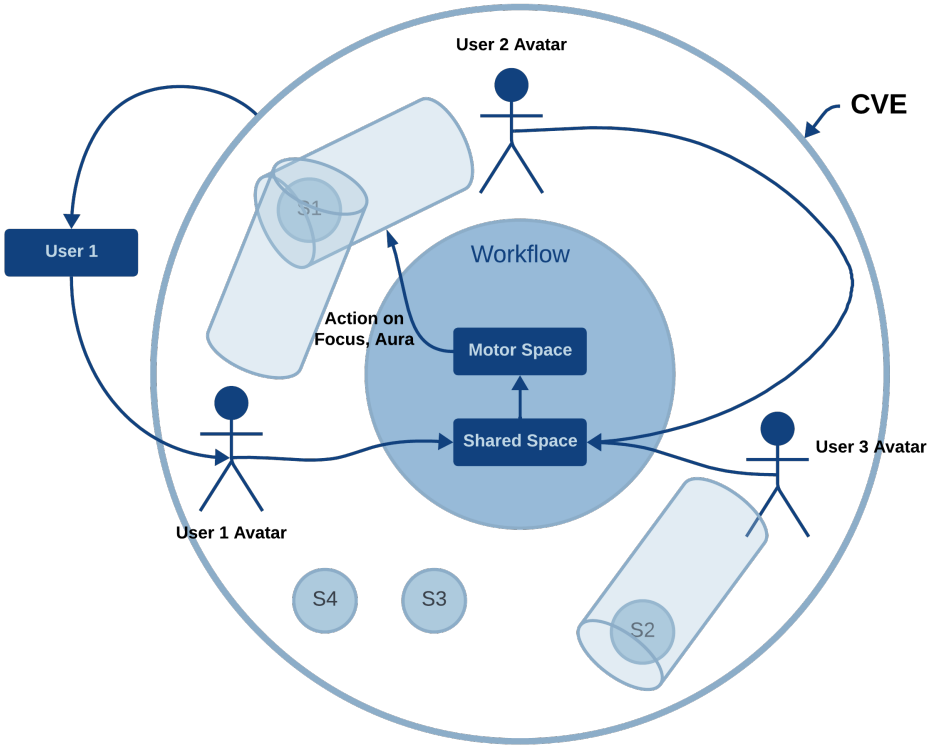


Figure 2.2: Portrayal adapted from the work of Domingues et al. [13]. This represents their workflow approach applied to a CVE, where cylinders represent user foci. In this example, the nimbus of the object S1 is the set of users 1 and 2.

3DIFocus coincides with objects the user can interact with, meaning in their field of view. *3DINimbus* symbolizes all users that might interact with this source. *3DIAura* is a zone that enables specific interactions if users are situated within its area. *3DIAssistant* assists users during specific actions through a virtual guide. Lastly, *3DIAvatar* is the digital counterpart of a user. These concepts are illustrated in figure 2.2

Using this model, Domingues et al. [13] describe several assistance functions, in particular, an assistance function for helping the coordination of multi-user interaction with an object. They model multi-user interaction by inserting joints between the user avatars and the object, allowing for manipulation of that object using single-user interaction techniques. The assistance function then helps manage this interaction by creating visual guides for the users by changing the color of the avatars engaged in multi-user interaction, or by displaying the object's direction, for instance.

2.2 Concurrency Control

Margery et al. [33] developed a classification to assess the cooperation levels in multi-user systems. In systems with level 1 collaboration, users can perceive and communicate with each other in the virtual world through avatars. Level 2 systems enable users to manipulate objects within the

scene. Level 3 systems allow users to manipulate the same object simultaneously. Level 3 can be further subdivided based on the degree of independence in users' actions, such as independently changing different aspects of an object or combining inputs codependently. This last collaborative level, particularly the codependent mode, is considered the most natural and immersive form of collaboration, offering enhanced efficiency.

Broll [8] had previously defined something similar to levels 2 and 3, cooperative and collaborative interaction, respectively. Recognizing which interactions are concurrent and which are not is essential yet challenging. Without a notion of simultaneity, interactions become a series of individual moves by each user rather than a coordinated combination. In distributed environments, typical in collaborative virtual reality, where each user has a copy of the world, determining concurrency is complicated, especially considering communication delays.

Concurrency control prevents conflicts of concurrent updates arising from adverse network conditions. In its absence, users manipulating shared objects in conflicting directions may experience difficulty anticipating outcomes and encounter divergence of state [44], introducing confusion among participants, offering disparate views of what should be a shared world and, in turn, violating the essential consistency requirement [59]. Therefore, the study of concurrency control emerges as an essential research topic for developing systems with level 3 collaboration.

The following subsections describe three distinct approaches for implementing concurrency control. Object ownership methods use locking mechanisms to block users from interacting with an object or its properties unless they possess a corresponding lock or ownership. Attribute separation methods assign specific attributes or degrees of freedom of an object to individual users, making each user responsible for that attribute. Finally, distributed average techniques calculate the average of participants' actions, integrating and updating them distributively.

2.2.1 Object Ownership Techniques

Greenberg and Marwood [21] describe how traditional concurrency control strategies can be used in real-time groupware systems. In particular, they demonstrate how the method of privileged access through locking can be applied in these systems, describing two different approaches. A non-optimistic locking policy blocks users from interacting with an object until a lock has been granted, signifying ownership over the object. An optimistic locking policy allows users to manipulate an object before they know if a lock has been granted to them. If the lock has been denied, the object returns to its original state, and a repair must be done. Optimistic locking schemes can be further divided into two approaches. A fully-optimistic approach allows users to manipulate multiple objects with tentative locks, while a semi-optimistic approach only allows users to manipulate one object at a time.

BrickNet [54] is a toolkit for network-based virtual environments. It uses a pessimistic locking mechanism for object manipulation where objects can only be updated by the current owner. Users have to request ownership over an object to the server, and the server mediates object updates. DIVE [23] is a multi-user distributed virtual environment system. DIVE also employs a pessimistic locking mechanism, similar to BrickNet, where users require an object-based token

to interact with the object. Spline [62] is a software platform for developing distributed virtual environments that implements the Interactive Sharing Transfer Protocol, or ISTP [61]. Objects in the ISTP protocol, like DIVE, are only subject to changes from its owner process. This ownership can be transferred between processes. CAVERNsoft [29] is a collaborative software architecture that was used by the CAVE Research Network, which also uses the pessimistic locking approach.

The PaRADE collaborative environment [47, 44] takes a different approach to pessimistic locking by employing a predictive strategy. Developed to mitigate adverse network effects, PaRADE aims to enhance both responsiveness and consistency through advanced time management. In PaRADE, all events are timestamped with both wall-clock time and causal time, and efforts are made to predict events to reduce the impact of network delays. Combining these times is used for sufficient causal ordering, serving as a middle ground for Lamport's causal ordering [27] to enhance concurrency and responsiveness. This approach restricts changes to an object to originate from a single user, leading to the adoption of a predictive object transfer paradigm that anticipates entity ownership in advance.

The predictive object ownership transfer relies on application knowledge and heuristics to anticipate a user's intention to interact with an object, enabling the transfer in advance of the ownership to minimize the impact of network delays. This advance transfer is important for optimizing responsiveness and in scenarios where ownership may have been erroneously transferred, allowing for retransmission to the correct recipient. Yang and Lee [63] identified scalability challenges in the widespread dissemination of token requests to all users in the network. In response, they introduced the concept of an entity radius, wherein messages are selectively delivered only to individuals within this radius, forming an "entity-centric multicast group."

Pessimistic approaches for locking mechanisms help preserve data integrity in databases, for example. However, in settings such as collaborative virtual environments, not only does obtaining a lock necessitate user waiting, but ensuring sequential updates becomes less scalable with a growing number of users. Sung et al. introduce CIAO [59], a large-scale 3D layout system that uses a semi-optimistic policy for managing shared object manipulation to enhance responsiveness. Unlike traditional systems, users in CIAO can manipulate objects without waiting for a lock. To address potential confusion arising from multiple users interacting with an object, the CIAO interface incorporates awareness features, using translucent clones to indicate ongoing manipulations yet to be validated. The authors also address the challenges of concurrently interacting with hierarchically related objects.

2.2.2 Attribute Separation Techniques

The approaches discussed in the previous section are limited to a collaboration level of 2, as there is no stage where multiple users synchronize their inputs on a shared object. Additionally, those concurrency control mechanisms often introduce issues commonly called "surprise" [30]. Such surprises manifest as conflicts leading to the reversal of changes or unexpected alterations in the anticipated order of events due to concurrency control conflicts.

This section introduces strategies that elevate collaboration to level 3, enabling independent modifications of various attributes of an object simultaneously. The concept involves assigning ownership of specific attributes to individual users, allowing multiple users to collaboratively manipulate an object without needing sequential ownership.

In their work, Lee et al. [28] refer to the manipulation of an attribute of a shared object as a "task." However, they explain the importance of avoiding "surprises" or, in this context, "task-surprises." These surprises may arise when interruptions occur during user tasks by other tasks or when task dependencies result in unexpected states when executed concurrently.

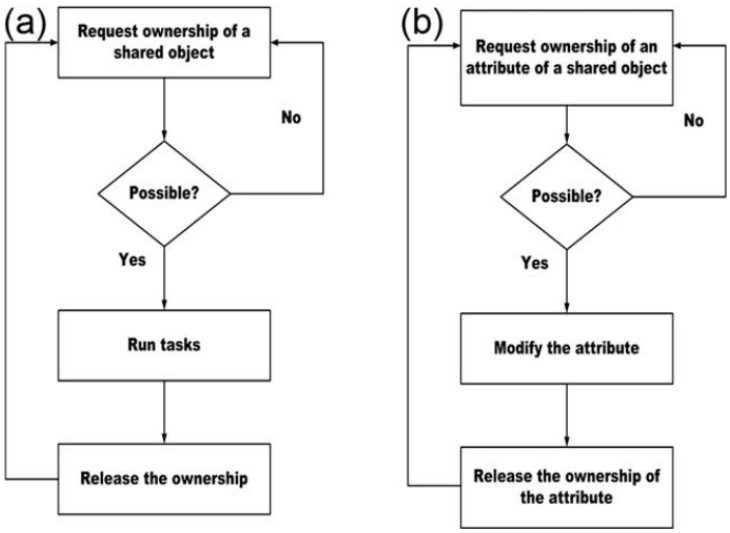


Figure 2.3: Illustration depicting the difference between ownership transfer and attribute separation [28].

Advancements in internet infrastructure around the turn of the century, particularly with the establishment of Internet2 [64] – a robust network implemented in the United States in 1996 for academic purposes – sparked new interest in exploring collaborative tasks in remote environments. Mortensen et al. [36] aimed to assess the feasibility of joint tasks when two users, separated by thousands of miles, interacted in a shared virtual environment within DIVE. The specific task involved both users lifting a stretcher together and moving it along a predefined path into a building. To achieve this, two handles were added to the stretcher for the users to manipulate, with the stretcher aligning itself based on the position and orientation of these handles. Their findings revealed that while data transmission speeds and throughput were sufficient, the lack of software support, particularly concerning lost data packets impacting consistency, posed a significant challenge.

Roberts et al. [45] investigated two distinct methods of shared object manipulation and how these methods were influenced by the display devices their users employed: an immersive walk-in display with tracked head and hands, and a desktop application. The two methods explored were sharing the same attribute and sharing through different attributes. To conduct this study,

they devised an experiment where users collaborated on constructing a gazebo in DIVE. Some sub-tasks required users to collaborate using the same attribute, such as jointly carrying a beam, while other tasks involved simultaneous manipulation of the object for different purposes, such as one person inserting a screw while another held the beam in place. This study also concluded that technological advancements had made such collaborative tasks feasible, but the ownership mechanism of DIVE failed to provide effective collaboration [46]. As such, it became evident that new systems needed to be developed with level 3 collaboration in mind.

Pinho et al. [41, 42] proposed a method for collaborative object interaction by separating degrees of freedom (DoF) and assigning them to different users. A degree of freedom in this context refers to a degree of control over the movement of an object, such as horizontal translation or rotation around an axis. As previously mentioned in section 2.3, the separation of DoF can be beneficial when precision is an important factor.

The goal of this approach was to use single-user interaction techniques, such as HOMER [7, 37], in collaborative manipulation, with the reasoning that they were well-researched, commonly understood, and perceived as more "magical" than natural, which was prevalent in most collaborative manipulation techniques then. The objective was to extend users' capabilities for multi-user object interaction. Additionally, the authors focused on awareness by implementing features like using a pointer to indicate the selected object, changing the object's color based on its state, and representing the DoF graphically, as shown in Figure 2.4. While the results were positive, two challenges were identified: testing the applicability with more than two users, and the selection of DoF for each user was predetermined in a configuration, limiting flexibility.

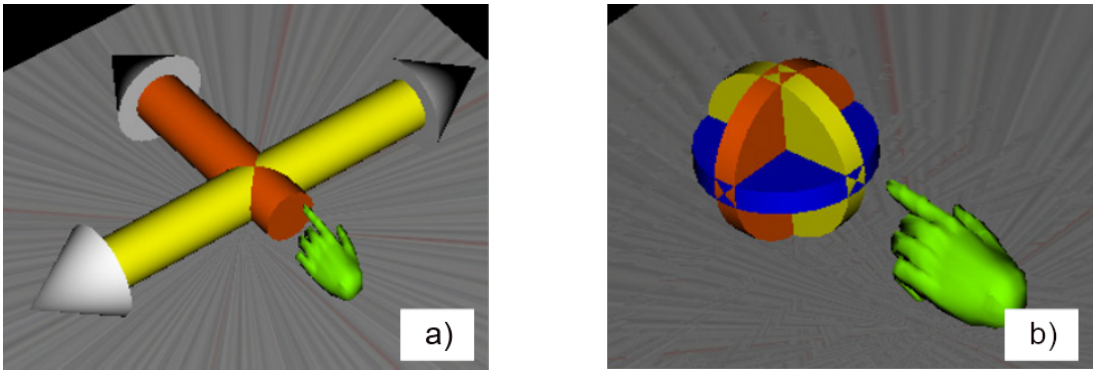


Figure 2.4: Illustration of translation (a) and rotation (b) pointers in [42]

Lee et al. [28] proposed a concurrency control mechanism based on fine-grained tasks. In this approach, if a user who does not own an object requests a task to manipulate a shared object, the task is permitted if it avoids conflicts with the owner's task and does not lead to task surprises. If the task is not allowed, non-owners can perform their tasks with duplicates of the object on a personal workspace, similar to the ghost images in CIAO [59]. The outcomes of these modifications in personal workspaces can be stored, allowing users to discuss and determine the final state of the object. Figure 2.5 demonstrates the flow of the concurrency control process.

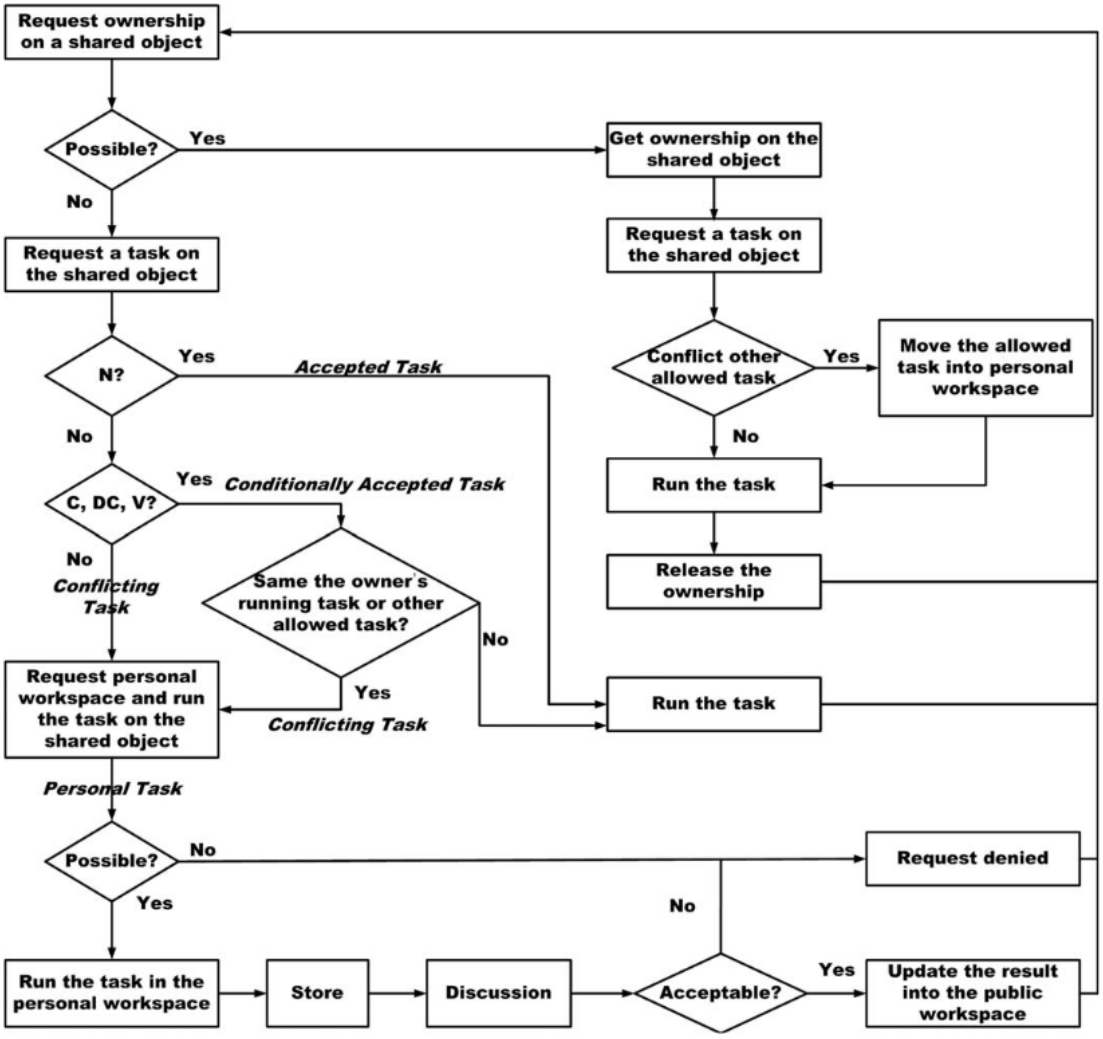


Figure 2.5: Illustration of the fine-grained task concurrency control system by Lee et al. [28].

