Designing Student Interactions to Explore Systems Thinking in Augmented Reality

by

Anna Weinstein

B.S. Computer Science and Engineering, Massachusetts Institute of Technology

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2023

Authored by: Anna Weinstein
Department of Electrical Engineering and Computer Science
May 18, 2023

Certified by: Eric Klopfer
Professor, Comparative Media Studies
Thesis Supervisor

Accepted by: Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Designing Student Interactions to Explore Systems Thinking in Augmented Reality

by

Anna Weinstein

Submitted to the Department of Electrical Engineering and Computer Science
on May 18, 2023, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Educational practices are shifting to incorporate new curriculum guidelines and new technologies. At the same time, the field of augmented reality (AR) is rapidly expanding, enabling a new world of opportunities and requiring new approaches to UI/UX design. Incorporating Augmented Reality into classrooms provides a unique opportunity to create engaging, immersive, and transportative learning experiences. The following work explores the intersection of these threads, asking, how do we start designing for student interactions within the augmented classroom. These discussions will be rooted in WIT, a project which is aimed at exploring how headset based AR can be used to teach complex-systems learning in middle school classrooms as well as providing a platform to develop similar experiences. These concepts will also be discussed at a broader scale, first presenting considerations based on the affordances of AR and current education practices, then diving into the technology underpinning these ideas.

Thesis Supervisor: Eric Klopfer
Title: Professor, Comparative Media Studies

Acknowledgments

Thank you to the Scheller Teacher Education Program and WIT teams as well as everyone else who supported me through this project.

Contents

List of Figures

Chapter 1

Introduction

This paper will explore the question - how do we design interactions for the Augmented classroom - under the scope of We're in this Together (WIT) which aims to re-design participatory simulations for complex systems learning using AR headset technology. Why tackle this problem now? The increasing accessibility and capabilities of headset based AR is opening up the possibility of their use as an educational tool. AR allows for immersion into worlds and experiences that may otherwise be inaccessible and provides the potential to visualize complex concepts and ideas while supporting embodied-cognition based learning. The virtual environment can also promote play and exploration by lowering negative consequences of actions and increasing intrinsic motivation. This promise is grounded in studies that have shown simple mobile AR can promote collaboration, autonomy, and general improvements in learning outcomes across disciplines (Schmitz et al., 2015). Some colleges and even some high schools are using textbook QR codes linked to supplementary material (Chiang et al., 2014). Similar approaches with AR features have also been used in museums, outdoor training, and other spaces (Huang, 2017). These mobile apps provide a window into the virtual world rooted in the physical. AR headsets, on the other hand, transform this window into a reality, increasing both the immersion and opportunity for interaction.

Imagine a classroom project investigating water availability within an ecosystem. Instead of working through a paper packet or web game, WIT aims to have the

students engage with the system as if it existed within their classroom. For example, students would be able to directly manipulate and inspect aspects of this system in real time and experience first hand the result of their actions. One student could reach in and shrink the surface area of a lake to reduce evaporation. They could then see the water level dynamically change with this resize as well as a graphic of the water level over time and the change to the projected level. To build this experience, you need three underlying components - a complex system simulator, a multiplayer game network, and interactions between the real and virtual worlds. This work will focus on the interaction lens of this project.

To begin thinking about the User Interface of this project we must first understand who the user is and what they are interfacing with. At the classroom level, we need to handle a socially compact and dynamic environment. At the students level, we need to accommodate 7th-8th grade students with no assumed prior experience with headset based AR. Each student then needs to be able to modify or examine the system as well as communicate with other students. At this point in the WIT project, what exactly these interactions look like remains unconstrained and therefore many different possible configurations need to be considered. This includes: individual students being given full freedom to explore a room-scale experience with communication solely through shared attention, or teams of students assigned to specific tables with modifiable elements constrained to the table itself with communication through virtual graphs or any other combination of physical/virtual configurations.

There are also no constraints on what the system actually looks like or the types of modifications that should be enabled within the system. In general, WIT aims to maintain the broadest scope possible to create the bedrock for further development.

This paper will first discuss the current understanding around teaching systems. Then present questions, considerations, and context through the lens of design practices and educational theories. Then the discussion will focus on systems thinking as a subspace with the potential for a significant impact from the application of AR. Finally, the technological realities of current headset AR will be explored.

Chapter 2

Teaching Systems Thinking

Before diving into the bulk of this work, let's quickly review the current understanding around teaching systems thinking.

The emphasis on teaching systems thinking is relatively new within the US. While teachers recognize its importance they report that they do not currently have the tools to teach it and general curriculum has not adjusted to the new standards. Note, that resources like Loopy or non-tech based lesson plans exist, they just are not commonly used within the classroom as curriculum can be slow to change.

In academia, there is some underlying research on the best ways to teach systems thinking. Most research shows that in order to counteract the default centralized and linear thinking of students, explicit scaffolding is needed to provide tools for examining more decentralized cases. For example, students need to be able to identify the variables and actors that are part of a system as well as how they are connected. These concepts then allow them to explore the more complex emergent behaviors like feedback loops and causality. These connections can be physically mapped and discussed in concrete ways to create mental models. However, the full complexity of a system over time is hard to grasp.

Practice is required to surface and test the assumptions within an individual's mental model. This practice can place the student either within the system or external to it. In the first case, the student is able to take on an agent's perspective and consider how their actions impact the system. In the second case, the student is able

to experience a wider array of levers within the system but lose the direct immersion. In either scenario, the level of complexity is difficult to effectively create within a traditional classroom experience.

This creates the opportunity for AR to not only provide a new interface for learning but directly improve a subset of curriculum that is uniquely setup to align with the affordances of augmented reality.

Chapter 3

Designing for Interactions

In this chapter, we will first discuss the established guidelines for designing interactions and how these guidelines can inform designing for Augmented Reality. This framing highlights the importance of providing feedback for the user to combat both the novelty and possible errors within this new technology space. Then we will discuss how the classroom environment could further inform design decisions.

3.1 Design Principles

Within the general field of human-computer interfaces Many guides, such as Nielsen's Heuristic Principles, Shneiderman's Eight Golden Rules, and Norman's Seven Principles have been established for general HCI. Nielsen's Heuristic Principles breaks down (1) Visibility of System Status (2) Match between system and the real world (3) User control and freedom (4) Consistency and standards (5) Error prevention (6) Recognition rather than recall (7) Flexibility and efficiency of use (Nielson, 1994). The third principle is important to consider in making the virtual feel real, but the second principle is most interesting when considering the unique challenges of AR which breaks down the division of the system and the real world. Instead of interacting with the virtual through an interface, you are attempting to bring the virtual world into the real world.

To understand this, consider that in either purely real or purely virtual worlds,

there is an inherent alignment between interaction and response. If you grab an object, your hand can not physically penetrate the object and naturally aligns with its structure. This behavior can be easily mirrored in virtual reality, by molding the virtual hand to the virtual object. However, when you move to AR you are no longer interacting through your virtual self. You can't achieve the target hand position through tactile feedback or through snapping your hand visualization to the object. This leaves an odd disconnect between the expected behavior of your hand and reality. Adding in a remote, removes this conflict and allows an increased accuracy but it removes the benefits that you gain from natural interaction. So how do we strengthen the tie between real and virtual?

First of all, research shows that good technology is imperative for successful AR (Görlich et al., 2022). That is to say, manipulating a physical object with a virtual overlay is better than manipulating a purely virtual object when and only when the overlay is well aligned. Similarly, manipulating a virtual object with your hand rather than a mouse only shows cognitive benefits if the tracking accuracy is high enough.

Assuming a strong technological base, AR applications tend to focus on two aspects to address the importance of melding the real and virtual worlds

1. Availability of Interaction:

   Clearly visualize what is possible to interact with and how to do so when not naturally aligned with the affordances of an object. These can be embedded or world-based with the objects like highlights for general capabilities, motion or audio queues to draw attention. External or screen based queues can also be used similar to a video game tutorial but these may detract from immersion. The exact balance between helpful cues and distraction needs to be tuned according to the range and quality of interactions within the world.

2. Feedback for User:

   The mapping between a user's gaze or point is also not exactly lined up with what they would expect from the world. This mapping can be made more concrete by adding feedback like a laser pointer or "mouse". Virtual objects

also lack tactile feedback unless using additional technology like a tactile glove. This feedback can be mirrored by physical motion or audio queues that help reinforce the connection between the real action and virtual impact.