The allowance types for each task classification can be seen on Table 2.3. *Conflicting tasks* are exclusive to the object's owner. *Conditionally accepted tasks* can be executed only if no one else is performing the task, while *accepted tasks* can always be executed. While their implementation demonstrated slower efficiency compared to other attribute-sharing mechanisms, Lee et al.'s approach reduced the occurrence of task surprises.

2.2.3 Distributed Average Techniques

This section explores techniques associated with codependent level 3 collaboration. Integrating level 3 collaboration into CVEs is challenging, especially in regards to the complexity of combining and integrating actions from multiple users, the absence of feedback inherent in physical interactions with an object, and network communication issues, especially concerning latency, which significantly influences the ability to detect simultaneous actions [48, 8].

Table 2.3: Allowance type for each task classification by Lee et al. [28].

Classification of Tasks	Allowance Type
Translation	Conflicting task
Rotation	Conflicting task
Scaling	Conflicting task
Fission	Conflicting task
Merge	Conflicting task
New	Accepted task
Remove	Conflicting task
Copy	Conditionally accepted task
Data calculation	Conditionally accepted task
Visual property	Conditionally accepted task

Ruddle et al. [48] investigated whether an asymmetric or symmetric approach was better suited for level 3 collaboration in the piano mover's task. This task involves two users maneuvering a large virtual object through confined spaces. Symmetric manipulation requires coordinated actions in direction and intensity, while asymmetric manipulation allows users to interact with the object differently, considering the average of the interactions. To address latency concerns, experiments were conducted on a single host computer. Interestingly, the results did not reveal a significant difference in the time participants took to complete the task, but instead, that performance depended on the task. However, the study highlighted the challenge of coordinating participants' movements, emphasizing the need for feedback, whether haptic or otherwise.

Friston et al. [18] present a concurrency control framework designed to enable the integration of level 3 collaboration in distributed virtual environments. This approach views the collaborative environment as a distributed data-fusion problem and incorporates elements such as prediction, distributed averaging, continuous authority, and constraint duplication. The methodology is rooted in consensus-based networking, where simulations share state updates with other nodes, and local solvers integrate these updates to establish a distributed average consensus of the system's state, ensuring consistency over time. Unlike bilateral operation or force-reflection systems, where clients submit inputs to a central authority, this approach calculates actions distributively, resulting in improved scalability. The authors conducted experiments that demonstrated the effectiveness of this technique in preserving causality, stacking objects, enabling level 3 collaboration, and providing support for haptic feedback.

2.3 DeskVR Interaction

Before the widespread adoption of stereoscopic displays, various effective approaches emerged for utilizing a multi-touch tabletop surface to interact with 3D objects with minimal fatigue [5, 58, 10]. Balloon Selection [5] employs the metaphor of manipulating a balloon attached to a string. This

approach divides a 3DOF positioning task into a 2DOF positioning task with one finger on the touch surface and a 1DOF string-pulling task with the other finger to control the height of the balloon. Pulling the fingers apart decreases the balloon's height and bringing them together increases its height. Balloon selection does not differentiate between right and left hands; instead, it prioritizes the order in which fingers are placed on the surface, designating the primary finger (anchor), secondary finger (stretching finger), and tertiary finger (selection finger). This design allows both right-handed and left-handed users to use the technique effortlessly. Additionally, the stretching finger may not always be held, allowing the balloon's height to remain fixed when removed. This feature, referred to as "String Height Clutching" by the authors, enables the extension of the balloon's height infinitely. This technique has shown low error rates, likely due to the user's hands being supported by the tabletop surface, resulting in significantly reduced hand tremor and arm fatigue.

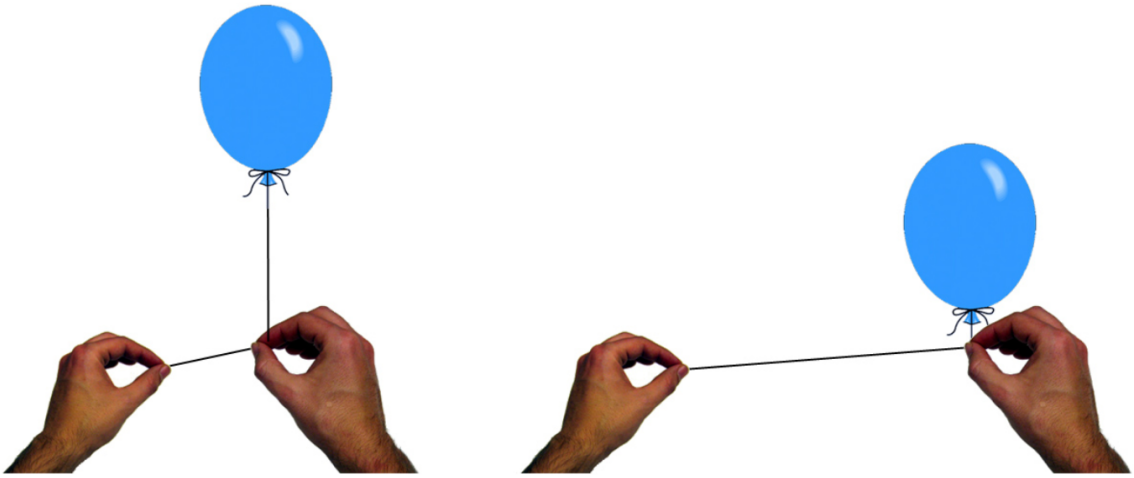


Figure 2.6: Illustration of the Balloon Selection metaphor [5].

Corkscrew Selection [5], similar to Balloon Selection, also breaks down the 3DOF task into a 2DOF positioning task and a 1DOF height task. However, it differs in the method used to adjust the height of the selection point, achieved through a rotational motion around a selection widget.

Triangle Cursor [58], on the other hand, creates an isosceles triangle with its two base vertices positioned at the touch points of two fingers, while the altitude's base point is located at the midpoint. Manipulating the triangle's position involves moving the fingers on the surface, while adjusting its height is controlled by scaling the triangle simlarly to a typical pinch gesture based on the distance between the two fingers. The height's upper limit is constrained by the diagonal of the touch surface. Additionally, users can initiate yaw rotation around an axis perpendicular to the table's surface by rotating the fingers around the midpoint.

In their evaluation comparing Triangle Cursor to Balloon Selection, the authors noted a slight preference in terms of speed and error minimization for Triangle Cursor. However, it's worth mentioning that the tasks necessitated the use of rotational techniques, which required an extension of Balloon Selection by using the rotation of the secondary finger around the primary finger as

rotational input.

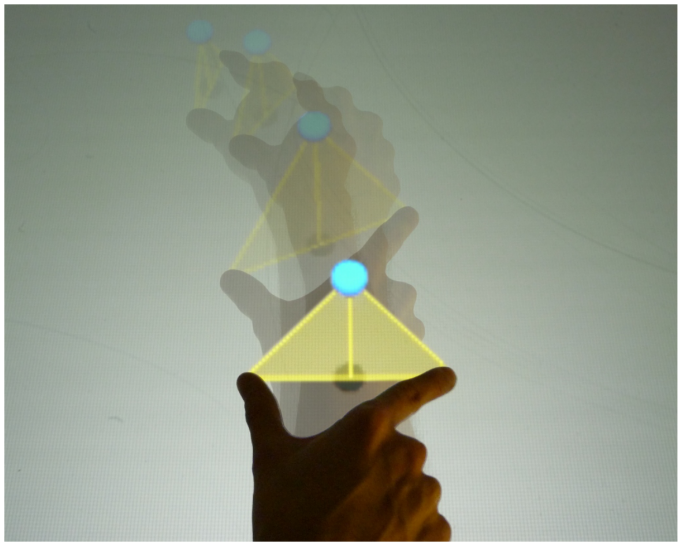


Figure 2.7: Illustration of the Triangle Cursor technique [58].

Zielasko et al. [67] introduced the concept of DeskVR, a fully immersive VR experience accessed entirely from an office desk while the user remains seated. Initially conceived to integrate with the workflow of data analysts, the application of this concept extends to broader applications, aiming to alleviate physical strain associated with standing, extend work periods, enhance accessibility, and boost overall productivity.

Zielasko et al. [66] surveyed to evaluate the advantages and disadvantages of standing versus sitting and the degree of embodiment in virtual reality experiences. The findings indicated that sitting generally scored higher in reducing cybersickness, enhancing comfort, ensuring safety, and improving accessibility. On the other hand, standing was preferred for perceived self-motion, locomotion preciseness, and engagement.

Given the constraints of DeskVR, environmental interactions must be re-evaluated, especially regarding selection, manipulation, and travel techniques. Because users remain seated in DeskVR, this enables interactions not common in typical VR scenarios, such as touch-based interactions on a desk. Zielasko et al. [65] explored this concept through the evaluation of four menu interaction arrangements in DeskVR: "Desk" aligns menus with the virtual desk, "Air" aligns menus with the user's task, and "DeskPlus" and "AirPlus" combine the previous scenarios with a physical desk to provide passive haptics. These configurations can be seen in Figures 2.8 and 2.9.

The results revealed diverse individual preferences for menu configurations. Some favored desk-aligned menus for tangibility, while others found it odd to touch a menu expected to be virtual. Additionally, menus aligned with the virtual desk required more head movement, making them less efficient. However, variants with a physical desk were favored for reducing physical strain.

Sousa et al. [56] introduced VRRRRoom, a virtual reality radiology reading room with a desktop surface for interacting with medical images. Medical images are displayed in 3D above a

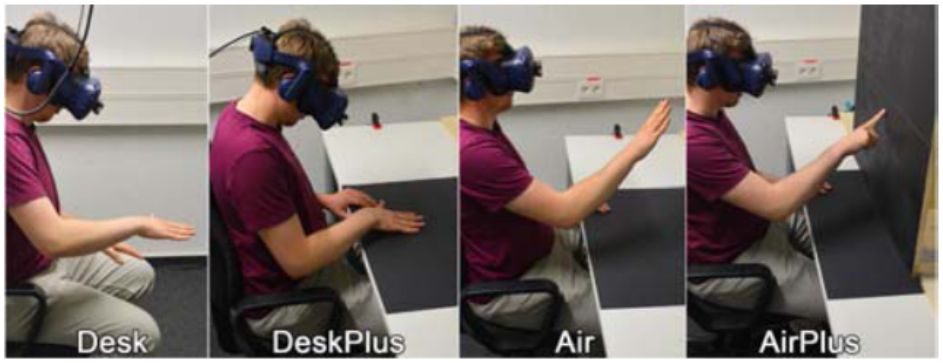


Figure 2.8: The four different scenarios studied in [65]. The "Desk" scenario aligns the menu with a virtual desk, while the "Air" scenario aligns the menu with the task. The "DeskPlus" scenario aligns the virtual desk with a physical desk, while "AirPlus" aligns the menu with the taks and a vertical board.

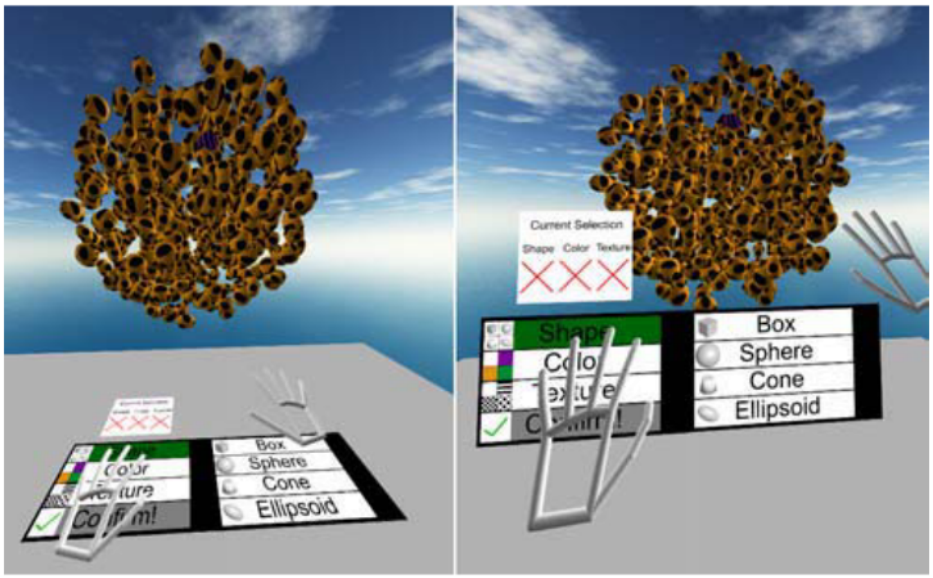


Figure 2.9: Experimental settings in [65] for desk-aligned menus on the left and task-aligned menus on the right.

virtual desk and can be manipulated using indirect touch controls. Users can change volume slices and adjust brightness with their left hand, while rotating the image and changing the scale can be done with their right hand, as depicted in Figure 2.10. The gesture-based interaction minimizes the need for users to move their head to view controls, as illustrated in Figure 2.11.

The visualization of controls on a virtual desk, exemplified in VRRRRoom [56] and illustrated in Figure 2.11, serves as a valuable tool for instructing users about available gestures and conveying information such as the current volume slice. Zielasko et al. [68] conducted an experiment to assess the impact of introducing this virtual desk into the virtual environment, analyzing its effects on performance, cybersickness, and presence. Their findings indicated no significant dif-

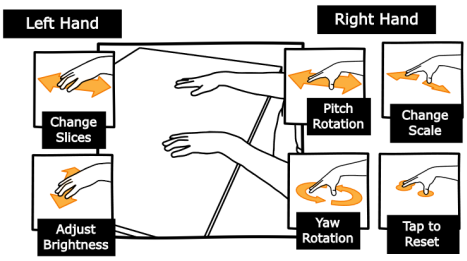


Figure 2.10: Gesture dictionary in [56]

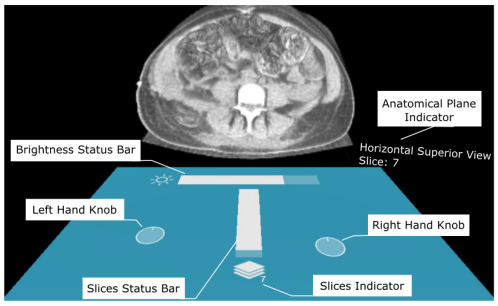


Figure 2.11: Virtual desk, control indicators, and medical images rendering in [56]

ferences in the evaluated aspects, suggesting the potential for expanding the desk functionality to incorporate menus or controls.

Regarding the selection and manipulation of 3D objects, many techniques are viable in both standing and sitting positions. However, techniques requiring extra controllers may be less suitable, as finding them after being set down while immersed in an HMD can be difficult [67]. Almeida et al. [2] developed a DeskVR-specific object manipulation technique called SIT6. This indirect touch-based technique utilizes a gesture dictionary with separated degrees of freedom: three for translation and three for rotation, as seen in Figure 2.12. Indirect touch means that users do not directly touch the object through a screen display but interact with it indirectly, as the former would not be practical in VR using an HMD [35]. While SIT6 may not be as fast as other state-of-the-art mid-air techniques, it was found to be as effective and less physically demanding.

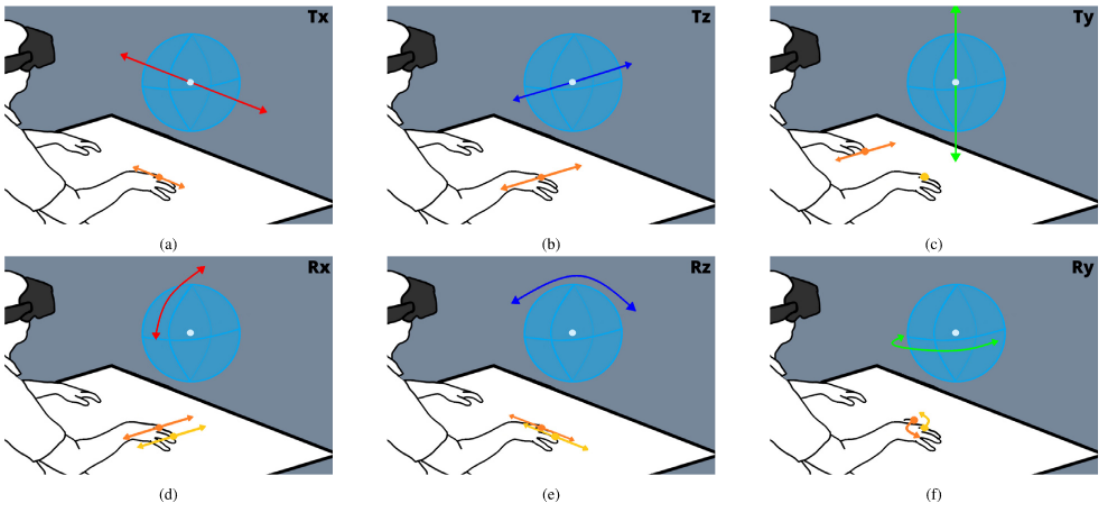
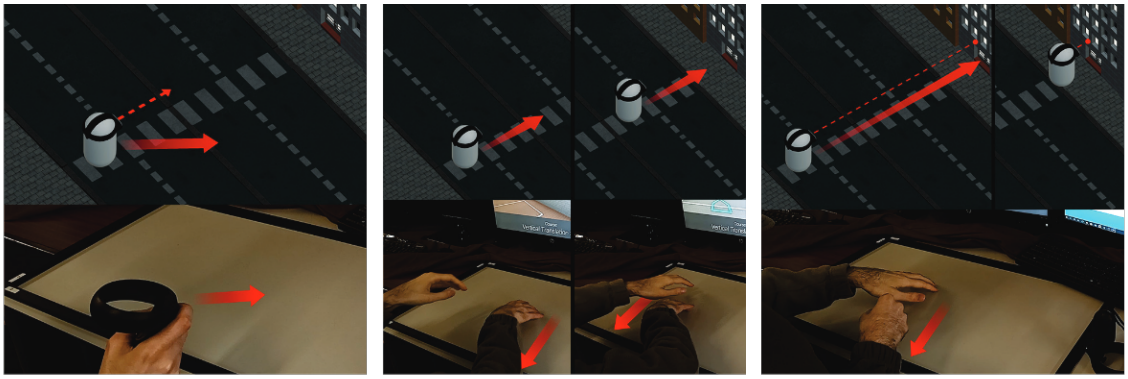


Figure 2.12: The gesture dictionary of SIT6 [2]. Gestures (a), (b), and (c) represent translation movements, while gestures (d), (e), and (f) represent rotation movements.

Designing traveling techniques in DeskVR is challenging because users are seated, rendering real walking impractical despite its potential presence enhancement. While presence is crucial for reducing cybersickness [67], an effective travel technique in DeskVR should aim to enhance

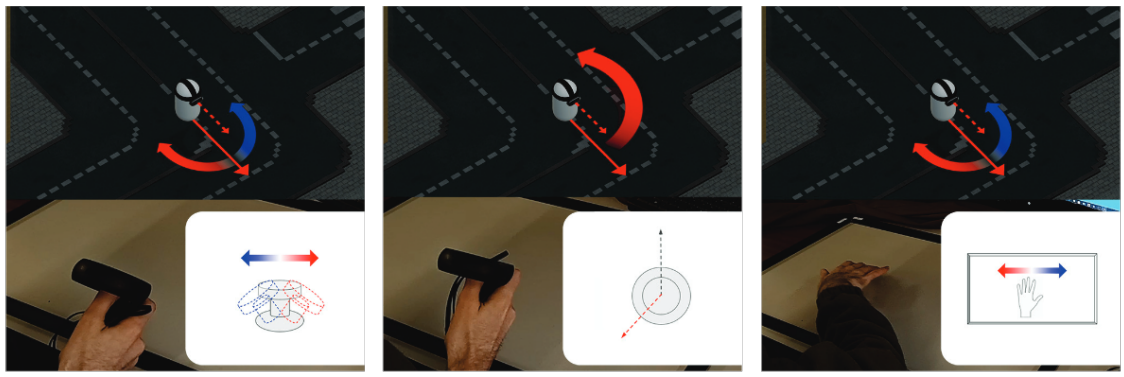
presence without necessitating tiring motions or additional body tracking, such as leg movements. Amaro et al. [3] devised three travel and four orientation techniques, seen in Figure 2.13. One travel technique utilized a VR controller named Continuous Directional Movement. In contrast, the others employed a touch surface and gestures, known as Dog Paddle and Drag'n Go.



(a) (b) (c)

Figure 2.13: Travel techniques designed for DeskVR by Amaro et al. [3]: Continuous Directional Movement (a), Dog Paddle (b), Drag'n Go (c).

For orientation, Amaro et al. [3] introduced two techniques utilizing VR controllers, named Continuous Directional Rotation and Choose & Click, one technique utilizing a tactile surface, named Tactile Surface Dragging, and one technique relying on the orientation of the user's head, named Gaze Convergence. These techniques are illustrated in Figure 2.14. Among the movement techniques, Continuous Directional Movement outperformed its counterparts in performance and comfort, although it appeared to induce more nausea than Dog Paddle. The discomfort in Dog Paddle could stem from its repetitive, exaggerated motions, indicating a potential avenue for exploring more straightforward and less straining interactions. Concerning orientation, Tactile Surface Dragging exhibited superior overall performance, showing fewer cybersickness symptoms than Continuous Directional Rotation, which exhibited the most.



(a) (b) (c)

Figure 2.14: Orientation techniques designed for DeskVR by Amaro et al. [3]: Continuous Directional Rotation (a), Choose & Click (b), Tactile Surface Dragging (c).

2.4 Discussion

The key takeaway in developing interactions for DeskVR is prioritizing user comfort and leveraging the potential benefits of having a desk in front of the user while being mindful of the associated constraints. Numerous studies have investigated the effectiveness of touch-based interactions in both non-steroscopic contexts, such as Balloon Selection [5], Corkscrew Selection [10], and Triangle Cursor [58], and in the context of DeskVR, exploring various scenarios such as menu navigation [65], medical image data examination for radiologists [56], travel techniques [3], and object interaction [2]. The consensus from these studies suggests that, although a touch-based approach may not always be the most time-efficient, it significantly reduces physical strain, enhancing overall comfort and prolonged usage time for individuals. This aspect is essential for DeskVR, where the primary objective is to alleviate physical strain, improve accessibility, and boost productivity [67, 66]. For this reason, touch-based approaches seem preferable over mid-air interactions. Furthermore, Zielasko et al. [68] demonstrated that presenting a virtual table as a surrogate for the physical table has minimal impact on task performance, cybersickness, and presence, making it a viable option for menus and other interactions without compromising the user experience.

The absence of physical feedback during simultaneous interactions by multiple users creates unpredictability in real-time multi-user collaboration [49]. While haptic devices like the Phantom Omni [52] offer a potential solution by allowing users to manipulate objects and perceive forces applied by others [48], their integration into DeskVR, potentially involving substitutional reality – replacing real physical objects such as haptic devices with virtual counterparts [53], is beyond the scope of this work. Furthermore, the limited mobility of users in DeskVR constrains simultaneous multi-user object interaction, particularly when considering physics-based interaction techniques such as concurrency control using consensus-based networking [18].

Considering these factors, level 3 collaboration is unsuitable, and thus, there is no need for distributed average concurrency control techniques. Additionally, degree-of-freedom separation techniques appear restrictive in configuration and interaction while potentially confusing and difficult to use. As such, the fine-grained approach of Lee et al. [28] is appealing, allowing users to discuss different object interactions and arrangements while sharing their perspectives. Relevant work has applied this concept in VR, such as the implementation of object previews in the work by Pereira et al. [40]. Extending this concept to DeskVR is a promising avenue for exploration.

Moreover, providing social visibility, awareness, and accountability in collaborative environments is pivotal in shaping how we communicate [16, 22]. The chosen approach should provide these elements, enabling users to communicate more effectively. For instance, it should indicate who is interacting with an object, its owner, and who is engaged in the conversation, seamlessly integrating these aspects within the DeskVR environment. Additionally, it should leverage the spatial attribute of VR, incorporating a spatial model of interactions [4, 13]. Users within the proximity of an object's aura could trigger interactions, such as audio communication, enabling them to focus on the task at hand with fewer distractions. This aligns with the concept of translucence as discussed by Erickson and Kellogg [16].

Chapter 3

Replico

After evaluating the literature, there appears to be a promising avenue for exploring a method for DeskVR users to engage in discussions and communicate about various objects and areas of interest within a shared virtual environment. This approach could enhance communication effectiveness, especially considering the challenges associated with verbally referencing objects and describing spatial locations [39]. Moreover, given the unique constraints of DeskVR, a research gap exists in applying such methods to this specific environment. Thus, this work introduces Replico, a collaborative touch-based DeskVR approach that utilizes the world-in-miniature (WIM) [57] metaphor to facilitate reasoning about 3D models.

3.1 Overview

After recognizing the potential of a collaborative DeskVR approach to enhance communication and reasoning about 3D models, it became important to outline the basic needs such an approach should meet. These requirements stem from the desire to address common challenges faced by users in virtual environments, aligning with DeskVR's goals of minimizing physical effort to enable longer periods of productive work, reducing mistakes to prevent frustration, and getting more done in less time. Additionally, the approach should be easy to understand to facilitate seamless collaboration with others and allow users to communicate effectively about objects and areas of interest even when they are out of sight.

Furthermore, users must know where others are, who they are, and what they are doing in the virtual space to work well together. Users should be able to easily determine the location and activities of their counterparts so they can coordinate and talk effectively. Importantly, all interactions and tasks should be achievable while seated so they can keep working without getting too tired.

To meet these needs, the chosen approach utilizes the world-in-miniature (WIM) metaphor [57]. The WIM is a miniature replica of the virtual environment that can be easily manipulated within arm's reach and viewed from multiple angles, making it a good fit for DeskVR. Changes

made in the miniature model are reflected in the full-scale model. Additionally, the WIM is effective for displaying social information, such as users' locations and where they are looking, as well as indicating who is working. Each user has a personal view of the WIM, while the to-scale model is shared.

The approach allows users to create points of interest to facilitate communication about objects and zones of interest. These points are uniquely identified by a number and an appearance corresponding to their owner. For instance, one user's points of interest might be represented by green striped spheres, while purple checkerboard cubes represent another user's points. These points of interest are visible in both the to-scale 3D model and its WIM counterpart and remain visible even if occluded.

Users are attached to virtual tables that correspond to their real-life counterparts. They can either remain at separate tables or join another user's table to share the same point of view of the to-scale model. Users can also teleport around the 3D model, taking their table with them if they are alone or creating a new one if they are at someone else's table.

To reduce physical effort, the approach uses touch-based gestures for interactions, such as moving the replica and creating points of interest. Literature suggests that touch-based interactions help reduce physical strain and increase comfort [5, 2]. Specifically, the approach incorporates the Balloon Selection metaphor [5] for creating points of interest. Balloon Selection was chosen mainly due to its "String Height Clutching" feature, which allows users to increase the height of the selection cursor infinitely. This is in contrast to the Triangle Cursor [58], and it is less straining than Corkscrew Selection [10], which requires rotational movement.

3.2 Actions

This section details the specific actions users can perform within Replico using touch-based gestures. These actions encompass various interactions and transformations. Similar to Balloon Selection [5], these gestures are handedness-agnostic, meaning they can be performed with either the left or right hand. The following subsections outline the key actions and how to perform them.

3.2.1 Replica Transformations

Transformations to the replica closely resemble the common gestures associated with touch screens on mobile phones for zooming, panning, and rotating images. Translation in the XZ plane is done by placing one or more fingers on the touch table and moving them. Yaw rotation around the Y-axis is achieved by rotating the fingers around the center of their positions, which also serves as the center of rotation. Scaling the replica in all dimensions is accomplished using a pinch gesture, with the scaling base centered on the midpoint of the fingers and the WIM's base y position.

Translation along the Y-axis is achieved by first placing a finger from the primary hand on the table, followed by two fingers from the secondary hand. The fingers of the secondary hand control translation on the XY plane, while the fingers of the primary hand can perform all the previously mentioned transformations.

3.2.2 Balloon Selection

Replico uses the Balloon Selection metaphor [5] to create, delete, and acknowledge points of interest, as well as teleportation, and join another user's table. To initiate the balloon selection gesture, the user places a finger from their primary hand on the table, followed by a finger from their secondary hand. This action brings up a balloon on the user's table relative to the WIM, with a copy visible on the to-scale 3D model. The primary finger controls the XZ position of the balloon while moving the fingers apart lowers the balloon, and bringing them together raises it. Users can remove the secondary finger without changing the balloon's height, allowing for "String Height Clutching."

To perform a *selection*, the user briefly adds a second finger from their secondary hand to the touch table. This *selection* action can create points of interest at the balloon's position, delete points of interest if the balloon intersects with one of the user's points, acknowledge points of interest if the balloon intersects another user's unacknowledged points, and join other users' tables if it intersects another user's table.

For *teleportation*, the user similarly adds a second finger from their secondary hand to the touch table but holds it in place until an arrow appears on the balloon, indicating the teleportation's end orientation. To rotate the balloon, the user performs a gesture similar to the replica's rotation gesture by rotating their fingers around their midpoint. Either hand can be removed to reposition, but removing both hands cancels the teleportation. To confirm the teleportation, the user taps again with the second finger of the secondary hand.

3.3 Awareness

Following Table 2.2, Replico incorporates most elements of workspace awareness. Users have distinct appearances that uniquely identify them, which is also reflected in their points of interest, albeit with some distinguishing features. User tables are represented in the WIM by outlined miniatures, positioned and oriented as they are in the to-scale model, and are visible behind objects. These miniature tables display who is at each table through miniature representations of the users.

Points of interest created by other users are initially in an *unacknowledged* state, marked by a vertical line capped with a symbol that resembles the owner's appearance and includes the balloon's identification number. This system, combined with the acknowledgment gesture described in 3.2.2, allows users to distinguish recently created points of interest from previously created ones.

These representations blend abstract and mimetic representations of social information as described in [16]. They are mimetic in representing real-world entities like tables and user avatars, yet abstract as they are symbolic and easy to create and manipulate.

Thus, Replico allows users to ascertain the presence of others in the workspace, their locations, and their viewing directions by observing the tables in the replica. Users can identify others by

their appearance, determine collaborators by noting who shares the same table, and discern authorship through the appearance of points of interest. Points of interest, in turn, indicate which objects users are working on. Additionally, users can track recent activities by checking unacknowledged points.

3.4 Summary

This chapter introduces Replico, a collaborative DeskVR approach designed to enhance communication and interaction in the analysis of 3D models. Using the world-in-miniature (WIM) metaphor, Replico addresses common challenges in virtual collaboration, such as spatial referencing and awareness of other users' activities.

First, it explains the requirements such an approach should consider, such as minimizing physical effort, reducing mistakes, ensuring efficiency, effective communication about objects and areas of interest, ease of understanding, providing awareness of other users, and enabling all interactions to be done while seated. To meet these requirements, Replico allows users to manipulate a miniature replica of the virtual environment (WIM). This approach ensures that changes made in the miniature are reflected in the full-scale model. Replico also enables the creation of uniquely identified points of interest, aiding the communication about objects and zones of interest within the virtual space. Users are attached to virtual tables corresponding to their real-life counterparts, allowing them to join others' tables or teleport around the 3D model.

Section 3.2 explains the different touch-based gestures users can perform within Replico: translation, rotation, and scaling of the WIM; and creation, deletion, and acknowledgment of points of interest, as well as teleportation and joining users' tables, using balloon selection. Section 3.3 explains how Replico incorporates workspace awareness: users and their points of interest share a unique appearance, making them identifiable; user tables are represented in the replica; and points of interest can be acknowledged to identify recent points of interest quickly.

Chapter 4

Implementation of a Prototype

This chapter describes the implementation of the prototype created to test the viability of Replico. It begins by explaining the architecture, hardware, software, and tools used to develop the prototype. The chapter then details how hand detection is performed, the various states within the system, the methods for displaying feedback to users, and the networking implementation details.

4.1 Architecture

To develop the prototype, two HTC Vive Pro 2 headsets and two multi-touch surfaces – a 32-inch infrared frame and a 47-inch capacitive Displax Skin Ultra[1] touchscreen – were used. The Unity[2] game engine was chosen for its robust VR support, personal previous experience, extensive community resources, and excellent compatibility with external C# libraries, which helped reduce development risks.

Unity's OpenXR plugin[3] controls VR hardware communication, handling input and rendering with minimal effort, and managing all API calls automatically. OpenXR[4] is an API standard developed by Khronos for XR applications, including VR, and is widely adopted across many XR devices with conformant runtimes. Due to its ubiquity and recency, it was chosen over alternatives such as using SteamVR directly.

Since Unity's OpenXR plugin interfaces with Unity's Input System, the Unity Enhanced Touch API[5] was used instead of the standard Unity Touch API to maintain a consistent input management system. The Enhanced Touch API provides automatic finger tracking and keeps a history of touch interactions.

[1] https://www.displax.com/skin-ultra
[2] https://unity.com/
[3] https://docs.unity3d.com/Packages/com.unity.xr.openxr@1.11/manual/index.html
[4] https://registry.khronos.org/OpenXR/specs/1.1/html/xrspec.html
[5] https://docs.unity3d.com/Packages/com.unity.inputsystem@1.0/api/UnityEngine.InputSystem.EnhancedTouch.html

The first-party Unity Netcode for GameObjects[6] library was used for networking. This library offers a straightforward abstraction of networking logic and is easy to use and set up for client-server or distributed authority topologies. Its simplicity is well-suited for a small number of clients, making it ideal for this prototype.

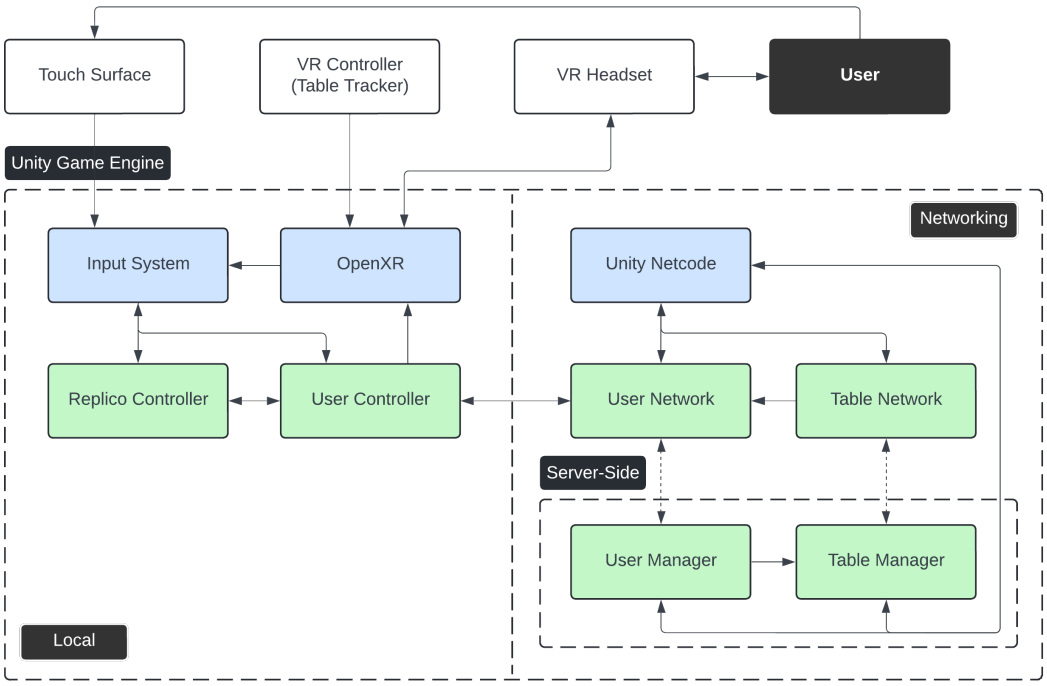


Figure 4.1: System architecture. Modules in blue represent Unity libraries, while the prototype implements modules in green.