These guidelines aim to reduce both the short and long term cognitive load of engagement through explicit feedback. Feedback allows users to learn how to use the system quicker and how to interact with the system more quickly. This is especially important within the educational setting where there likely neither an assumption of previous experience nor the time to walk through a tutorial.

3.2 The Augmented Classroom

So what might AR look like within the classroom from a design perspective?

There are some unique aspects to consider when designing for the classroom as a whole. Unlike the average AR experiences, there will be multiple people within a relatively small shared space. So how do you create a sense of immersion that is not disrupted by others and capitalizes on open space within the classroom? Consider two extremes.

1. Concentrated Interaction:

   In this scenario, the majority of the salient features of the world are either centralized on a table or limited to the extremes of the room. This preserves the structure of the classroom, with free human movement allowed across floor spaces and virtual objects placed in inaccessible areas. Occlusion can then be handled through simple proximity, since there should be no interference between an individual and the table centralized visualizations they care about. This also shifts the level of social interactions by physically forming smaller groups. Note that this concentration of salient features has a down side. If multiple people need to interact with objects within a small space, they will face more conflicts for control. These conflicts could happen in two ways. (1) Person A and Person B want to interact with the scene, but Person A occludes the visuals for

Person B's headset such that hand tracking doesn't work. This can be resolved by making the control points external to the objects through a ray caste or a control panel. Or this could be navigated by limiting the number of interactions a group needs to do at any given time. (2) Person A and Person B both attempt to move an object at the same time. This can be seen as a permissions issue like you would see in a fully online entity. Guaranteeing a shared view of object locations will help prevent conflicting attempts at control since you can see someone actively manipulating the object - making the abstract issue of version control an issue of seemingly physical control.

2. World Embedding:

   In this scenario, the room itself becomes a cohesive simulation with no division or concentration of interactables. This enables students to move freely and interact with the classroom as a whole rather than being filtered into groups. This has the highest potential for the student to act as a cohesive part of the system, enabling them to not just reach into the system but to exist within it. However, it also adds more complex intersections between real and virtual which could detract from the overall experience.

The ideal solution may lie somewhere between these two - balancing the need for immersion, social connection, and a technically feasible experience. Occlusion is also a false binary, you could modify the transparency of objects to allow for both real and virtual input at a given point. However, partial occlusion further separates the virtual and real components which may in the end detract from the immersive nature.

Chapter 4

Educational Theory

Within curriculum design, there is a wide array of considerations. AR affords unique opportunities to capture both the collaborative aspects of in person classroom activities and the interactivity of a more digital medium. In this chapter, we will explore the importance of physical embodiment and collaboration and how these lenses can inform our design.

4.1 Learning and Motion

Based on both neurological and behavioral evidence, the theory of embodied cognition suggests that perception and action are inherently linked. Learning and understanding abstract concepts depend on the physical body and its interactions with the world. In a cross sectional review, integrating the physical body in learning activities was shown to have significant benefits across multiple domains and age groups (Fugate, et. al, 2018). It is important to note that this theory applies to both task-related and non-task-related embodiments. Conceptually congruent gesturing - gestures which directly tie to a concept - whether explicitly explained or not can improve learning outcomes and other cognitive measures (Kang, 2016). More general motion provides benefits related to both engagement and learning outcomes as well (Johnson-Glenberg, 2017). So any level of embodiment - interaction, motion, and physicality - has possible learning benefits by forming connections between motion

and abstract concepts but incorporating and understanding effective embodiment is difficult. Since the majority of headsets do not support dynamic object detection, we must maximize embodiment with the assumption of no physical objects.

Tangibility within UI designs has been shown to enhance immersion as long as there is strong tracking alignment and low cognitive load (Bozgeyikli & Bozgeyikli, 2021). This caveat results in the majority of positive examples relying more heavily on marker (QR code) based tracking which allows the virtual overlay to more efficiently track its real counterpart. These examples also use the higher fidelity object tracking to reduce the need to track the hand motions themselves. These capabilities are publicly available through Vuforia and MergeCube however there is not enough support for the true tangible object in AR headsets which tend to use markerless tracking and rely exclusively on controls or hand tracking to move virtual objects. Interacting with purely virtual objects does have an upside. Without a physical counterpart, virtual objects are able to move independently, creating a potentially more engaging and dynamic scene. In addition, one study found, participants preferred fully virtual objects to augmented objects when there were errors in the overlay alignment (Jeffri & Rambli, 2020).

4.1.1 Tangible UI Examples

There are a couple levels of augmentations that we can look at for inspiration which build in complexity.

- AR Cards (2d):

  The most common example of TUI in AR are these simple Cards which populate virtual content above them (Diaz et al., 2006). They have been used for games, educational content, and even interactive business cards. The cards act as a simple marker to anchor the corresponding objects. The image series above comes from one exploration of this context specifically looking at collaboration. Card sets have also been developed focusing on more individual exploration of technical content like those produced by Getting Nerdy with Mel & Gerdy

Figure 4-1: An example of an AR Card based Education activity from Getting Nerdy with Mel & Gerdy

('Getting Nerdy", 2019).

- AR Cubes (3d):

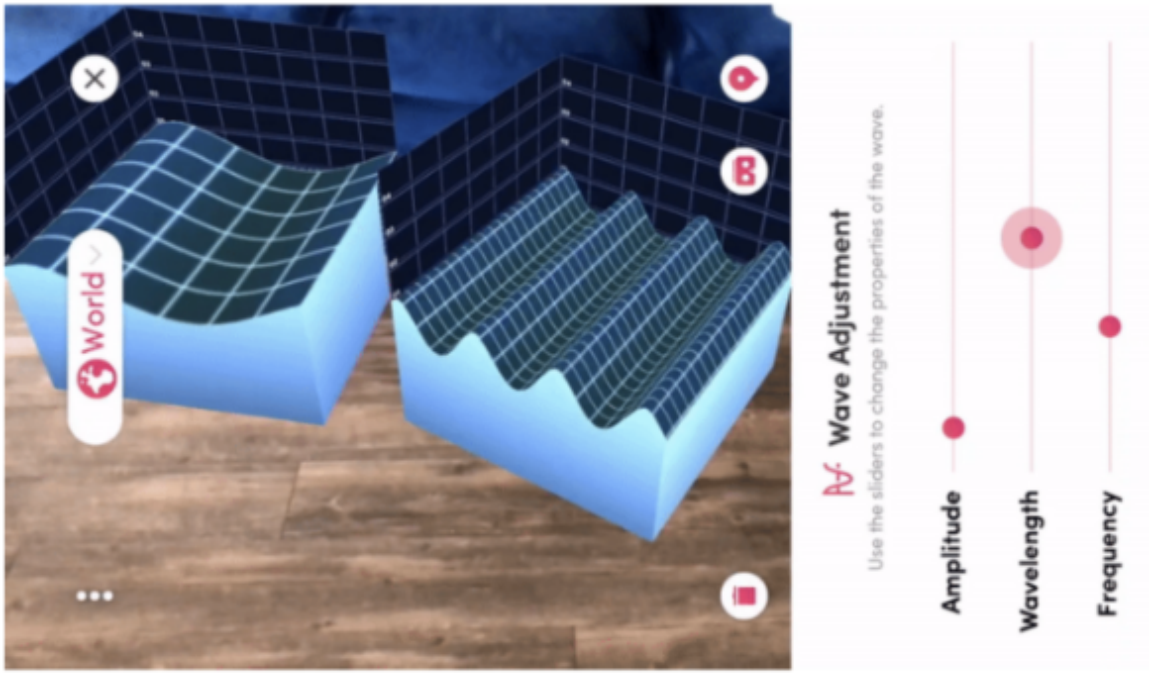


Figure 4-2: A sample app from MergeCube sxploring wave patterns

The underlying functionality of AR Cards can then be used to construct a 3D object like Mergecube, where AR overlays a physical object and dynamically responds to its manipulation (MergeCube). Each face of the cube is covered with a distinct high contrast pattern, allowing virtual content to be accurately

anchored. Apps have been developed within the education space based on this technology, showing promising possibilities for fostering engagement.

This technology is able to work with multiple cubes as long as those cubes are relatively stationary. For single cubes, the object can be physically turned to explore or modify the virtual overlay.