The system architecture diagram in Figure 4.1 demonstrates the integration of various software and hardware components, describing both local and networked elements. OpenXR manages communication with the VR headset and the table tracker, rendering images to the headset, updating the virtual camera's position, and interfacing inputs to Unity's Input System. Unity's Enhanced Touch API processes touches sensed by the touch surface from the Input System. The Replico Controller and User Controller manage interaction logic and user actions, communicating with the Input System and OpenXR. Networking is handled by Unity Netcode, which synchronizes user and table data across the network via the User Network and Table Network, using a client-server topology. The server-side User Manager and Table Manager manage data for users and tables. The User Manager tracks users in the system and communicates with the User Network objects through remote procedure calls (RPCs) and network variables. The Table Manager handles table creation, deletion, and management logic, communicating with Table Network objects using RPCs and network variables.

[6] https://docs-multiplayer.unity3d.com/netcode/current/about/

4.2 State Machine

The interpretation of touch surface input varies depending on the gestures employed. To address this, a state machine was created. A state machine consists of defined states with distinct transitions, where each state processes the input differently. Implementation was achieved through the state design pattern, wherein each step is represented as a class that alters the behavior of the Replico controller. Figure 4.2 provides a diagram illustrating the implemented state machine.

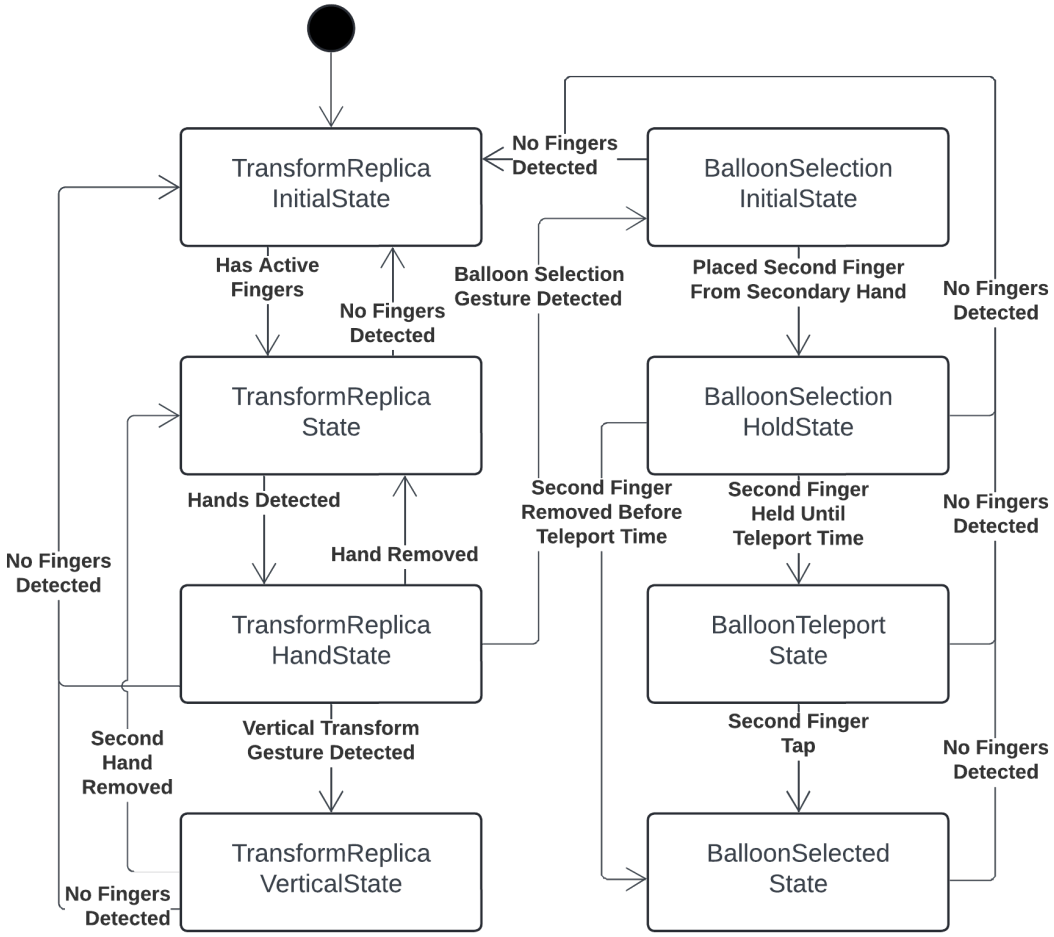


Figure 4.2: State machine diagram.

The `TransformReplicaInitialState` is the starting state where no fingers are detected, and therefore no controls are active. In this state, finger touches are checked every frame using the Enhanced Touch API, which updates every frame because the Input System update mode is set to Dynamic. When at least one finger is detected, the system transitions to `TransformReplicaState`. All states revert to this initial state whenever no fingers are detected.

The `TransformReplicaState` is entered when fingers are detected, but no hands are recognized yet. In this state, the user can move the replica using the gestures described in Section 3.2.1 and

implemented in Section 4.3. Every frame it checks for hands using the method described in Section 4.4.1. When both hands are detected, it transitions to `TransformReplicaHandState`.

In `TransformReplicaHandState`, users can move the replica as in `TransformReplicaState`. This state detects the user's gestures in every frame using the method described in Section 4.4.2. It transitions to `TransformReplicaVerticalState` if the vertical transform gesture is detected, or to `BalloonSelectionInitialState` if the balloon selection gesture is detected. If any hand is removed, it transitions back to `TransformReplicaState`.

In `TransformReplicaVerticalState`, the user can move the replica as in `TransformReplicaState` using their primary hand, while the secondary hand performs translation on the XY plane. To allow users to temporarily remove the secondary hand to reposition their fingers without terminating the gesture, the secondary hand may be removed for up to 0.55 seconds before the controller transitions back to `TransformReplicaState`.

In `BalloonSelectionInitialState`, the user performs the balloon selection gesture as described in Section 3.2.2. The primary finger moves the balloon on the XZ plane, while moving the fingers together raises the balloon and moving them apart lowers it. The balloon's height is saved between gestures, so the user doesn't have to raise it each time they start balloon selection. It transitions to `BalloonSelectionHoldState` when a second finger is added to the secondary hand. Instead of transitioning instantly to the initial state when no fingers are detected, there is a grace period of 0.15 seconds. This grace period accounts for potential hardware tracking failures that could prematurely stop the balloon selection gesture, potentially causing frustration.

The `BalloonSelectionHoldState` monitors how long the user holds down the second finger from the secondary hand. If the finger is held for 0.4 seconds or the primary hand moves, the state transitions to `BalloonTeleportState`. Removing the finger before 0.4 seconds results in different actions: joining a table if the balloon intersects one, acknowledging or deleting a point of interest if it intersects one, or creating a new point of interest if it intersects none, thereby transitioning to `BalloonSelectedState`.

In `BalloonTeleportState`, the user can rotate their balloon using the same gesture as rotating the replica, following the calculations described in 4.3. Confirmation of teleportation occurs if the user taps with the second finger of their second hand (removing and placing it again) or if two touches are detected within the hand detection distance outlined in 4.4.1. Upon confirmation, the controller transitions to `BalloonSelectedState`.

The purpose of `BalloonSelectedState` is to act as a buffer between balloon selection interactions and replica transformations. Users can only perform replica transformations after removing all finger touches, at which point the controller transitions back to `TransformReplicaInitialState`.

4.3 Replica Transformations

This section discusses the transformations applied to the replica in states `TransformReplicaState`, `TransformReplicaHandState`, and `TransformReplicaVerticalState`.

Translation is achieved by calculating the distance $\vec{D} = C_i - C_{i-1}$, where C_i and C_{i-1} are the centers of the active touches on the touch surface from the current frame and the previous frame, respectively, as shown in Equation 4.1.

$$\begin{aligned} \mathbf{F}_i &= \{(x_{i_k}, y_{i_k}) : \ k = 1, \ \ldots, \ n\} \\ \mathbf{minF}_i &= (\min_k x_{i_k}, \ \min_k y_{i_k}) \\ \mathbf{maxF}_i &= (\max_k x_{i_k}, \ \max_k y_{i_k}) \\ C_i &= \frac{\mathbf{minF}_i + \mathbf{maxF}_i}{2} \end{aligned} \tag{4.1}$$

Here, $\mathbf{F}_i$ represents the positions of the n fingers in the i-th frame, $\mathbf{minF}_i$ and $\mathbf{maxF}_i$ are the minimum and maximum x and y coordinates from the set of finger positions. Essentially, C_i is the geometric centroid of the smallest rectangle that can enclose all the touch points. This distance is then multiplied by a factor t, which can be adjusted for each 3D model, resulting in $\vec{T} = t \cdot \vec{D}$. This yields a 2D vector that is added to the replica's position in the XZ plane.

Scaling is achieved by first calculating $s = \left(\bar{d}_i / \bar{d_{i-1}}\right)^c$, where $\bar{d}_i$ and $\bar{d_{i-1}}$ are the average distances of the active touches to the center of those touches from the current frame and the previous frame, respectively, and c is a constant used to modulate the scaling effect. The calculation for the average distance is shown in Equation 4.2.

$$\bar{d}_i = \frac{\sum_{k=1}^{n} \|C_i - F_{i_k}\|}{n} \tag{4.2}$$

Finally, this scaling is applied to the replica around a base point. To do this, the pivot point in world coordinates, pivot_w, is first converted to local coordinates, pivot_l, relative to the replica. The replica's local scale is then multiplied by s. After scaling, the position of the pivot in world coordinates, pivot_k, calculated by converting pivot_l back to world coordinates, will not be equal to pivot_w. To correct this, the displacement $\Delta\vec{\text{pivot}} = \text{pivot}_w - \text{pivot}_k$ is calculated and added to the replica's world position.

Rotation is achieved by calculating the fingers' average rotation $\bar{\theta}$. The calculation for this is shown in Equation 4.3, where $\vec{\text{dir}}_{i_k}$ is the vector direction from the center of the fingers to the k-th finger in the i-th frame, $|\theta_k|$ is the angle between the vector direction of the previous frame and the current frame for the k-th finger. The angle θ_k is determined by adjusting $|\theta_k|$ based on the cross product's z-component to account for direction, as the arccosine function's range only goes from 0 to π. The replica is then rotated around the Y axis that passes through $(C_{i_x}, 0, C_{i_y})$ with the angle $\bar{\theta}$, using Unity's `RotateAround`.

$$
\begin{aligned}
\vec{\text{dir}}_{i_k} &= \mathbf{F}_{i_k} - C_i \\
|\theta_k| &= \cos^{-1}\left(\frac{\vec{\text{dir}}_{i-1_k} \cdot \vec{\text{dir}}_{i_k}}{\|\vec{\text{dir}}_{i-1_k}\|\|\vec{\text{dir}}_{i_k}\|}\right) \\
\theta_k &= \begin{cases} |\theta_k| & \text{if} \quad (\vec{\text{dir}}_{i-1_k} \times \vec{\text{dir}}_{i_k})_z < 0 \\ -|\theta_k| & \text{if} \quad (\vec{\text{dir}}_{i-1_k} \times \vec{\text{dir}}_{i_k})_z \geq 0 \end{cases} \\
\bar{\theta} &= \frac{\sum_{k=1}^{n} \theta_k}{n}
\end{aligned}
\tag{4.3}
$$

In the `TransformReplicaVerticalState`, the primary hand can perform XZ translation, rotation, and scaling, while the secondary hand can only perform translation on the XY plane. Only the fingers from the primary hand are considered for transformations with the primary hand. The secondary hand's fingers can only perform translation as previously described, but instead of $\vec{T}$ being applied to the replica's XZ position, it is applied to the XY position.

The transformations are not directly applied to the replica; instead, they are applied to a target object. The replica then follows this target object using Unity's `SmoothDamp` for position and scale, and `SmoothDampAngle` for each Euler rotation angle. This helps to reduce jitter caused by low-frequency or low-accuracy touch input updates.

4.4 Gesture Detection

This section explains how the different gestures – transformation, vertical transformation, and balloon selection – are distinguished. Section 4.4.1 explains the method for detecting and distinguishing the user's hands, and Section 4.4.2 describes how the vertical transform and balloon selection gestures are differentiated.

4.4.1 Hand Detection

Detection and distinction of hands are important for recognizing Replico's touch-based gestures. The prototype took a simplistic approach to hand detection, using input solely from the touch surface and Unity's Enhanced Touch API. The method involves detecting two clusters based on finger proximity. For this purpose, this prototype uses a naive K-Means clustering algorithm [32, 31, 60] with a k value of 2. The algorithm was implemented using the ML.NET library[7], a machine learning library for .NET. For compatibility with Unity, the .NET Standard 2.1 version was used. The distance function used is the Euclidean distance between the finger positions on the touch surface in pixels, and the Yinyang initialization algorithm [11] is applied.

Other clustering algorithms, such as DBSCAN [17], were not used because they do not allow the specification of a fixed number of clusters (k) or because they perform better with a larger

[7] https://github.com/dotnet/machinelearning

number of clusters. K-means is perfectly adequate in this case with a maximum of 10 points (one for each finger) and only 2 clusters.

The K-means algorithm returns two clusters of fingers when more than one finger is placed on the touch surface. However, this can result in two fingers from the same hand being detected as separate clusters. To address this, a distance threshold between cluster centroids is used to determine if the clusters represent separate hands. The threshold distance is measured relative to a min-max normalized value of the screen dimensions in pixels. A distance threshold of 0.18 was found to be effective through testing.

Initially, before any hands have been detected in the `TransformReplicaState` shown in Figure 4.2, the distinction between the primary and secondary hands is maintained by queuing fingers based on the order of their touch. The cluster containing the first detected finger represents the primary hand. Once both hands are detected in the `TransformReplicaHandState` and beyond, hands are updated each frame by reapplying the K-means algorithm. The distinction is then made by counting how many fingers from the previously detected primary hand are in each newly detected cluster; the cluster with the most fingers from the primary hand is associated with it. To update the hands, new fingers in the clusters are added to the corresponding hands, and previously detected fingers remain in their respective hands unless they have been removed from the touch surface. This approach allows left-handed and right-handed users to perform all Replico gestures easily.

4.4.2 Distinguishing Vertical Transform and Balloon Selection

The vertical transform and balloon selection gestures, described in Sections 3.2.1 and 3.2.2 respectively, can be easily confused with the pinch gesture required for scaling the replica. To aid in this distinction:

- **Vertical Transform:** At least one finger on the primary hand and at least two fingers on the secondary hand. The secondary hand must remain stationary for 0.2 seconds.
- **Balloon Selection:** Exactly one finger on the primary hand and exactly one finger on the secondary hand. Both hands must remain stationary for 0.2 seconds.

The vertical transform only checks if the second hand has moved, allowing the user to add the secondary hand while transforming with the first. The criterion for determining if a hand hasn't moved is that none of its fingers have moved past a threshold δ from the position where they were first placed. This threshold is measured relative to a min-max normalized value of the screen dimensions in pixels, with testing indicating 0.01 as an appropriate value. Once a finger has moved, it will be considered moved until it is removed.

4.5 Table Tracking

The user's table is tracked to a real-world table using a VR controller, as shown in Figure 4.3. The controller is positioned pointing toward the user instead of forward from the table so that the

tracked controller position matches the table's corner. If it pointed the other way, a translation based on the controller's length toward the table corner would be necessary, which is not feasible for all controller types.

Figure 4.3: Tracking the table using a VR controller.

When attaching a user to a virtual table, such as when the user joins a table or teleports, the orientation and position of the tracker must match the orientation and position of the table's attach point, which is an empty GameObject on the table's Prefab. To achieve this, the user's orientation is first updated using `MatchOriginUpCameraForward`, followed by updating the position with `MoveCameraToWorldLocation`, both functions from the OpenXR plugin.

The `MatchOriginUpCameraForward` function requires two parameters: an up vector and a forward vector. The up vector is set to match the attach point's up vector, assuming the user and controller are on flat ground. The forward vector, $\vec{v}_{\text{forward}}$, is calculated as shown in Equation 4.4. Here, $\mathbf{q}_{\text{tracker}}$ represents the quaternion rotation of the tracker, and $\mathbf{q}_{\text{attach}}$ represents the quaternion rotation of the table's attach point. $\Delta\theta$ is the rotation of the attach point relative to the tracker. The yaw component is isolated to ignore pitch and roll, preventing rotation along the x and z axes due to the controller rolling, assuming the user is on flat ground. $\mathbf{q}_{\text{target}}$ is the user's target rotation, combining the user's current yaw rotation with the relative rotation of the attachment point. Finally, the forward vector is obtained by multiplying $\mathbf{q}_{\text{target}}$ by the (0,0,1) vector. The `MoveCameraToWorldLocation` function requires a position parameter. This position is calculated using the equation $P = P_{\text{attach}} + P_{\text{user}} - P_{\text{tracker}}$.