- AR Spheres (3d):

  Just last year, a lab created a globe that could be physically manipulated while

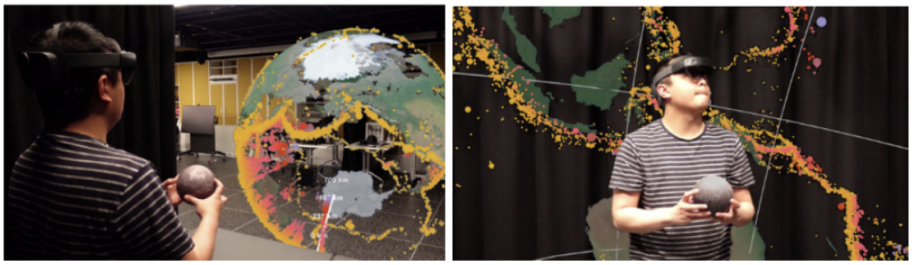


Figure 4-3: Visualization of exocentric and egocentric world views from "Tangible globes for Data Visualisation in augmented reality" (Satriadi et al., 2022)

displaying virtual content in HoloLens2 (Satriadi et al., 2022) . This exploration proposes a series of unique ways to directly interact with a tangible interface, essentially transforming the sphere into the simulation itself.

In addition, they explore various ways to less directly control a simulation, using the sphere as a control rather than the full simulation. They explored both an exocentric and egocentric view, which mirrors well to the goals of this project. Due to the added complexity of the sphere they had to add extra features to enable sufficient tracking (Satriadi et al., 2022). Many similar projects have used HTC VIVE trackers inside a transparent sphere. However, this implementation embedded infrared LED's to allow for a more realistic tangible sphere. These hidden active tracking markers operate in a similar manner to the QR codes used in the cards and cube discussed above.

Each of these examples provides interesting ideas, however, this project aims to create a design that is accessible for schools specifically using headset based AR. Without

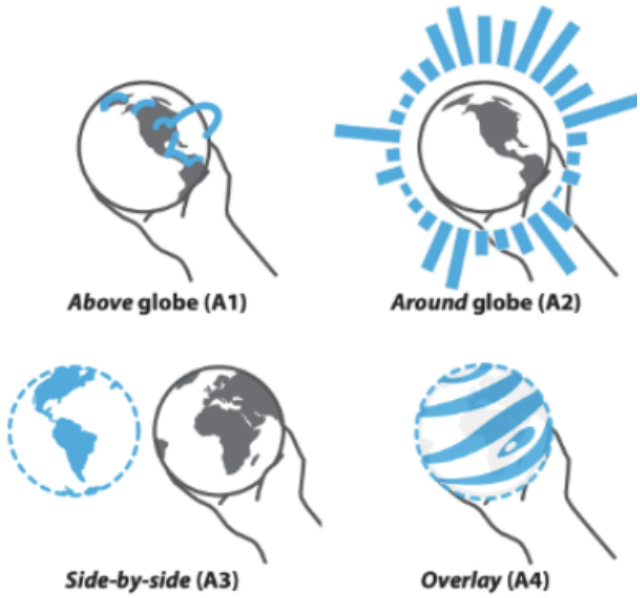


Figure 4-4: Visualization of various overlay options for physical content as in "Tangible globes for Data Visualisation in augmented reality" (Satriadi et al., 2022)

modification, the currently available AR headsets do not provide support for the dynamic object tracking that is needed to create similar functions. Instead, we need to figure out a way to create similar effects without the integration of a physical object.

4.2 Learning and Collaboration

Social interaction improves learning outcomes, especially for tasks involving conceptual rather than procedural knowledge (Plass, et.al, 2013). Collaboration specifically has shown more positive benefits than competition, by allowing for cross-cueing, complementary knowledge, error-correction, attention management, observation learning, and a number of other theorized aspects (Nokes-Malach, et. al, 2015). The benefits of collaboration can be achieved in fully virtual text based mediums (Hostetter, 2013). So while there could still be some variations with how personal interactions are perceived without the ability to see the actual face of the person they are talking to when in a headset, the negative impact should be minimal. However, this benefit is not

universal. The most common downside referenced in digital studies, is the limitation to a shared screen, control point, or method of interaction. AR inherently addressed this issue but allowing each individual to be a distinct actor within their world. For collaboration in general, the emergence of a leader impacts the effectiveness of social interactions in increasing learning (Johnson-Glenberg, et al., 2014). Collaboration requires that each individual has some level of individual responsibility. It also requires effective communication to bridge the gap between the individual and group. The introduction of a leader influences each of these components.

A successful social dynamic can be built around five components as proposed by Johnsons.

- Accountability: Individual accountability and responsibility towards the larger goal
- Positive interdependence: An individual's success is tied to the success of the group (Laal, 2013)
- Frequent interaction: Student's should be frequently engaging with each other.
- Interpersonal skills: opportunities for leadership, decision making, conflict resolution and other social skills
- Group Processing: Group accountability for functioning as a collective.

Structured conversations, visualization tools, and external support can help further reduce the cognitive demands of social congruence.

Linking back to systems thinking, the fine tuned balance of individualistic, competitive, and collaborative components is already present. No single actor is able to achieve an intended result in isolation and if feedback loops are effective also can not fully out compete other actors. This breakdown can be used to create roles for each student at the actor or system level which tread the line between specialization and generalization. Allowing for both unique ownership over a part of the problem as well as collaboration through shared experiences. The Johnson framework above yields the following questions for consideration.

Accountability: What level of processing allows individuals ownership without overwhelming them? What level of system interaction allows a user to still feel like their actions are meaningful? Interdependence: How does each role relate to the others? Are there any points of information or action items that can be combined across roles? Frequent interaction: How do you maintain a shared world view despite the unique variations needed to indicate role specific actions? Interpersonal skills: How is information shared between roles in a way that drives team wide decisions and discussions? Group processing: How does the system provide feedback on how well the team is doing overall? At what scale is group collaboration important?

Chapter 5

Technical Exploration

For the WIT Project, we wanted a headset with color AR capabilities, as well as hand tracking capabilities and some level of room awareness.

The following work explores the current state of technology and the corresponding affordances and limitations applied to this framework. This exploration as well as the considerations presented thus far, then build towards one possible proof-of-concept demonstration for graphical representations, shared worldviews, direct and indirect modes of interaction, and variability in visibility and intractability.

5.1 Available Technology

5.1.1 Headset Options

At the start of this project, the goal was to develop for the broadest set of devices possible. So we began by exploring the different software and hardware options. Due to the goal of WIT, we needed a headset that had the ability to create AR experiences ideally using passthrough. We also wanted to prioritize some level of hand tracking over only using controllers. The choice of headset was then primarily driven by cost and availability within the time frame of this project. The Varjo, MagicLeap, HoloLens, Pico 4.0, Lynx, and HTC Vive were considered. However, the Meta Quest Pro was ultimately chosen for development.

To see the full table of headsets and a breakdown of some of their capabilities see Appendix A.

5.1.2 Software Options

While we were deciding on and waiting for the headsets to come in. I also conducted some exploration into the various SDK's available that could support interactions within our project. This could either be handled by using dynamic object tracking and interactions with physical objects or by hand tracking and interaction with purely virtual objects.

	Feedback	Interaction
Direct	Awareness of proximity or initial collision detection (hover)	Reaction to surface contact (press) Parented transform (grab) Parented scale (2 hand grab)
Indirect	Raycast intersection (point)	Hand Gesture Trigger (pinch)

Figure 5-1: A basic breakdown of interactions with virtual objects

Ideally, the technical demo would be built using packages that were compatible with multiple devices. However, there was not a general package that was also compatible with the Oculus Quest and had the desired capabilities. See Figure 5-2 for some of the available platforms as well as their capabilities.

There were no options to reasonably enable object tracking within our chosen headset, which limited the options for the eventual demo. However, the increasing availability of headset based AR leaves space for the possibility of object tracking that is seen in mobile AR to become available in the future. This hope, as well as the desire to mirror the impact of augmenting physical objects motivated the exploration of mobile based AR in the next section.

	Capabilities	Supported Platforms
AR Foundation	Plane detection, Object Tracking, Hand Tracking, Occlusion	Unity
AR Core	Plane detection, Augmented Images	Android, iOS, Unity, Web
MRTK	Hand Interaction • Hold, Move, Pose	Windows Mixed Reality devices (Hololens)
Vive	Hand Tracking and joint visualization • Left/Right label Gesture Detection • Point, Fist, Okay, Thumbs up, open palm, peace	Vive headsets, Android Phones
ManoMotion	Skelton Tracking, Hand Segmentation Gesture Analysis • Grab, Pinch, Point Object Anchoring relative to skeleton	Android, iOS mobile platform Unspecified headsets
Vuforia	Object Tracking and 3D augmentation Interaction through occlusion	ARCore, ARKit, MagicLeap
Unity XRI	Hand Interaction • Hover, Select, Grab, *Poke **added in Feb 2023 update*	Meta Quest (Oculus), OpenXR, Windows Mixed Reality,
Meta OVR	Hand Interaction • Hover, Select, Grab (1 and 2 hand), Poke, Press, Ray, Snap, Hand Pose	Meta Quest (Oculus)

Figure 5-2: Various SDKs or Platforms considered for tech demo

5.2 Mobile Development

ManoMotion

The first series of technical exploration focused on the quality of hand tracking for AR, using ManoMotion. This package was chosen since it is used within industry standard mobile AR applications and also advertises possible integration with headsets. ManoMotion's developer track and sample app provide explicit access to joint location and orientation, continuous and discrete gesture detection, as well as other metrics shown in Figure 5-3.

This visualization allowed for initial exploration of hand tracking quality. Since it's essentially built for mobile use it did not handle multiple hands as well as a headset is able to. The most notable feature from this testing was how much the algorithm struggled with different hand angles. As you would expect, a fully visual palm or back of hand with full fingers is easily registered. A fist with knuckles visible, the profile of a hand, or the palm with fingers obstructed is also fairly robust. However, as soon as the center of the hand is occluded, even with the wrist and fingers visible the hand

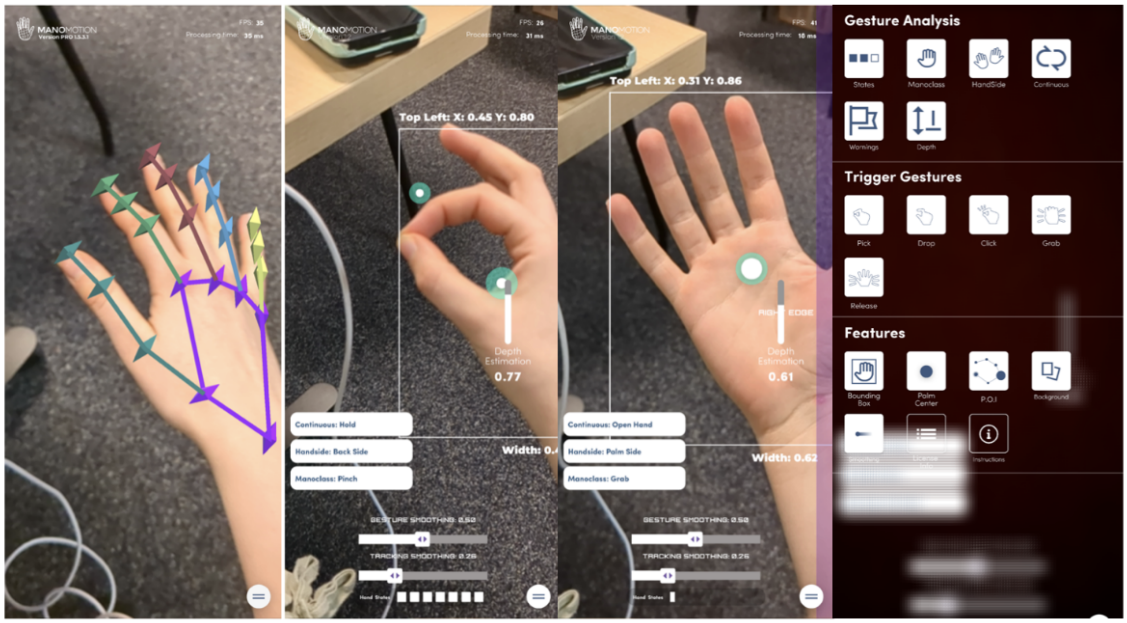


Figure 5-3: The visuals provided within ManoMotion's example app

is not detected. See Figure 5-4 for a visualization of these different configurations.

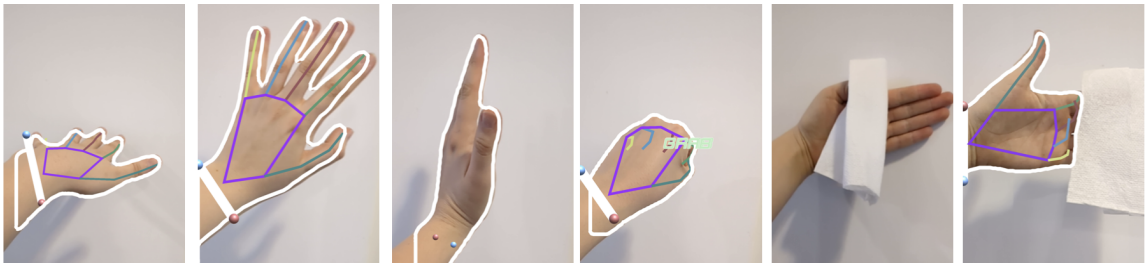


Figure 5-4: ManoMotion and the limitations of hand detection.

This observation highlights an issue that remains even within the headset. Hand detection is fundamentally limited by the visibility of the hand. Note that in object tracking you are able to use extended tracking to handle loss of visibility. However, the same algorithms can not be used for the unpredictable motion of hands. This constraint may inform the design of a virtual world in two ways. First, interactions should limit gestures where the hand is rotated in ways that are difficult to track. Second, if a hand interaction has a visual effect, that visual should be able to be seen within the same frame as the interaction. This constraint will also help the user form connections between tools and their impact.

ManoMotion also provides an example of overlaying or anchoring virtual content onto a hand with a corresponding demo - https://www.manomotion.com/ring-try-on-tutorial/.The work put into recreating this demo provided valuable learning on how to develop a project from Unity and use the information provided from hand tracking to build functionality. The demo also helped guide my perspective when talking about designing tools or character overlays for the WIT project in general. It provided an explicit example of how delays in tracking can impact the feel of a virtual overlay since different tracking confidences could be explored through different hand rotations.

Vuforia

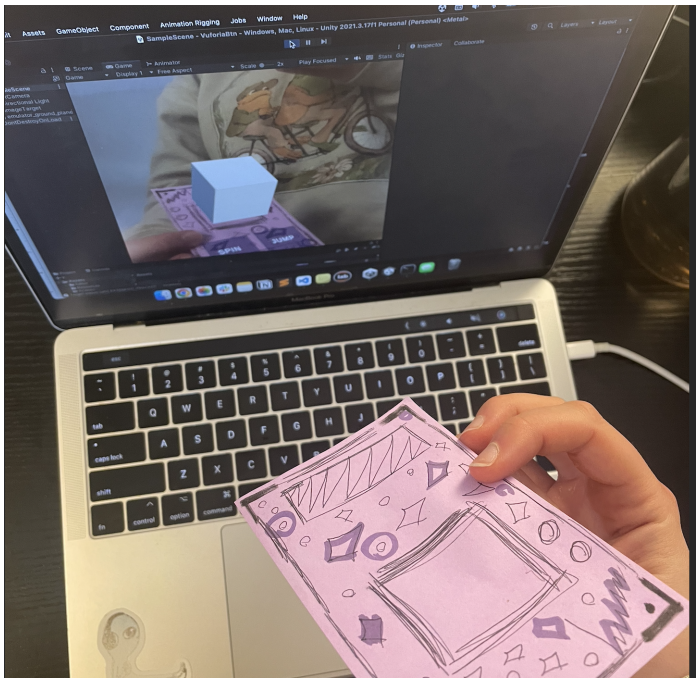


Figure 5-5: Something

Next, I looked into marker based anchoring and object tracking for virtual overlays. For a simple look into this topic, MergeCube provides some basic apps with a printable cube. However, I chose to use Vuforia for this exploration since it was compatible with more headsets and provided the most developer capabilities. Vuforia provides support for 2d cards as well as more complex shapes. Unlike other plat-

forms which autodetect a small subsection of objects, the developer is responsible for defining all aspects of these objects. For the development below, I used a simple 2d image, since the Vuforia demo apps allowed me to experience 3d shapes and I mainly wanted to focus on how to create the overlay and some simple experiments. Iit is important to note, however, that the 3d objects were significantly more intuitive and entertaining than the 2d objects since they felt like a physical object rather than a simple anchor.