Because the controller may fall accidentally while the user is interacting with the touch surface and cannot see it when using the VR headset, it is only used to track the table when the user first joins a table. To achieve this, the tracker's rotation relative to the user is calculated using $\mathbf{q}_{\text{local}} = \mathbf{q}_{\text{tracker}} \cdot \mathbf{q}_{\text{user}}^{-1}$ and stored. Additionally, the tracker's local position relative to the user is calculated using the user's `InverseTransformPoint` function and stored.

$$\begin{aligned} \Delta\theta &= \mathbf{q}_{\text{tracker}}^{-1} \cdot \mathbf{q}_{\text{attach}} \\ \Delta\theta_Y &= \text{Quat}\,(0, \text{yaw}(\Delta\theta), 0) \\ \mathbf{q}_{\text{user}_Y} &= \text{Quat}(0, \text{yaw}(\mathbf{q}_{\text{user}}), 0) \\ \mathbf{q}_{\text{target}} &= \mathbf{q}_{\text{user}_Y} \cdot \Delta\theta_Y \\ \vec{v}_{\text{forward}} &= \mathbf{q}_{\text{target}} \cdot (0,0,1) \end{aligned} \tag{4.4}$$

To convert the local rotation back to world rotation, the calculation $\mathbf{q}_{\text{tracker}} = \mathbf{q}_{\text{user}} \cdot \mathbf{q}_{\text{local}}$ is used. Similarly, the local position is converted back to world position utilizing the user's `TransformPoint` function. These world coordinates and rotations are then used in the previously described calculations.

4.6 Visual Feedback

The prototype uses various visual indicators as forms of feedback. These include a virtual touch frame with finger indicators described in Section 4.6.1, a visual representation of the touch frame limits detailed in Section 4.6.2, tables and points of interest visible in both the 3D model and the replica as described in Sections 4.6.3 and 4.6.4, and effects related to balloon selection in Section 4.6.5, among others.

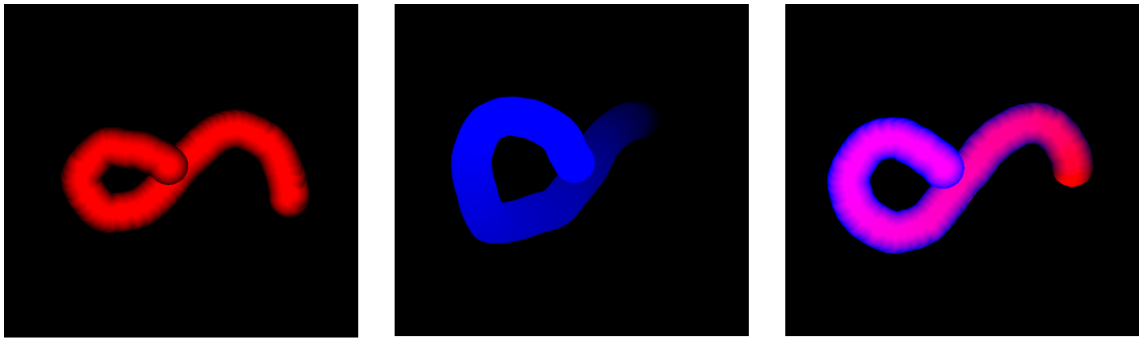
4.6.1 Virtual Touch Frame

In VR, users cannot see their hands or where their fingers are positioned. The prototype includes finger indicators within the virtual touch frame to address this issue, as depicted in Figure 4.4. Each finger is assigned a distinct color based on the order in which it was placed on the frame. Additionally, the finger trail shrinks from the current finger positions to previous positions, helping users understand their finger movements over time.

Figure 4.4: Touch indicators for four fingers on the touch frame.

This is implemented using a compute shader and a shader built with Unity's Shader Graph. A compute shader is a program that runs on the GPU outside of the normal rendering pipeline.[8] It is most useful for executing highly parallel algorithms. In this case, the compute shader processes finger positions, performs calculations, and stores results in render textures. The Shader Graph shader then uses these textures to render finger trail indicators in every frame.

Each render texture stores data for two fingers using two channels per finger per pixel. This results in five textures, each with dimensions matching the greatest nearest power of two between the screen width and height. One channel stores the reverse distance from the pixel to the center of the finger trail, ranging from 1 (closest to the center) to 0 (outside the trail radius), similar to a signed distance function. The other channel records the decay of the pixel, where 1 indicates the most recent position, and 0 indicates total decay. These channels are depicted in Figure 4.5, with red representing the distance to the trail's center and blue representing the decay.



a) b) c)

Figure 4.5: The different components stored on the render texture for each finger: a) reverse distance to center; b) decay; c) distance and decay combined.

A compute shader function is executed by several compute shader thread groups for each of the three dimensions: X, Y, and Z. In this case, the function defines for each group 8 threads for the X dimension, representing the pixel coordinate on the X axis, 8 threads for the Y dimension, representing the pixel coordinate on the Y axis, and 1 thread for the Z dimension, representing the render texture being processed. When the compute shader is executed, it runs using $\text{texture}_{\text{width}}/8$ groups on the X axis, $\text{texture}_{\text{height}}/8$ groups on the Y axis, and 5 groups on the Z axis, one for each render texture.

The compute shader takes several inputs: five different render textures, two `float4` structured buffers of size 5 (one for the current finger positions and one for the finger positions of the previous frame), a `float` structured buffer of size 10 that stores the average inclination of each finger trail, a linear decay rate δ, a finger radius in pixels r, and the time elapsed between the last frame and the current frame in seconds Δt. Each `float4` vector in the structured buffers represents the screen-space coordinates of two fingers, with the first two floats for one finger and the next two floats for another finger.

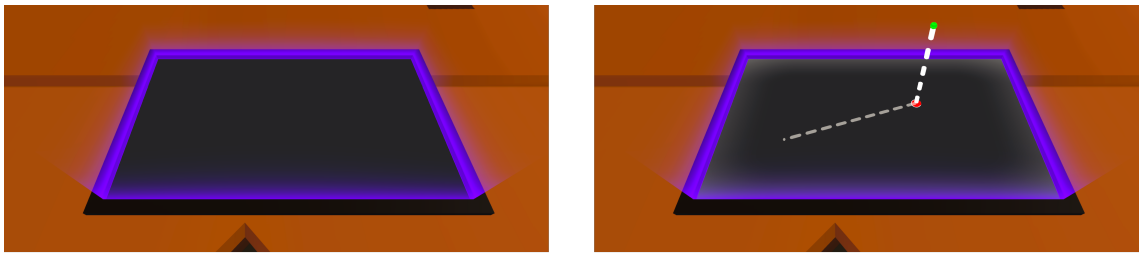
[8] https://docs.unity3d.com/Manual/class-ComputeShader.html

A simplified description of the algorithm for each finger and each pixel operates as follows: first, it calculates the decay λ_i of the pixel using the previous frame's decay value λ_{i-1} by computing $\lambda_{i-1} - \delta \cdot \Delta t$. Next, it calculates the distance d and reverse distance d_{rev} from the pixel to the line segment that starts at the finger's position in the previous frame and ends at the current finger position. This calculation, shown in Equation 4.5, creates a capsule-like shape. In this equation, A is the finger's position in the last frame, B is the current finger's position, r is the trail's thickness, and P is the pixel's position.

$$\begin{aligned} \vec{AB} &= B - A \\ \vec{AP} &= P - A \\ h &= \text{clamp}\left(\frac{\vec{AP} \cdot \vec{AB}}{\vec{AB} \cdot \vec{AB}}, 0, 1\right) \\ d &= \frac{\|\vec{AP} - h \cdot \vec{AB}\|}{r} \\ d_{\text{rev}} &= \text{clamp}\,(1 - d, 0, 1) \end{aligned} \tag{4.5}$$

Based on how much the pixel is in front of the line segment, considering the average inclination of the finger trail, the algorithm linearly interpolates between $\max\left(d_{\text{rev}}, d_{\text{rev}_{i-1}}\right)$ and d_{rev} to calculate the value to store in the render texture. This approach allows the finger trail to overlap at the front and smoothly blend behind. The final decay value stored in the texture is λ_i, with an additional 1 added if $d \leq 1$, clamped between 0 and 1.

Finally, the fragment shader samples each render texture. It subtracts the decay value from the distance to create a shrinking effect, applies a border by comparing the resulting value with a threshold, and assigns a color based on the finger's order.



a) b)

Figure 4.6: Glow effect indicating gesture detection on the touch surface. (a) The initial state with no gesture detected. (b) The glow effect activates when a gesture is detected.

The virtual touch frame illuminates with a glowing effect whenever the system detects vertical transform or balloon selection gestures, signaling the user that their gesture has been recognized. This glow effect, demonstrated in Figure 4.6, is achieved using a shader that uses a rounded box signed distance function.[9] The strength of the glow is animated using the function $-\left(2\sqrt{t}-1\right)^2+$

[9] https://www.shadertoy.com/view/Nlc3zf

1, where t represents the elapsed time since the animation began. Initially, this function rises quickly until the result reaches 1 at $t = 0.25$, after which it gradually diminishes. The animation halts at $t = 0.8$ and resumes from that point when the gesture concludes.

4.6.2 Frame Limit Indicator

The balloon's position on the XZ axis during balloon selection is constrained by the boundaries of the virtual touch frame. To help users understand these boundaries, even when the frame is obscured by the replica, a purple illumination effect outlines the limits. This effect is shown in Figure 4.7.

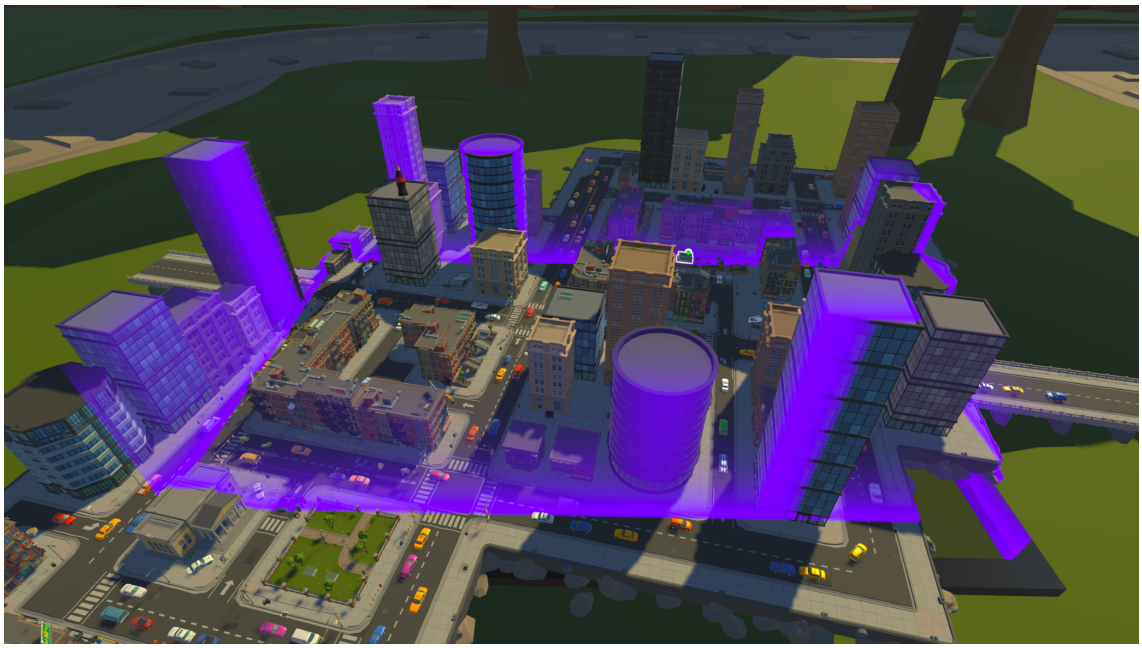


Figure 4.7: Illumination effect on the replica indicating the limits of the touch frame.

The illumination effect is achieved using a shader applied to a transparent rectangular prism extending from the touch frame's base. The shader's primary function is to gradually diminish the illumination effect as the distance from the prism increases. This is accomplished using a modified version of a shader initially designed for a stylized water effect[10], created using Unity's Shader Graph.

To calculate the distance d, a vector $\vec{CA}$ is obtained from the camera to the fragment's position on the prism using the View Vector node. This vector is then normalized to $\hat{v}$. The depth texture is sampled to obtain the distance from the camera to the point occluded by the prism, $|CB|$. The normalized vector is multiplied by this distance, resulting in $\vec{CB} = \hat{v} \cdot |CB|$. Adding this vector to the camera's position gives the position of the occluded point, $B = C + \vec{CB}$. The distance vector $\vec{BA}$ is obtained by subtracting the occluded point's position from the fragment's position on the

[10] https://ameye.dev/notes/stylized-water-shader/

prism's surface, $\vec{BA} = A - B$. Finally, the length of this vector is calculated to obtain the distance, $d = \|\vec{BA}\|$.

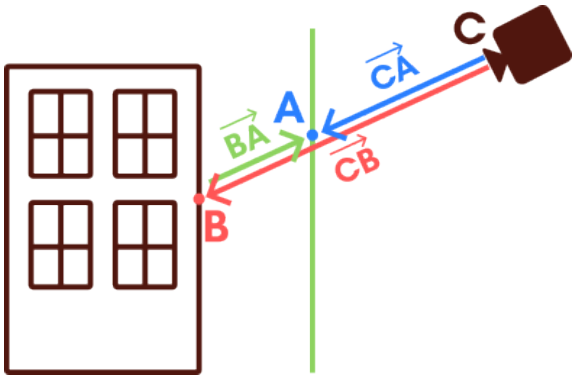


Figure 4.8: Diagram illustrating the steps to calculate d.

To achieve the gradual effect, the function $x = e^{-\frac{d}{0.05}}$ is applied. This ensures that when $d = 0$, the effect is at full power, declines rapidly, and then tapers off. This behavior is shown in graph a) of Figure 4.9. To soften the effect at the borders, the function described in Equation 4.6 and depicted in graph b) of Figure 4.9 is applied to x. This adjustment causes the effect to start at 0.2 power at the border, rise smoothly to 0.8 power, and then taper off as the distance increases. This progression is illustrated in graph c) of Figure 4.9.

The function $-37.5x^3 + 82.5x^2 - 60x + 15.2$ was derived from a cubic polynomial for creating a smooth curve between two points: (c,m) and $(k,m+b)$, shown in Equation 4.7.[11] In this case, the parameters are $c = 0.8$, $m = 0.8$, $k = 1$, $b = -0.6$, $p = 0$, and $q = -1.5$.

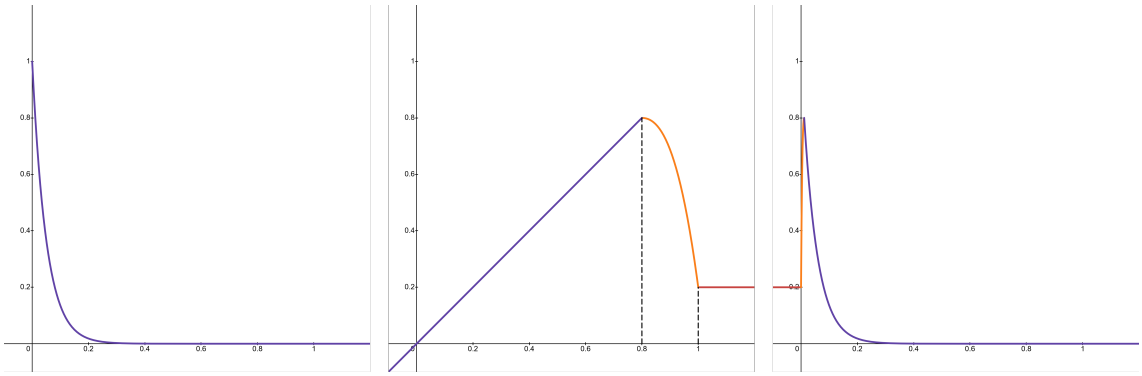
$$\alpha = \begin{cases} x & \text{if} \quad x \leq 0.8 \\ -37.5x^3 + 82.5x^2 - 60x + 15.2 & \text{if} \quad x > 0.8 \\ 0.2 & \text{if} \quad x > 1 \end{cases} \tag{4.6}$$

$$(p+q-2\cdot b)\cdot\left(\frac{x-c}{k-c}\right)^3 + (3\cdot b - 2\cdot p - q)\cdot\left(\frac{x-c}{k-c}\right)^2 + p\cdot\left(\frac{x-c}{k-c}\right) + m \tag{4.7}$$

4.6.3 Virtual Table

As shown in the literature [65, 56, 68], the presence of a virtual table can be helpful in presenting information. This prototype uses the virtual touch frame described in Section 4.6.1 to represent that information. While the table is useful for displaying information, it can obscure much of the to-scale model, especially if the user wants to look down. To make it less intrusive, the table

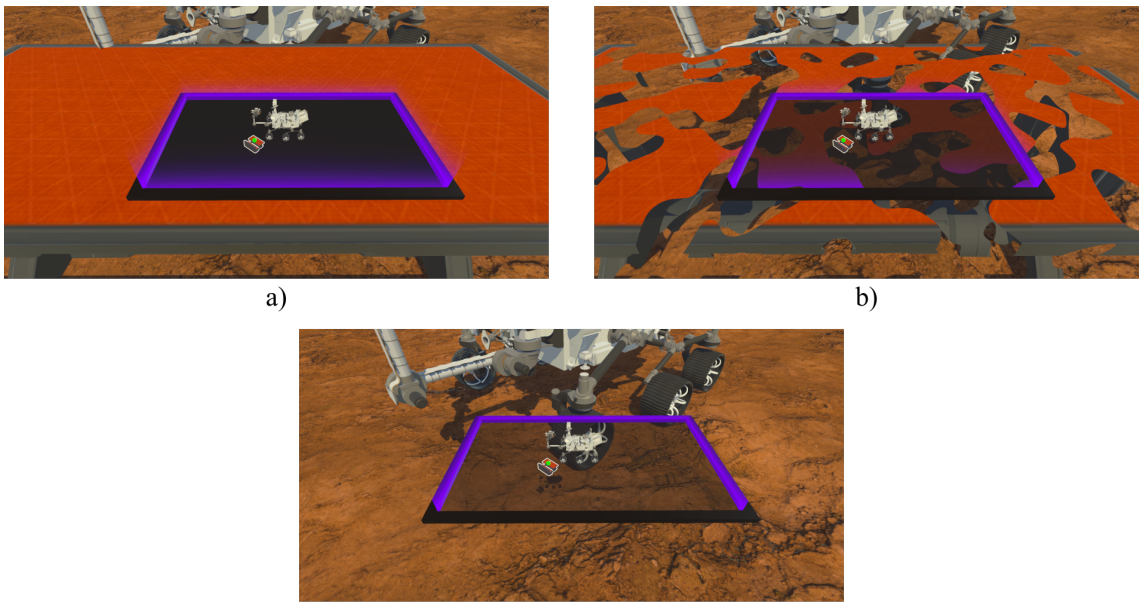
[11] https://math.stackexchange.com/a/2209953



a) b) c)

Figure 4.9: Graphs illustrating the functions used to modify the intensity of the limit illumination effect. Graph a) shows $e^{-\frac{d}{0.05}}$ where the horizontal axis represents distance d. Graph b) displays the function described in Equation 4.6, with the horizontal axis representing x. Graph c) depicts the function from Equation 4.6 with the horizontal axis representing distance d.

begins to fade and becomes invisible after 2 seconds of the touch surface not detecting any fingers, as shown in Figure 4.10.



a) b)

c)

Figure 4.10: The transition of the virtual table from: a) fully visible; b) half-visible; c) fully invisible.