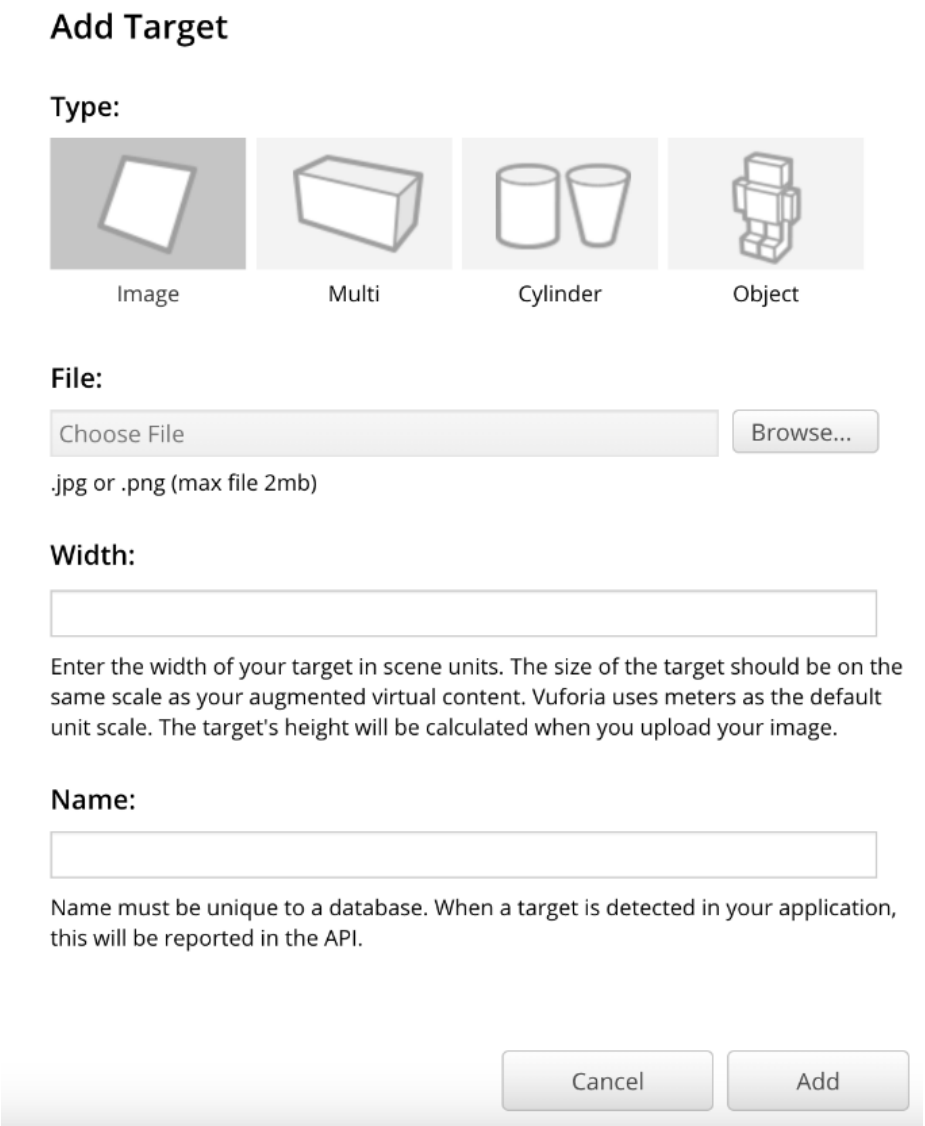


Figure 5-6: Vuforia allows for the creation of a simple 2D marker as well as cubes, cylinders, or more complex models. On creation a target with and name are set that then influence how these targets appear in Unity.

The first stage of this process was simply to set up the markers for use in Vuforia. Unlike other platforms, Vuforia does not limit you to a classic QR Code. Instead, you can use any image with enough features, opening space for design of even the physical target itself. This step is done through their web interface and then the resulting object is imported into Unity. You could also directly upload images to Unity but you lose out on some capabilities. The web interface allows you to see which components are actually being used for tracking. I explored how different shapes, line weights, and target materials influenced the tracking. Figure 5-7 shows the detected features for each of the targets I created.

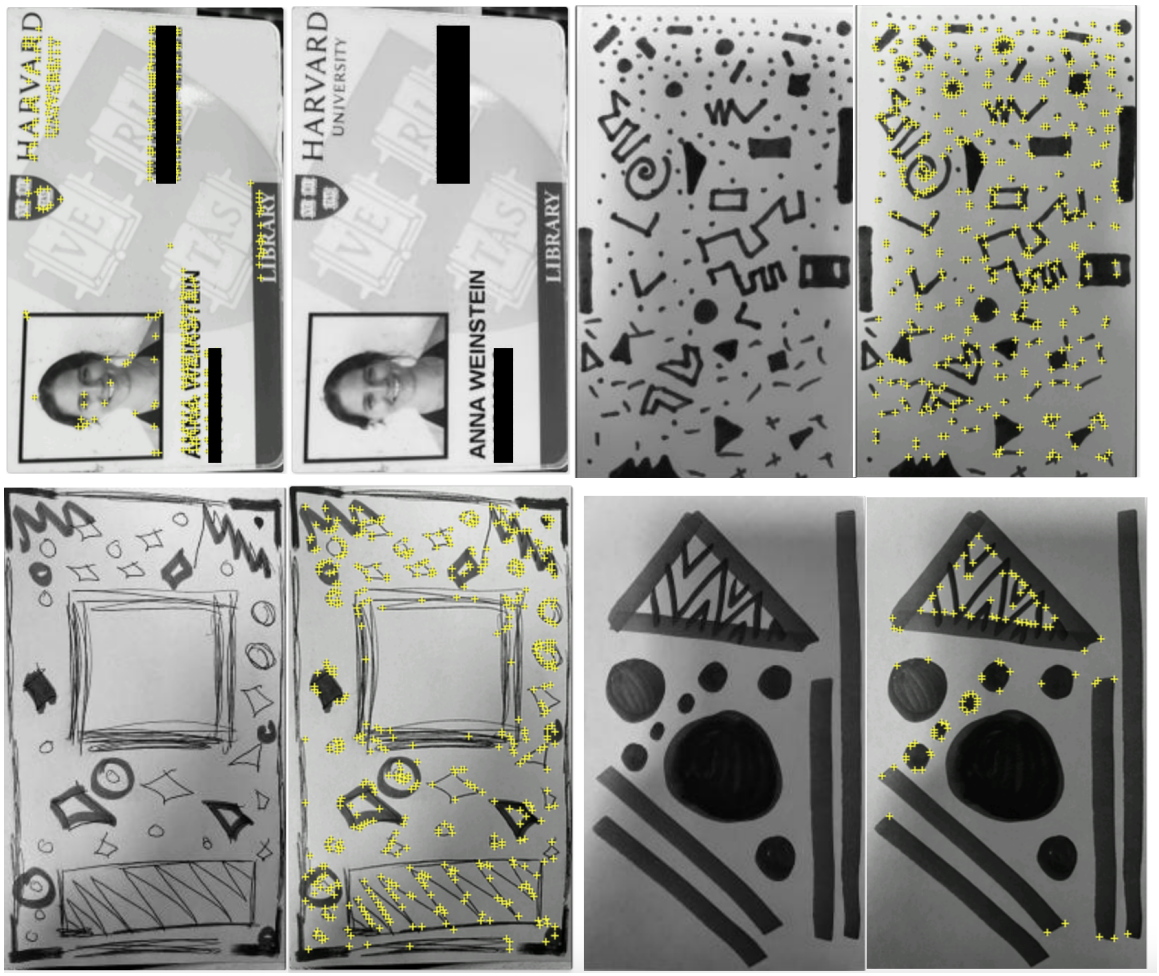


Figure 5-7: Key features detected within a series of Vuforia image trials

To use these anchors I then created a simple AR scene within Unity and created a GameObject "Image Target" to handle the image tracking and corresponding virtual

overlay. I then added 3 scripts

1. Vuforia Image Target Behaviour: Defines the target that this object should track. It allows for only one target image but is able to track multiple instances of it. This script also defines the expected size of the image to improve tracking.

2. Default Observer Event Handler: Allows for additional functionality to be built around the detection of the image target.

3. Image Target Preview: visualizes your image target in scene to allow you to create overlays relative to features on the target.