This effect is achieved using a shader that takes the pixel's world space position and applies a simplex 3D noise function[12] to determine the pixel's alpha clip threshold. After 2 seconds of inactivity, the table's alpha value is reduced using a smooth animation curve. The alpha clip

[12] https://github.com/JimmyCushnie/Noisy-Nodes

threshold then determines whether a pixel is visible or invisible. Alpha blending was not used because it caused visual artifacts, making the table visible from behind itself.

User tables are also visible in the replica as miniatures, as shown in Figure 4.11 and described in Section 3.3. These miniatures feature an outline effect to help them stand out from the surrounding environment, using a free Unity package.[13] They also glow intermittently to draw attention, increasing the lightness of the table's color through an animation using a quadratic easing in-out function.[14] The miniatures display who is at the table by showing a user seated at it, as seen in image c) of Figure 4.11. These miniatures do not scale with the replica, keeping their size constant, similar to markers on a map.

They remain visible behind objects in the replica to help users quickly identify their and others' tables. This is achieved by rendering the table miniature in an additional render pass using the depth buffer to determine the appropriate material. If the table is behind an object, it appears slightly transparent and in a single color; otherwise, it uses the normal material.[15]

a) b) c)

Figure 4.11: The table miniature visible in the replica. Image (a) shows the table behind an object, image (b) shows the table within the replica, and image (c) shows two users at the table.

4.6.4 Points of Interest

As mentioned in Section 3.3, the points of interest reflect the appearance of their creators. Figure 4.12 demonstrates this: the first user's points of interest are green-striped spheres, while the second user's are purple checkerboard cubes. Each point of interest is marked with a number that rotates to face the camera and glows intermittently to draw attention, achieved using a Fresnel effect with a sine curve animation.

Similar to the miniature tables, points of interest are visible behind objects in the replica and do not scale with the replica. This is shown in image (a) of Figure 4.13, where a point of interest is visible behind a building in the replica with a muted color and slight transparency. However,

[13] https://assetstore.unity.com/packages/tools/particles-effects/quick-outline-115488

[14] https://assetstore.unity.com/packages/vfx/shaders/shader-graph-easing-193427

[15] https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@10.4/manual/renderer-features/how-to-custom-effect-render-objects.html

a) b)

Figure 4.12: Point of interest appearance based on the creator's appearance. Image (a) shows points of interest from the first user, and image (b) shows points of interest from the second user.

in the 3D model, the point of interest is not visible behind the building, as shown in image (b). This ensures that the points of interest do not distract or confuse users when looking at the replica. Points of interest in the 3D model scale with distance so users can see them from afar but do not become too large when close, preserving essential details.

Points of interest created by another user that are not yet acknowledged are marked with a vertical line, as shown in Figure 4.14. This line is capped with a symbol displaying the point of interest's identification number, resembling the point of interest's appearance to help users quickly identify unacknowledged points. The line and symbol always face the user using a vertical billboard effect. The line is also visible behind objects in the replica and scales with distance, as seen in image (c) of Figure 4.14, ensuring it can be seen from any angle. If the marker does not fit within the user's field of view, it flips upside down to remain visible, as shown in image (d) of Figure 4.14.

4.6.5 Balloon Selection

The balloon selection gesture is indicated by a set of dashed helper lines: one on the touch frame connecting the primary and secondary hands, and another vertical line from the primary hand to the balloon, using a vertical billboard effect, as shown in Figure 4.15. The balloon follows the appearance of the creator's points of interest. The helper lines and the balloon are visible behind objects in the replica with reduced opacity. When the secondary hand is inactive, the helper line on the touch frame loses its opacity, as shown in image (b) of Figure 4.15. If a segment of that line is behind an object and the secondary hand is inactive, that segment becomes invisible.

a) b)

Figure 4.13: Points of interest visibility. Image (a) shows a point of interest in the replica that is visible behind the building. Image (b) shows that the same point of interest is not visible in the 3D model behind the building.

The dashed lines are created using a shader that takes the positions of the hands, calculates the start and end positions of the dash segments using the percentages for each dash and gap, then draws the dash segments based on the calculations described in Equation 4.5. The dash segments at the ends are masked with a line from the start to the end position to ensure they have rounded ends.

The balloon and the vertical helper line are also visible in the 3D model, as shown in Figure 4.16, helping users understand the balloon's position in the real world. Both are not visible behind objects in the 3D model. The balloon scales with distance so users can see it from afar.

When the balloon intersects a point of interest, the point of interest becomes outlined, indicating it is selected, as shown in image (a) of Figure 4.17. When the balloon intersects a table, the table grows in size, as shown in image (b) of Figure 4.17.

4.7 Networking

The prototype utilizes Unity's Netcode for GameObjects library to handle networking, enabling synchronization of GameObjects and Scenes across clients. This simplifies the management of the prototype's networking components. The networking architecture follows a client-server topology,

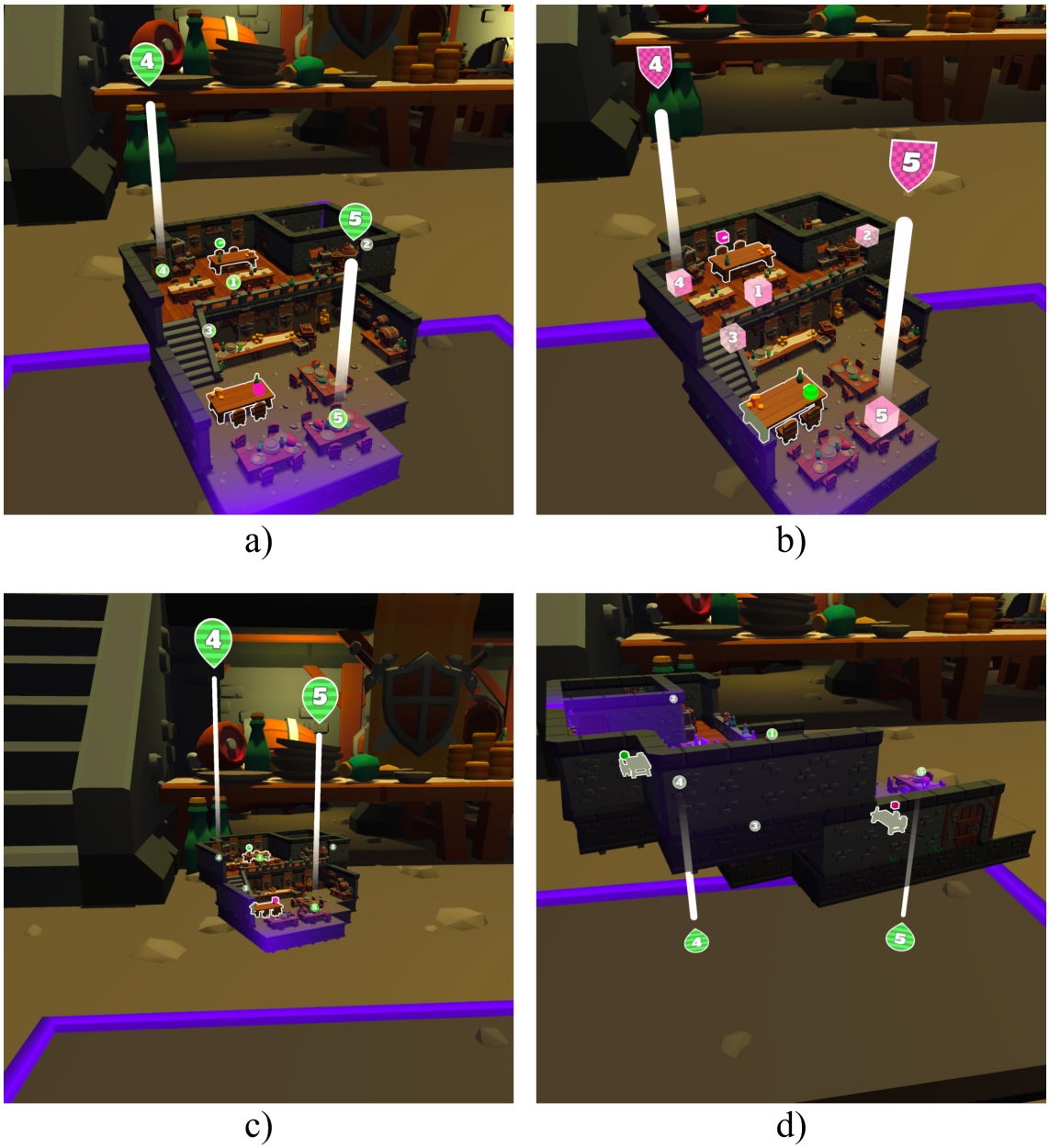
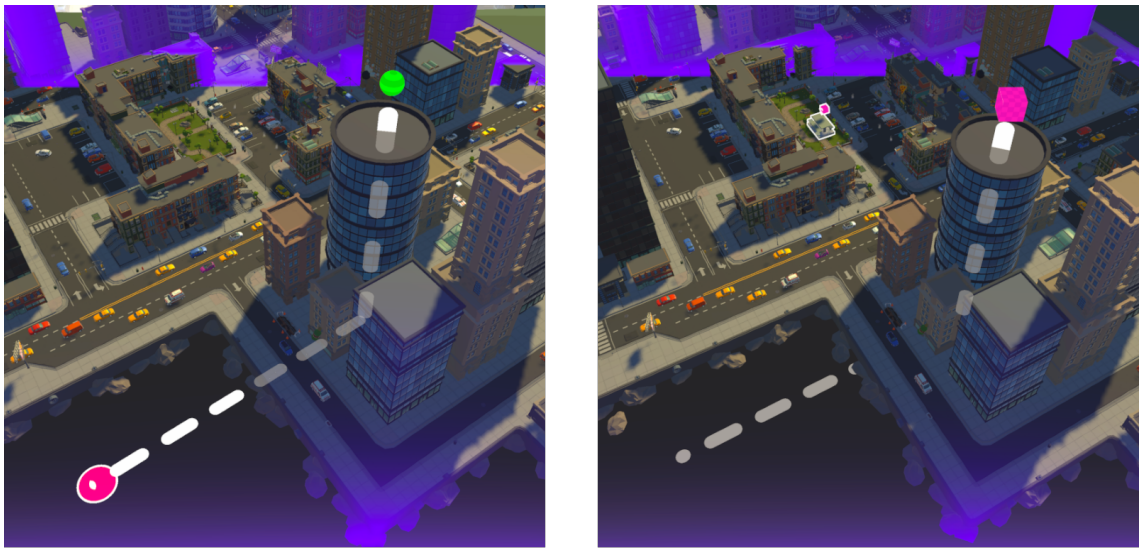


Figure 4.14: Point of interest markers. Image (a) shows markers for the first user, and image (b) shows markers for the second user. Image (c) demonstrates the scaling of the markers with distance. Image (d) displays the markers flipped upside down to ensure they are always visible.

illustrated in Figure 4.18. Specifically, it uses a client-hosted server name a host, where one client is also the server. This architecture was chosen for its simplicity in setup and management, and because it meets the prototype's requirements.

Each client has a user network object for each connected user and a table network object for each table in the scene. A NetworkObject is a GameObject that interacts with Netcode. Before an instantiated NetworkObject is synchronized across clients, it must be spawned. In the client-server architecture, only the server has the authority to spawn and despawn NetworkObjects. By



a) b)

Figure 4.15: Balloon selection helper lines. Image (a) shows the balloon for the first user, and image (b) shows the balloon for the second user with the secondary hand removed.

default, NetworkObjects are owned by the server and tied to its lifecycle. However, user network objects are an exception. These player NetworkObjects are automatically spawned by the server whenever a client connects, assigned to the client with ownership, and despawned when the client disconnects.[16]

NetworkObjects share data through network variables, synchronized with new clients when they join the server and with existing clients when the data changes. Network variables can have different permissions for reading and writing data. In a client-server architecture, the default setting is that the server has read-write permissions, while clients have read-only permissions.[17] Sections 4.7.1 and 4.7.2 describe the user and table network objects in detail.

The server manages the user and table network objects through the User Manager and Table Manager, respectively. These managers store data about connected users and tables, such as the association of user IDs to Netcode client IDs, the current point of interest ID, and the connected user IDs. They also handle the creation and destruction of network objects. Detailed functions of these managers are described in Sections 4.7.3 and 4.7.4.

The managers communicate with network objects through updates on network variables and remote procedure calls (RPCs). An RPC allows methods to be called on objects in another executable. In Netcode, RPCs execute methods on NetworkObjects across clients. A client can call an RPC on the server, and the server can call an RPC on a client. Clients can also call RPCs on other clients, though this passes through the server as a proxy.[18] Section 4.7.5 describes the sequence

[16] https://docs-multiplayer.unity3d.com/netcode/current/basics/networkobject/

[17] https://docs-multiplayer.unity3d.com/netcode/current/basics/networkvariable/

[18] https://docs-multiplayer.unity3d.com/netcode/current/advanced-topics/message-system/rpc/

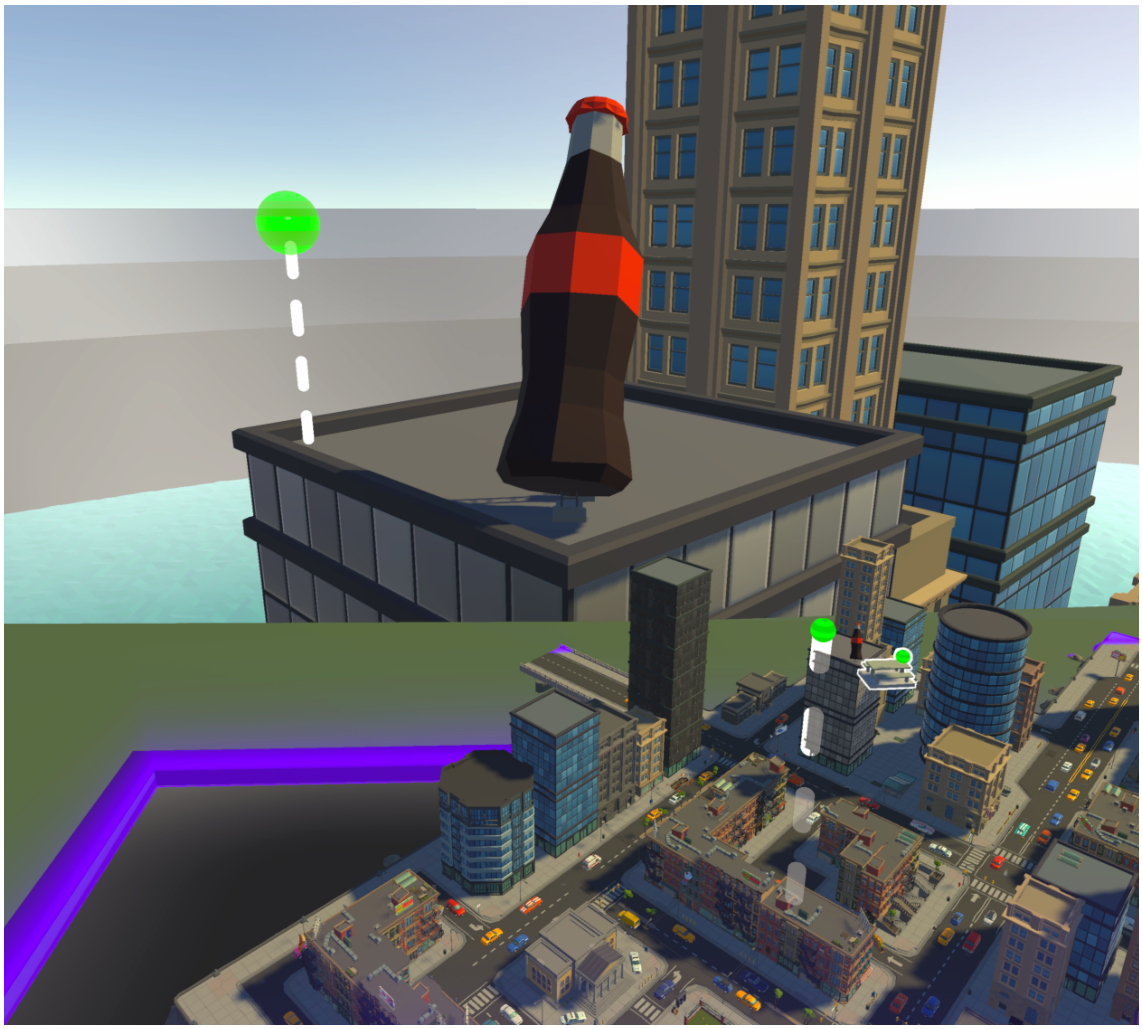


Figure 4.16: Balloon selection visible both in the replica and in the 3D model.

of events during the prototype's execution, including the RPCs used and other network-related events.

4.7.1 User Network Object

User network objects use network variables to store and synchronize data about the users. They have three network variables: user ID, a list of points of interest, and user transform data. The user ID identifies the user and determines their appearance. The list of points of interest contains the ID, position, and user ID of each point created by the user. The user transform data includes the user's position and rotation.

The user ID and points of interest can only be modified by the server, while the user transform data is updated by the client. This design choice simplifies the synchronization of the table tracking algorithm described in Section 4.5. When a user joins, the server assigns the user ID and updates the points of interest list as the user creates or removes points. The client updates the user transform data whenever the user moves.

a) b)

Figure 4.17: Balloon selection intersection with a point of interest in image a) and a table in image b).

When a user network object receives an update to the user ID or when the user spawns, it updates the user's appearance. If the user owns the user network object, it also updates the balloon's appearance of the balloon selection.

When a user creates a point of interest, a temporary point of interest is first created on the client, without an identification number, in both the replica and the 3D model. The temporary point of interest's position and user ID are sent to the user manager through an RPC. The user manager assigns an identification number to the point of interest and updates the user network object's points of interest list. Whenever this list is updated, the user network object determines the type of update. If the user is the owner and a point has been added, the temporary point of interest is updated with the identification number. If the user is not the owner, the replica controller is updated with the new point of interest, and marks it as unacknowledged. If a point of interest is removed, the user network object removes the point of interest from the replica controller. When the user network object spawns, the local client creates the user's points of interest in the replica and the 3D model.

4.7.2 Table Network Object

Table network objects have three network variables: transform data, the user ID seated at the first seat, and the user ID seated at the second seat. The transform data includes the table's position and rotation. Only the server can modify these variables, managed by the table manager. When a table network object is spawned, clients create a table replica in the replica with the defined transform and seat data. When the table is despawned, the table replica is destroyed. Late-joining users acquire all the spawned table network objects and create table replicas accordingly.

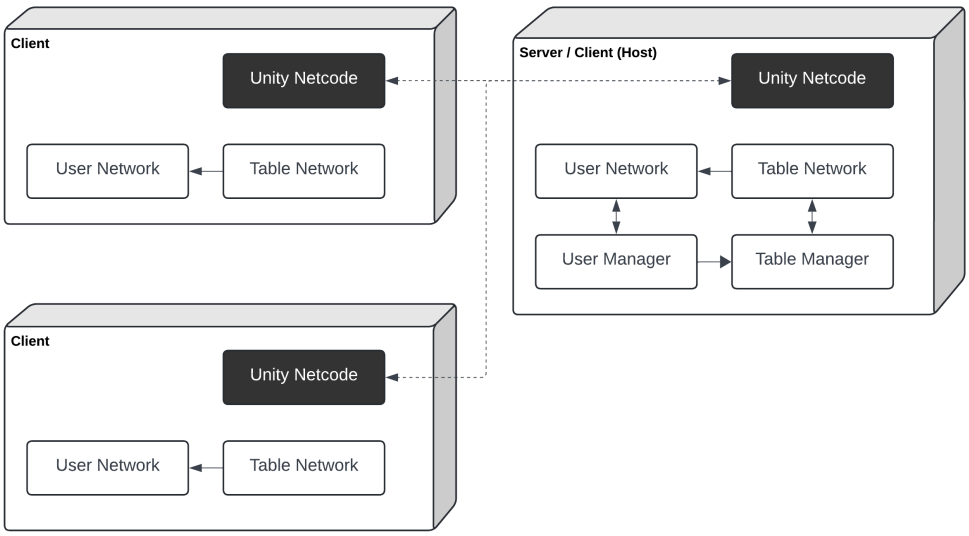


Figure 4.18: Networking architecture, using a client-server topology.

When the server updates the table network object's transform data, clients update the local table and table replica's transform data. If a client is seated at the table, the client moves to the new position using the tracking data, as described in Section 4.5. When the server updates the table network object's seat data, clients update the table replica to reflect who is seated at each seat.

4.7.3 User Manager

The user manager is responsible for tracking and communicating with connected users. It maintains the association of user IDs to Netcode client IDs, the current point of interest ID, and the list of connected user IDs. When a user connects, Netcode triggers the `OnClientConnected` event on the user manager. The user manager assigns a user ID and updates the list of connected user IDs. For this prototype, only two users can be connected, so the user manager has a list of available user IDs, 0 and 1, which it assigns to new users. After updating the user's ID network variable, it requests the table manager to add the player to an available table, as described in Section 4.7.4. Once a table is assigned, the user manager sends an RPC `MoveUserToTableClientRpc` to the user network object to move the user to the table, as shown in Figures 4.19 and 4.21. When a user disconnects, the user manager removes the user ID from the list of connected user IDs and adds it back to the list of available user IDs. It then requests the table manager to remove the player from their table, as depicted in Figure 4.23. The spawning and despawning of user network objects are handled automatically by Netcode.

When a user creates a point of interest, the user's network object sends an RPC `CreatePointOfInterestRpc` to the user manager. The user manager increments the current point of interest ID and updates the user network's points of interest list. Because this list is a network variable, it is synchronized across all clients, and the user network object handles the update accordingly,

as described in Section 4.7.1. When a user removes a point of interest, the user network object sends an RPC `RemovePointOfInterestRpc` to the user manager. The user manager then removes the point of interest from the network object's list.

When a user teleports or joins a table, the user network object sends an RPC `MoveUserToPositionRpc` or `MoveUserToTableRpc` to the user manager. The user manager then instructs the table manager to move the user to the new position or table, as outlined in Section 4.7.4. Afterward, the user manager sends an RPC `MoveUserToTableClientRpc` to update the user network object's position, as illustrated in Figures 4.20 and 4.22.

4.7.4 Table Manager

The table manager oversees the table network objects and assigns users to tables. Its primary functions include assigning newly connected users to available tables, managing user teleportation, handling users joining tables, and removing users from tables upon disconnection.

When assigning a user to a table, the table manager first checks for available seats at existing tables. If a seat is available, the user is assigned to it. If no seats are available, the table manager creates a new table and assigns the user to the first seat. These scenarios are depicted in Figures 4.19 and 4.21. When a user disconnects, the table manager removes them from their table. If the table becomes empty, it is despawned. This process is illustrated in Figure 4.23.

When handling user teleportation, the table manager first checks if the user is alone at their current table. If they are, it moves the table to the new position. If the user is not alone, a new table is created at the new position, and the user is assigned to the first seat. These scenarios are depicted in Figure 4.20.

When a user joins a table, the table manager removes them from their original table. If the original table becomes empty, it is despawned. The user is then assigned to the first available seat on the new table. This process is shown in Figure 4.22.

4.7.5 Sequence of Events

This section outlines a typical scenario that demonstrates the key interactions between users, tables, and the network. The diagrams illustrated only show the interaction between one client and the server at a time, from the perspective of the owner of the depicted user network object. Whenever a network variable is updated, the change is synchronized across all clients.

The scenario begins with a user creating a server, thereby becoming its host. In this case, RPCs are local and executed instantly since the server also functions as a client. The client then connects to the server, as shown in Figure 4.19. The user manager assigns the user the ID 0 by updating the user's network variable and the list of connected user IDs. Upon updating this ID, the user's model is updated to the correct appearance. The user manager then requests the table manager to assign the user to an available table. Since no other users are connected, the table manager creates a new table, Table Network 1, and assigns the user to the first seat. When the table is spawned, it

requests the local client to create the table replica on the WIM. The user manager then sends an RPC to the user network object, moving the user to the table.

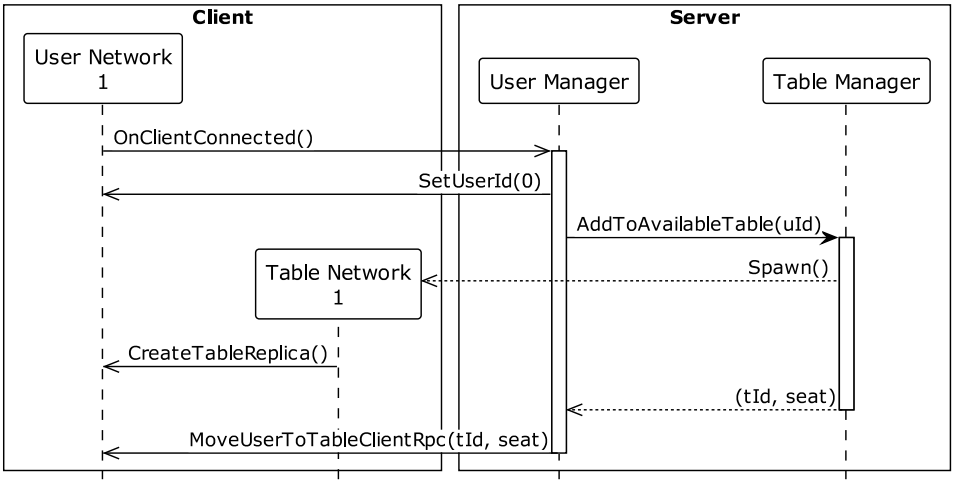


Figure 4.19: Sequence diagram of a user connecting and creating a table.

Next, the user creates a point of interest. Initially, a temporary point of interest is created on the client, followed by sending a `CreatePointOfInterestRpc` to the user manager. The user manager increments the current point of interest ID and updates the user network's points of interest list. When the user network object receives this update, it assigns the identification number to the temporary point of interest, as it is owned by the client.

The user then teleports to a new position. The user network object sends a `MoveUserToPosition Rpc` to the user manager, which instructs the table manager to move the user to the new location. Since the user is alone at the table, the table is moved to the new position, updating its network variables accordingly. The table network object, upon receiving these updates, adjusts the table replica on the client, and moves the user to the new position.

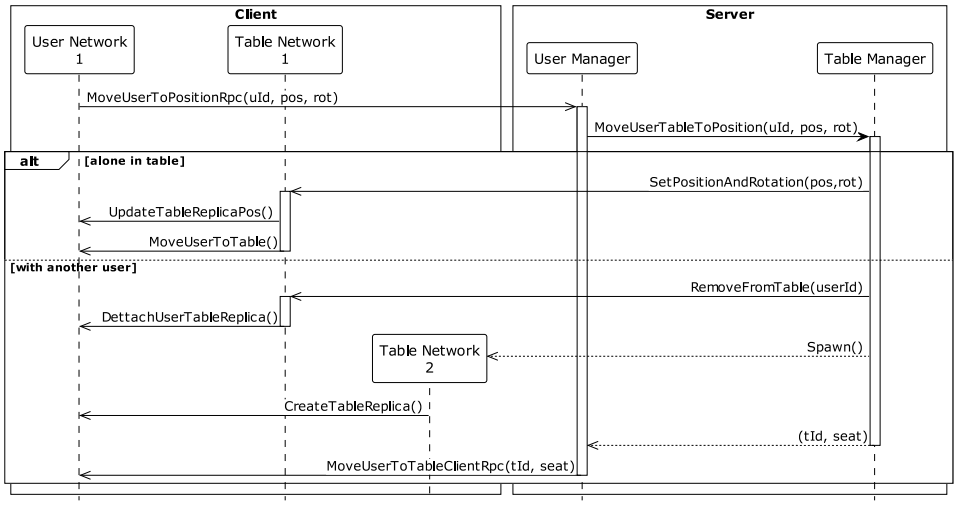


Figure 4.20: Sequence diagram of a user teleporting.

Later, another user connects to the server, as illustrated in Figure 4.21. The user manager assigns this user the ID 1 and updates the list of connected user IDs. Consequently, the user network object is updated with the new user ID and the corresponding appearance. The table manager assigns the new user to the second seat of Table Network 1, updating its seat network variables. The table network object then updates the table replica on each client, displaying the new user in the second seat. Following this, the user manager sends an RPC to the user network object to position the user at the table.

Upon joining, the second user gathers information about the first user's points of interest by loading the point of interest data from the first user's network object, marking them as unacknowledged. Since acknowledgment data is local, no network communication is required. The second user would also load table data from other table network objects if any additional tables existed.

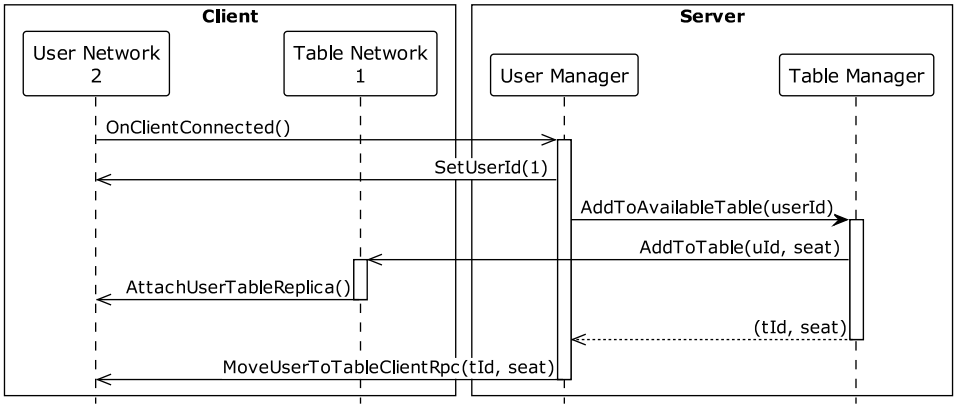


Figure 4.21: Sequence diagram of a user connecting and joining an existing table.

Next, the first user moves to a new position, as depicted in Figure 4.20. The user network object sends a `MoveUserToPositionRpc` to the user manager, which instructs the table manager to relocate the user. Because the user is not alone at the table this time, the table manager removes the user from Table Network 1 and spawns a new table, Table Network 2, assigning the user to the first seat. The new table network object updates the table replica on each client to reflect the user's new position, while the old table network object removes the user from the replica. Finally, the user manager sends an RPC to the user network object, moving the user to the new table.

Following this, the second user joins the second table, as shown in Figure 4.22. Their network object sends a `MoveUserToTableRpc` to the user manager, which instructs the table manager to move the user to the second table. The table manager removes the user from Table Network 1 and assigns them to the second seat of Table Network 2. Since Table Network 1 is now empty, it is despawned. The table network objects update the table replicas on each client accordingly. The user manager then sends an RPC to the user network object, positioning the user at the new table.

Finally, the first user disconnects, as depicted in Figure 4.23. The user manager removes the user ID from the list of connected user IDs and returns it to the list of available user IDs. It then instructs the table manager to remove the user from their table. The table manager removes the user from Table Network 2, and since the table is now empty, it is despawned. The table network

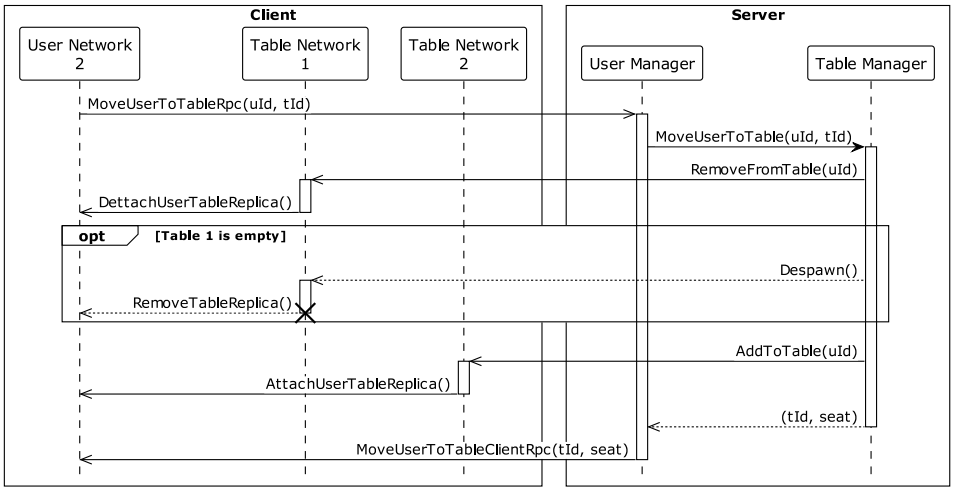


Figure 4.22: Sequence diagram of a user joining a table.

object updates the table replica on each client to reflect the user's removal. The user network object is automatically despawned by Netcode.

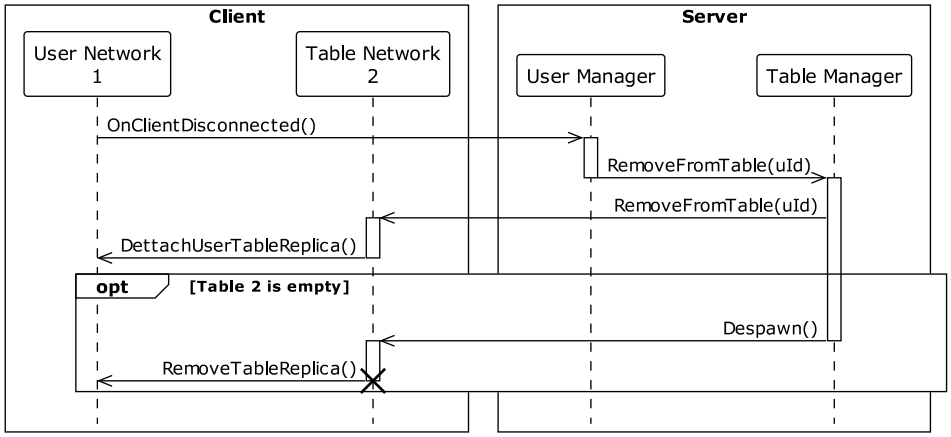


Figure 4.23: Sequence diagram of a user disconnecting.