I added a simple cube as a child of the target to serve as an example of a "virtual overlay". Then two Vuforia buttons to allow for interaction beyond movement of the object itself. The buttons are fairly simple in terms of interaction fidelity, using percent occlusion to determine whether a button is being "pressed". However, they allowed me to trigger behaviors like motion of the virtual object relative to the image anchor. These two modes of motion - through movement of the anchor and through interaction with a button seemed like an important duality to explore.

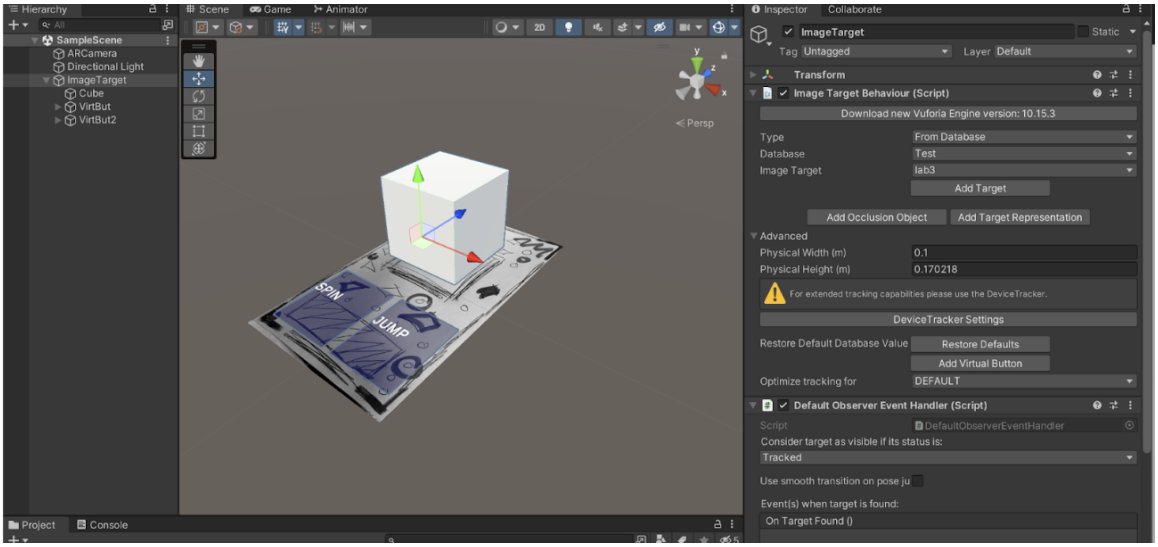


Figure 5-8: The unity setup for a simple Vuforia marker application

I also explored various forms of animation and movement to see how they impacted the perception of reality or animacy. This exploration included having the object

appear as soon as the target was in view, automatically on button press, or animated growth on button press. It also explored linear motion using arrow keys, linear translations, and rotations. Rotation and scaling proved more robust in this setting since depth was a little difficult to process on a screen. This is less important but still noteworthy when moving into headset based AR.

I finally explored interactions between virtual objects anchored to separate targets. Essentially you could use the arrow keys to create then drive a car to collect a cube on another target. This demo was a mirror of one created by Ellen to explore spatial anchors using ARCore. The most surprising takeaway from this exploration was the difficulty of resolving local and global space especially relative to interactions. This conflict will be explored more at the end of this paper.

Chapter 6

Headset Development

As previously mentioned, the Oculus headsets are not compatible with the majority of outside packages or SDKs. It does support Unity's XRI, however XRI did not support the basic features we wanted to use at the time of this project. Instead, we used OVR - Oculus built in SDK which integrates with Unity. OVR provides access to controller and hand based interactions broken down into grab, ray cast, poke, and gesture detection. Depending on the version of OVR you use, the overall setup of the scene varies slightly as well as how the hands are visualized within the scene.

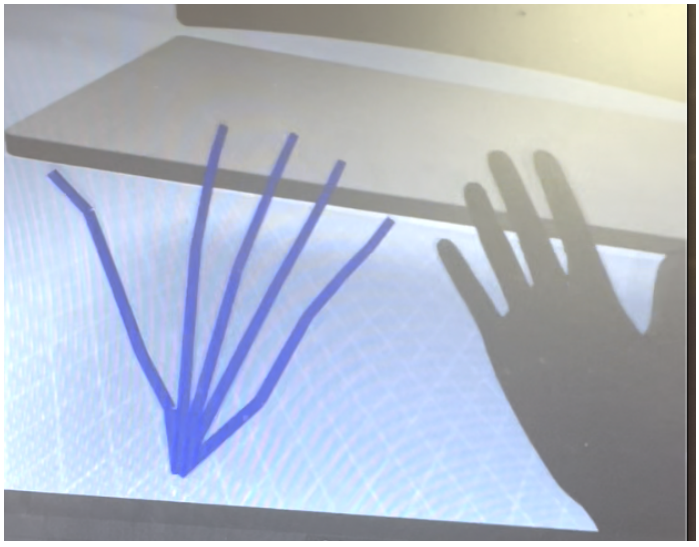


Figure 6-1: The most basic visualization of the OVR skeleton and hand mesh within VR

However, the basic information available and the setup of interactable objects

remains fairly constant. The first step in development was to get the headset to detect and visualize the user's hands. In virtual reality these visualizations are obviously crucial but they also serve an important role within the augmented scene. The slight overlay includes visualizations of when an action is triggered by showing a change in color of the fingertips.

Once the hands were set up for interaction, I explored the various options for interactables using the built in VR sample scenes. This exploration informed the decisions made throughout the demo creation. The most important observation, however, was the confirmation of the importance of physical feedback observed when using the virtual spray bottle. This interactable was set up for a palm grab and was large enough that if the object was real, your hand would not fold back on itself. However, when grabbing this object I would fully close my hand. The hand pose on the right, which attempts to align with the virtual representation, was incredibly uncomfortable. This is important because it leads to a discrepancy between the virtual and real hand as seen in Figure 6-2 When moving to Augmented Reality, this "hand snap" is no longer an option for increasing the reality of interactions.

Figure 6-2: Difference between real and virtual hand position when grabbing a virtual object

6.1 Local Interactions

6.1.1 Pokeable Objects

This type of interaction lets you interact with the surface of an object creating a cue for both a hover and an actual poke selection. You can modify the distance that a hover or poke occurs at and use the start or end of a hover or poke as a trigger.

- Button:

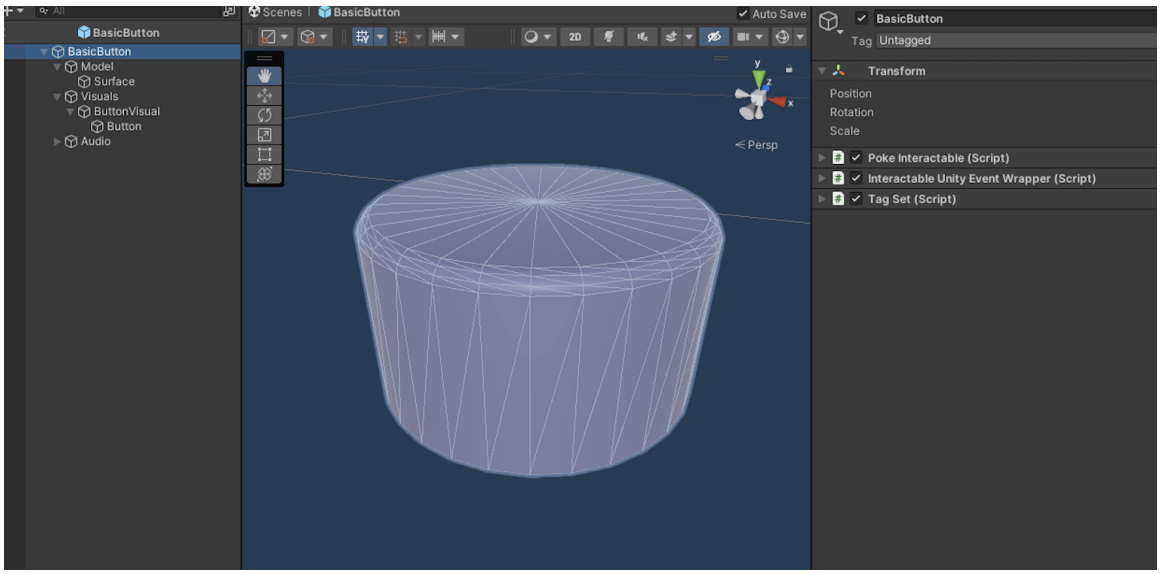


Figure 6-3: The hierarchy of an OVR based button within Unity.