4.8 Summary

This chapter details the implementation of a prototype for Replico. It starts with an overview of the system architecture and the hardware and software environment utilized, including Unity, Netcode for GameObjects, OpenXR, two HTC Vive Pro 2 headsets, and two multi-touch surfaces.

The chapter continues by describing the state machine used to manage the prototype's states. These states range from the initial state, to `TransformReplicaState` where users can manipulate the replica, and `BalloonSelectionInitialState` where users can create, delete, and acknowledge points of interest, teleport, and join tables.

Furthermore, the chapter details the calculations required to use the multi-touch input to transform the replica. Then, it details the gesture detection mechanisms used, including the utilization

of the K-means clustering algorithm implemented via the ML.NET library for hand detection. Additionally, it outlines the use of VR controllers to track the user's interactions with virtual tables.

The chapter describes the visual indicators implemented in the prototype. These include visual cues such as finger trails on the virtual touch frame, glow effects signaling gesture detection, illumination effects marking the touch frame's boundaries on the replica, and the visual representation of tables, table replicas, points of interest, and their corresponding visual appearances.

The networking implementation is explained, describing the architecture, user and table network objects, user and table managers, network variables, and Remote Procedure Calls (RPCs). The section ends with an outline of a typical sequence of events that unfold during the prototype's operation.

Chapter 5

Evaluation

5.1 Research Questions

5.2 Setup

5.3 Methodology

5.3.1 Test Scenarios

5.3.2 Tasks

5.3.3 Data Logging

5.3.3.1 Objective Data

5.3.3.2 Subjective Data

Chapter 6

Conclusions

This preparatory report for the dissertation explains the significance of fostering collaboration within DeskVR environments. It examines the foundational concepts of collaboration, explores methods to enhance communication in the digital realm, assesses the efficacy of VR in facilitating collaboration, and discusses the advantages and limitations specific to DeskVR.

Through a literature review, three primary themes were encountered. The first revolves around computer-supported cooperative work, investigating how users collaborate using computers. Within this topic, the exploration covered social translucence, workspace awareness, and the spatial model of interactions. The second area focused on concurrency control, which involves techniques used to preserve consistency amidst concurrent updates to objects. Three main approaches for concurrency control were detailed: object ownership, attribute separation, and distributed averages. The third area explored DeskVR interaction techniques, uncovering various methods for interacting with DeskVR environments, such as travel and object manipulation. This exploration provided insights into how to leverage DeskVR's constraints effectively.

The review highlights the potential value of creating replicas to enhance communication and maintain consistency in object arrangements within shared virtual environments in DeskVR. Moving forward, the next steps involve continuous experimentation with technology to refine the proposal, providing a deeper understanding of its requirements. The anticipated challenges encompass designing effective environmental interactions, presenting social information in a comprehensible and unobtrusive manner, and addressing the networking components. These challenges will guide the subsequent phases of research and experimentation.

References

[1] Marilyn Jager Adams, Yvette J. Tenney, and Richard W. Pew. Situation Awareness and the Cognitive Management of Complex Systems. *Human Factors*, 37(1):85–104, March 1995.

[2] Diogo Almeida, Daniel Mendes, and Rui Rodrigues. SIT6: Indirect touch-based object manipulation for DeskVR. *Computers & Graphics*, 117:51–60, December 2023.

[3] Guilherme Amaro, Daniel Mendes, and Rui Rodrigues. Design and Evaluation of Travel and Orientation Techniques for Desk VR. In *2022 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 222–231, March 2022.

[4] Steve Benford and Lennart Fahlén. A Spatial Model of Interaction in Large Virtual Environments. In Giorgio de Michelis, Carla Simone, and Kjeld Schmidt, editors, *Proceedings of the Third European Conference on Computer-Supported Cooperative Work 13–17 September 1993, Milan, Italy ECSCW '93*, pages 109–124. Springer Netherlands, Dordrecht, 1993.

[5] Hrvoje Benko and Steven Feiner. Balloon Selection: A Multi-Finger Technique for Accurate Low-Fatigue 3D Selection. In *2007 IEEE Symposium on 3D User Interfaces*, March 2007.

[6] Ray L. Birdwhistell. *Introduction to Kinesics: (An Annotation System for Analysis of Body Motion and Gesture)*. Department of State, Foreign Service Institute, 1952.

[7] Doug A. Bowman and Larry F. Hodges. An evaluation of techniques for grabbing and manipulating remote objects in immersive virtual environments. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, I3D '97, pages 35–ff., New York, NY, USA, April 1997. Association for Computing Machinery.

[8] W. Broll. Interacting in distributed collaborative virtual environments. In *Proceedings Virtual Reality Annual International Symposium '95*, pages 148–155, March 1995.

[9] Herbert H. Clark. *Using Language*. 'Using' Linguistic Books. Cambridge University Press, Cambridge, 1996.

[10] Florian Daiber, Eric Falk, and Antonio Krüger. Balloon selection revisited: Multi-touch selection techniques for stereoscopic data. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, AVI '12, pages 441–444, New York, NY, USA, May 2012. Association for Computing Machinery.

[11] Yufei Ding, Yue Zhao, Xipeng Shen, Madanlal Musuvathi, and Todd Mytkowicz. Yinyang k-means: A drop-in replacement of the classic k-means with consistent speedup. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 579–587, Lille, France, 07–09 Jul 2015. PMLR.

[12] Alan Dix, Janet E. Finlay, Gregory D. Abowd, and Russell Beale. *Human-Computer Interaction*. Pearson, 3 edition, September 2003.

[13] Christophe Domingues, Frederic Davesne, Malik Mallem, and Samir Otmane. Collaborative 3D Interaction in Virtual Environments: A Workflow-based Approach. In *Virtual Reality*, chapter 3. IntechOpen, January 2011.

[14] Paul Dourish and Victoria Bellotti. Awareness and coordination in shared workspaces. In *Proceedings of the 1992 ACM Conference on Computer-supported Cooperative Work*, CSCW '92, pages 107–114, New York, NY, USA, December 1992. Association for Computing Machinery.

[15] Mica R. Endsley. Toward a Theory of Situation Awareness in Dynamic Systems. *Human Factors*, 37(1):32–64, March 1995.

[16] Thomas Erickson and Wendy A. Kellogg. Social translucence: An approach to designing systems that support social processes. *ACM Transactions on Computer-Human Interaction*, 7(1):59–83, March 2000.

[17] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

[18] Sebastian Friston, Elias Griffith, David Swapp, Simon Julier, Caleb Irondi, Fred Jjunju, Ryan Ward, Alan Marshall, and Anthony Steed. Consensus Based Networking of Distributed Virtual Environments. *IEEE Transactions on Visualization and Computer Graphics*, 28(9):3138–3153, September 2022.

[19] David M. Gaba, Steven K. Howard, and Stephen D. Small. Situation Awareness in Anesthesiology. *Human Factors*, 37(1):20–31, March 1995.

[20] William W. Gaver. Sound Support For Collaboration. In Liam Bannon, Mike Robinson, and Kjeld Schmidt, editors, *Proceedings of the Second European Conference on Computer-Supported Cooperative Work ECSCW '91*, pages 293–308. Springer Netherlands, Dordrecht, 1991.

[21] Saul Greenberg and David Marwood. Real time groupware as a distributed system: Concurrency control and its effect on the interface. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, CSCW '94, pages 207–217, New York, NY, USA, October 1994. Association for Computing Machinery.

[22] Carl Gutwin and Saul Greenberg. A Descriptive Framework of Workspace Awareness for Real-Time Groupware. *Computer Supported Cooperative Work (CSCW)*, 11(3):411–446, September 2002.

[23] O. Hagsand. Interactive multiuser VEs in the DIVE system. *IEEE MultiMedia*, 3(1):30–39, 1996.

[24] Christian Heath, Marina Jirotka, Paul Luff, and Jon Hindmarsh. Unpacking collaboration: The interactional organisation of trading in a city dealing room. *Computer Supported Cooperative Work (CSCW)*, 3(2):147–165, June 1994.

[25] E. Hutchins. The Technology of Team Navigation. In Jolene Galegher, Robert E. Kraut, and Carmen Egido, editors, *Intellectual Teamwork: Social and Technological Foundations of Cooperative Work*, page 552. Routledge, 1 edition, June 1990.

[26] Eunjee Kim and Gwanseob Shin. User discomfort while using a virtual reality headset as a personal viewing system for text-intensive office tasks. *Ergonomics*, 64(7):891–899, July 2021.

[27] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[28] Jun Lee, Mingyu Lim, HyungSeok Kim, and Jee-In Kim. Supporting Fine-Grained Concurrent Tasks and Personal Workspaces for a Hybrid Concurrency Control Mechanism in a Networked Virtual Environment. *Presence*, 21(4):452–469, November 2012.

[29] Jason Leigh, Andrew Johnson, and Thomas Defanti. CAVERN: A distributed architecture for supporting scalable persistence and interoperability in collaborative virtual environments. *Virtual Reality: Research, Development and Applications*, 2:217–237, December 1997.

[30] John M. Linebarger and G. Drew Kessler. Concurrency Control Mechanisms for Closely Coupled Collaboration in Multithreaded Peer-to-Peer Virtual Environments. *Presence*, 13(3):296–314, June 2004.

[31] S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, March 1982.

[32] J. MacQueen. Some methods for classification and analysis of multivariate observations. Proc. 5th Berkeley Symp. Math. Stat. Probab., Univ. Calif. 1965/66, 1, 281-297 (1967)., 1967.

[33] David Margery, Bruno Arnaldi, and Noël Plouzeau. A General Framework for Cooperative Manipulation in Virtual Environments. In Michael Gervautz, Dieter Schmalstieg, and Axel Hildebrand, editors, *Virtual Environments '99*, Eurographics, pages 169–178, Vienna, 1999. Springer.

[34] Susan E. McDaniel and Tom Brinck. Awareness in Collaborative Systems: A CHI 97 Workshop. In *The SIGCHI Bulletin*, volume 29, October 1997.

[35] D. Mendes, F. M. Caputo, A. Giachetti, A. Ferreira, and J. Jorge. A Survey on 3D Virtual Object Manipulation: From the Desktop to Immersive Virtual Environments. *Computer Graphics Forum*, 38(1):21–45, 2019.

[36] Jesper Mortensen, Vinoba Vinayagamoorthy, Mel Slater, Anthony Steed, Benjamin Lok, and Mary Whitton. Collaboration in Tele-Immersive Environments. pages 93–101, January 2002.

[37] Annette Mossel, Benjamin Venditti, and Hannes Kaufmann. 3DTouch and HOMER-S: Intuitive manipulation techniques for one-handed handheld augmented reality. In *Proceedings of the Virtual Reality International Conference: Laval Virtual*, VRIC '13, pages 1–10, New York, NY, USA, March 2013. Association for Computing Machinery.

[38] Donald A. Norman. *Things That Make Us Smart: Defending Human Attributes in the Age of the Machine*. Addison-Wesley Longman Publishing Co., Inc., USA, 1993.

[39] Ohan Oda, Carmine Elvezio, Mengu Sukan, Steven Feiner, and Barbara Tversky. Virtual Replicas for Remote Assistance in Virtual and Augmented Reality. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, UIST '15, pages 405–415, New York, NY, USA, November 2015. Association for Computing Machinery.

[40] Vasco Pereira, Teresa Matos, Rui Rodrigues, Rui Nóbrega, and João Jacob. Extended Reality Framework for Remote Collaborative Interactions in Virtual Environments. In *2019 International Conference on Graphics and Interaction (ICGI)*, pages 17–24, November 2019.

[41] Márcio S. Pinho, Doug A. Bowman, and Carla M. Dal Sasso Freitas. Cooperative object manipulation in immersive virtual environments: Framework and techniques. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, VRST '02, pages 171–178, New York, NY, USA, November 2002. Association for Computing Machinery.

[42] Marcio S. Pinho, Doug A. Bowman, and Carla M. Dal Sasso Freitas. Cooperative object manipulation in collaborative virtual environments. *Journal of the Brazilian Computer Society*, 14(2):53–67, June 2008.

[43] Pei-Luen Patrick Rau, Jian Zheng, Zhi Guo, and Jiaqi Li. Speed reading on virtual reality and augmented reality. *Computers & Education*, 125:240–245, October 2018.

[44] David Roberts and Robin Wolff. Controlling Consistency within Collaborative Virtual Environments. In *Eighth IEEE International Symposium on Distributed Simulation and Real-Time Applications*, pages 46–52, October 2004.

[45] David Roberts, Robin Wolff, Oliver Otto, and Anthony Steed. Constructing a Gazebo: Supporting Team Work in a Tightly Coupled, Distributed Task in Virtual Reality. *Presence Teleoperators & Virtual Environments*, 12:644–657, December 2003.

[46] David J. Roberts, Robin Wolff, and Oliver Otto. Supporting a Closely Coupled Task between a Distributed Team: Using Immersive Virtual Reality Technology. *COMPUTING AND INFORMATICS*, 24(1):7–29, 2005.

[47] D.J. Roberts and P.M. Sharkey. Maximising concurrency and scalability in a consistent, causal, distributed virtual reality system, whilst minimising the effect of network delays. In *Proceedings of IEEE 6th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 161–166, June 1997.

[48] Roy A. Ruddle, Justin C. D. Savage, and Dylan M. Jones. Symmetric and asymmetric action integration during cooperative object manipulation in virtual environments. *ACM Transactions on Computer-Human Interaction*, 9(4):285–308, December 2002.

[49] Roy A. Ruddle, Justin C. D. Savage, and Dylan M. Jones. Levels of Control During a Collaborative Carrying Task. *Presence*, 12(2):140–155, April 2003.

[50] Tony Salvador, Jean Scholtz, and James Larson. The Denver model for groupware design. *ACM SIGCHI Bulletin*, 28(1):52–58, January 1996.

[51] Leon D. Segal. Effects of checklist interface on non-verbal crew communications. Technical Report NASA-CR-177639, May 1994.

[52] Alejandro Jarillo Silva, Omar A. Domínguez Ramirez, Vicente Parra Vega, and Jesus P. Ordaz Oliver. PHANToM OMNI Haptic Device: Kinematic and Manipulability. In *2009 Electronics, Robotics and Automotive Mechanics Conference (CERMA)*, pages 193–198, September 2009.

[53] Adalberto L. Simeone, Eduardo Velloso, and Hans Gellersen. Substitutional Reality: Using the Physical Environment to Design Virtual Reality Experiences. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 3307–3316, New York, NY, USA, April 2015. Association for Computing Machinery.

[54] Gurminder Singh, Luis Serra, Willie Png, and Hern Ng. BrickNet: A Software Toolkit for Network-Based Virtual Worlds. *Presence: Teleoperators and Virtual Environments*, 3(1):19–34, February 1994.

[55] Markus Sohlenkamp and Greg Chwelos. Integrating communication, cooperation, and awareness: The DIVA virtual office environment. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, CSCW '94, pages 331–343, New York, NY, USA, October 1994. Association for Computing Machinery.

[56] Maurício Sousa, Daniel Mendes, Soraia Paulo, Nuno Matela, Joaquim Jorge, and Daniel Simões Lopes. VRRRRoom: Virtual Reality for Radiologists in the Reading Room. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 4057–4062, New York, NY, USA, May 2017. Association for Computing Machinery.

[57] Richard Stoakley, Matthew J. Conway, and Randy Pausch. Virtual reality on a wim: interactive worlds in miniature. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, page 265–272, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

[58] Sven Strothoff, Dimitar Valkov, and Klaus Hinrichs. Triangle cursor: Interactions with objects above the tabletop. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, ITS '11, pages 111–119, New York, NY, USA, November 2011. Association for Computing Machinery.

[59] Un-Jae Sung, Jae-Heon Yang, and Kwang-Yun Wohn. Concurrency control in CIAO. In *Proceedings IEEE Virtual Reality (Cat. No. 99CB36316)*, pages 22–28, March 1999.

[60] FORGY E. W. Cluster analysis of multivariate data : efficiency versus interpretability of classifications. *Biometrics*, 21:768–769, 1965.

[61] R. Waters, D.B. Anderson, and D.L. Schwenke. Design of the Interactive Sharing Transfer Protocol. In *Proceedings of IEEE 6th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 140–147, June 1997.

[62] Richard C. Waters, David B. Anderson, John W. Barrus, David C. Brogan, Michael A. Casey, Stephan G. McKeown, Tohei Nitta, Ilene B. Sterns, and William S. Yerazunis. Diamond Park and Spline:Social Virtual Reality with 3D Animation, Spoken Interaction, and Runtime Extendability. *Presence: Teleoperators and Virtual Environments*, 6(4):461–481, August 1997.

[63] Jeonghwa Yang and Dongman Lee. Scalable prediction based concurrency control for distributed virtual environments. In *Proceedings IEEE Virtual Reality 2000 (Cat. No.00CB37048)*, pages 151–158, March 2000.

[64] F. Yeung. Internet 2: Scaling up the backbone for R&D. *IEEE Internet Computing*, 1(2):36–37, March 1997.

[65] Daniel Zielasko, Marcel Krüger, Benjamin Weyers, and Torsten W. Kuhlen. Menus on the Desk? System Control in DeskVR. In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 1287–1288, March 2019.

[66] Daniel Zielasko and Bernhard E. Riecke. To Sit or Not to Sit in VR: Analyzing Influences and (Dis)Advantages of Posture and Embodied Interaction. *Computers*, 10(6):73, June 2021.

[67] Daniel Zielasko, Benjamin Weyers, Martin Bellgardt, Sebastian Pick, Alexander Meibner, Tom Vierjahn, and Torsten W. Kuhlen. Remain seated: Towards fully-immersive desktop VR. In *2017 IEEE 3rd Workshop on Everyday Virtual Reality (WEVR)*, pages 1–6, March 2017.

[68] Daniel Zielasko, Benjamin Weyers, and Torsten W. Kuhlen. A Non-Stationary Office Desk Substitution for Desk-Based and HMD-Projected Virtual Reality. In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 1884–1889, March 2019.