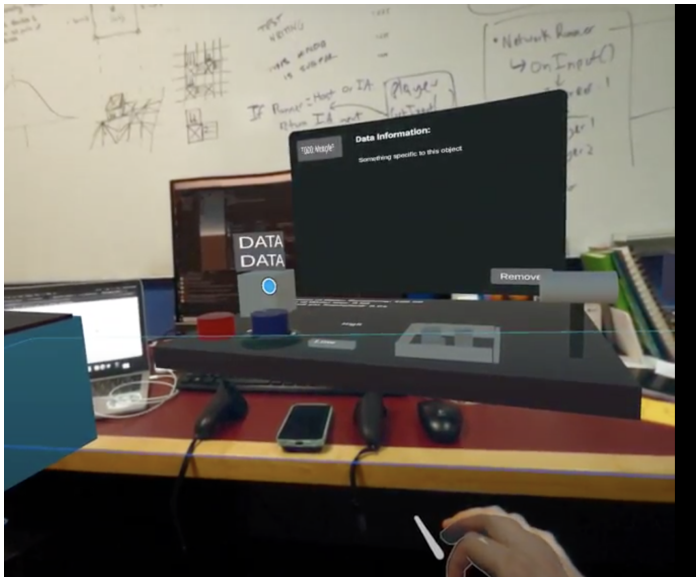
This prefab uses the poke interaction to cue two triggers - an on press, and an on release. Within the first iteraction of the demo, this cue is used to reset the water level.

The importance of feedback is highlighted in this object. In the final version, the hover, the press, and the release are all noted by either audio or visual components. On hover, the color of the button changes to denote it is pressable, on contact, it moves with your finger to press down, until its lowest point where it *clicks*. On release, the button then returns to its initial state.

6.1.2 Ray Cast

This allows you to interact at a distance, including selection through a pinch gesture. OVR builds in the visualization of interactions between the ray cast and the pointable surface.

- Inspectable:



This prefab displays a simple data preview window when hovered over and then posts its full data to a graph visualization when it is selected (See Appendix C for more technical detail on the graph creation.). This is done on top of the grab interactable, allowing for two levels of interaction without interference.
At a distance and on small objects this interaction can be hard to pinpoint since it does not align well with the actual point. So if there aren't any pointable objects near your target, you are left without a visualization.

6.1.3 Grabbable Objects

- Placeable Item: This allows an object's translation, rotation, or scale to be directly modified according to the movement of the hand or hands grabbing the objects. Based on the exact setup each of these functions can be limited across certain axes.

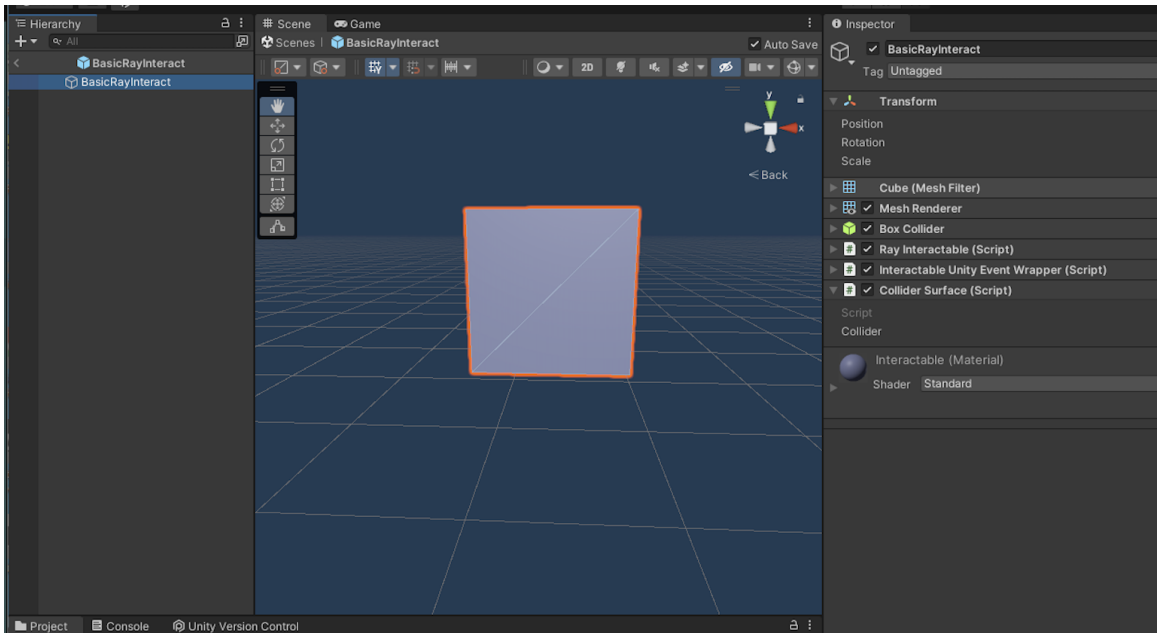
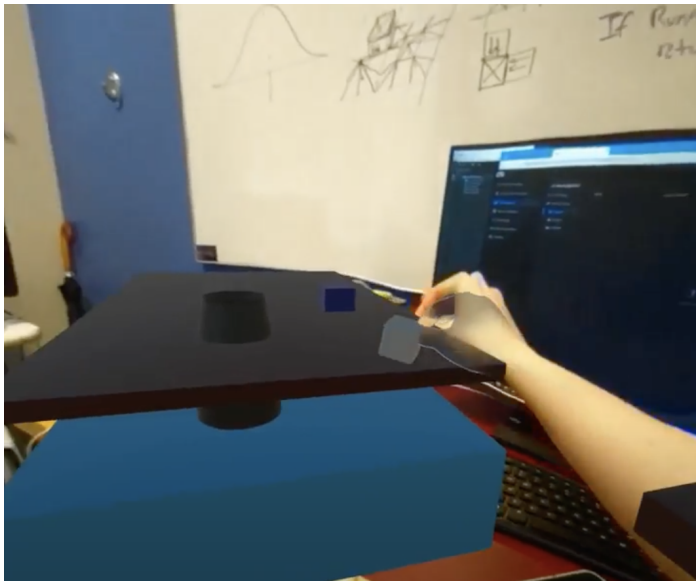


Figure 6-4: The Unity components required to for an object to be intractable at a distance

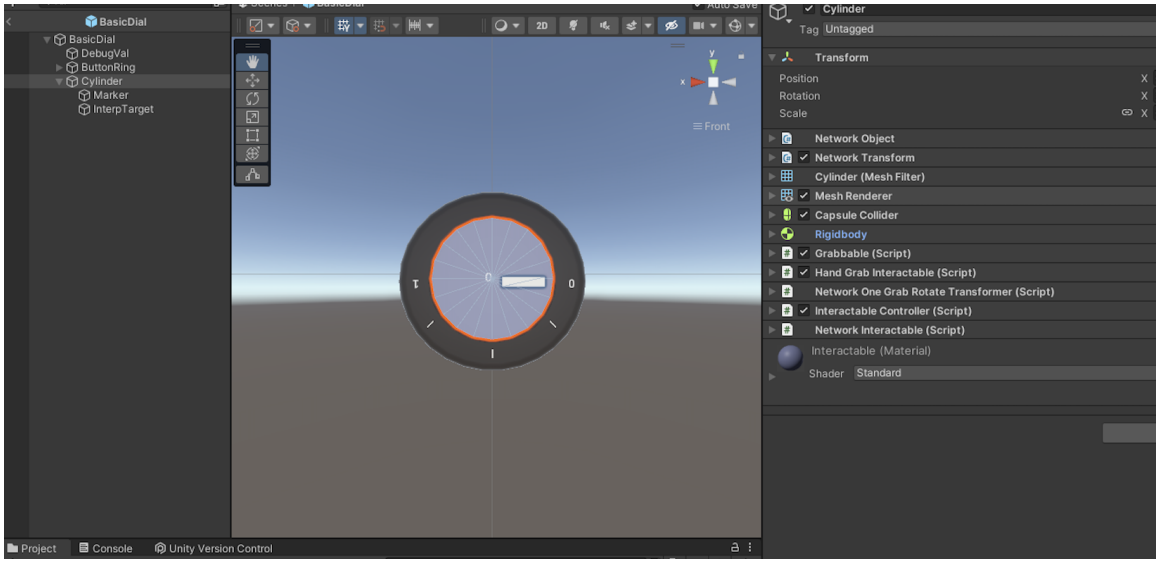


A grab requires the ability to translate, rotate, and scale objects through control points directly mapped to our hands. These transforms can be broken down into three stages. (1) An initialization which determines which points in the object are controlled by which attributes of the hand. (2) The actual manipulation which updates the object according to the hands translation and rotation or the scale of the object according to the distance between two control points. (3) The release which determines when the object stops being controlled and whether it maintains any motion.

This prefab is a simple object that can be grabbed and moved with one hand. It's paired with an object source and simulation so that someone can take one instance of it from its container and place it into the active part of the simulation. Using the rigidbody collision as a trigger, the object is then "activated" and can exhibit different behavior.

This basic interaction, as with all grabs, worked best for smaller objects since the only way to get physical feedback is through a full pinch. For larger objects, an open palm grab doesn't trigger as well and leaves you with an awkward hand position.

- Dial: This prefab was constructed using a grabble center cylinder with move-

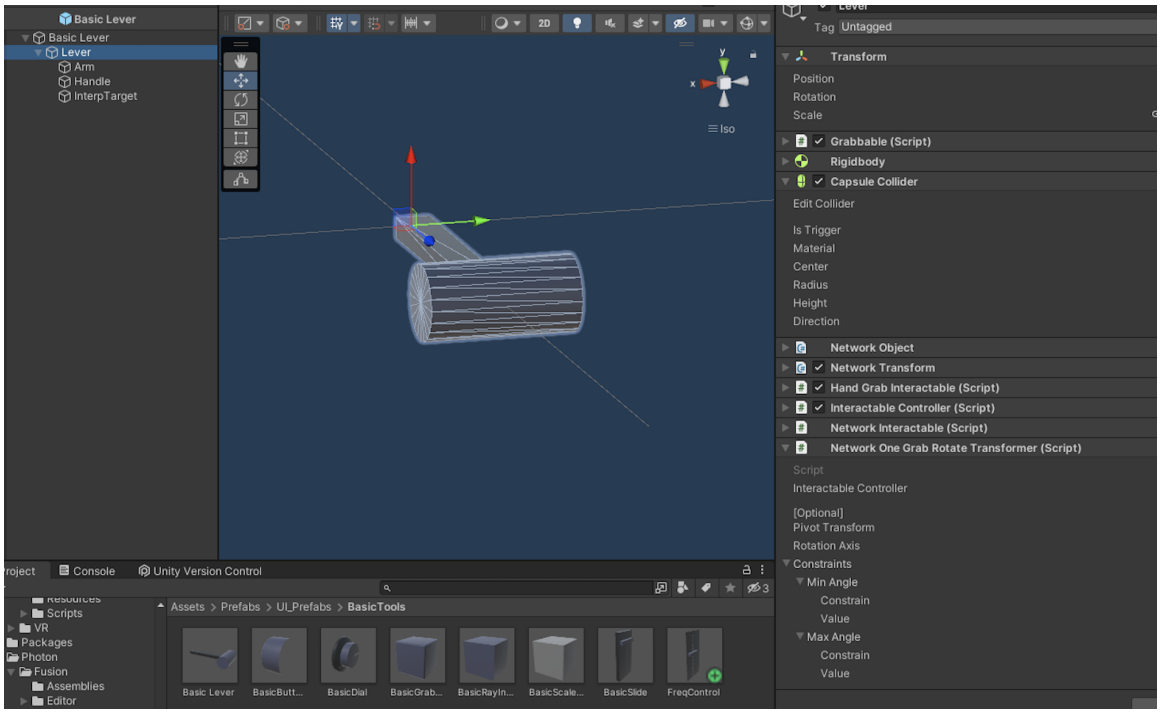


ment restricted to rotations around the y axis. The rotation was then parsed

to be able to be used as a control. Within the demo, for example it controls the rate of the flow of water.

This setup provided an interesting example of conflicting form. In reality the dial should rotate with the rotation of your fingers. However, within OVR it rotates relative to the vector between the dial center and your hand. In order to get a more accurate form, the movement of the dial would need to be tied to a specific finger. This isn't possible tho, since the fingers are typically included within this motion.

- Lever:



This prefab was constructed using a grabbable handle with movement restricted to rotations around the base of the lever. Unlike the dial, the output of this control was a discrete on/off.

Unlike the dial, the handle grab actually corresponded to the form factor. However, a lever is typically associated with a stronger level of resistance which the virtual component can't have. In addition, the motion of the handle highlighted the fact that the hand was not forced to track with the rotation creating a worse

separation between hand and tool than with the dial. These two factors resulted in a less satisfying if more intuitive interaction.

- Slide:

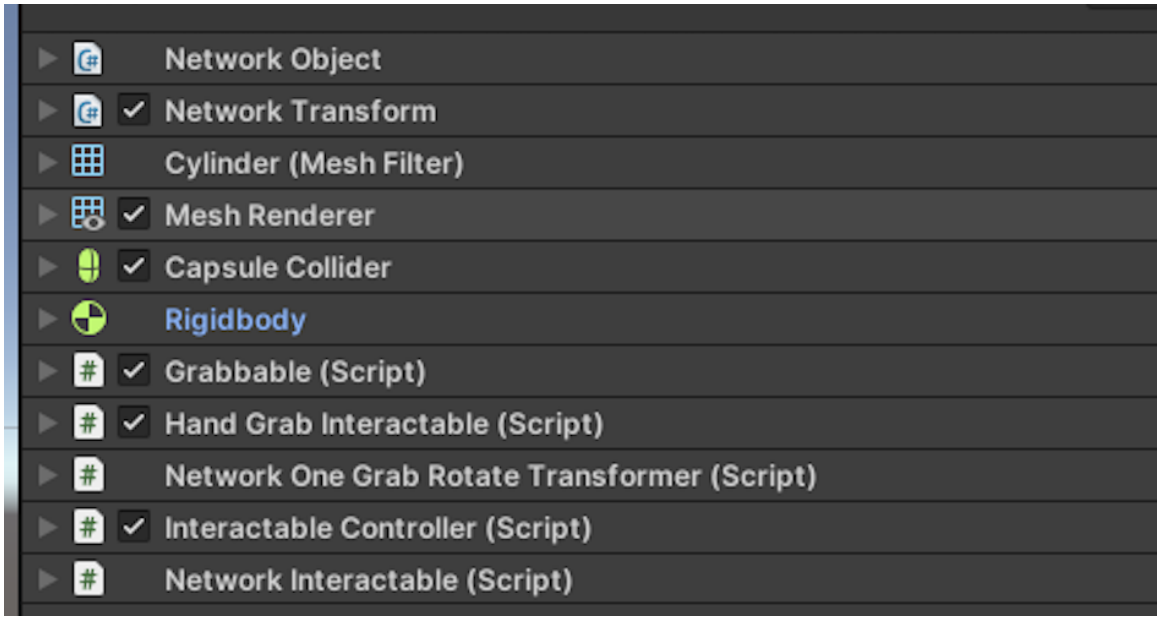
This prefab functioned similarly to the dial but with a constraint on translation rather than rotation.

Of the three basic controls, this interaction worked the best. The simple pinch required to move the slider resembled a realistic interaction, and the linear motion was easy to track.

Each of these components were then used to build up a simple scene where the controls modified a reservoir of water within a simple terrain. The button and lever reset the water level. The dial the rate at which the level decreased. The placeable would activate when added to the terrain of the scene. The slide controlled the number of objects that each placeable would then spawn. A simple text outputs the current level and flow rate of the scene.

6.2 Networked Interactions

In order to integrate these prefabs with the larger WIT project, a couple modifications needed to be made. Primarily, in order to make the experience multiplayer, photon fusion's hosted mode is used to maintain a ground truth central server. This prevents any individual headset user from making direct modifications to a networked game object. So in order to make networked objects interactable, local users interact with a local version and request to move the server object. The process described below specifically focuses on grabbable objects since this is the interaction that needs to dynamically update on the network side. However, a similar approach should work for networking a button's motion or other interactions with a shared visual component.



First, each object requires a series of new scripts. The interactable GameObject as a whole requires a Network Object and a Network Transform. The component of the game object that is moveable requires the addition of a Network Object, Network Transform, Interactable Controller, Network Interactable. The two sets of Network Objects and Transforms are needed because of how fusion handles parenting. Each Network Object is responsible for all objects parented under it until a new GameObject is reached. Additionally, a Network Object is required to directly modify that element.

In order to create smooth movement, an InterpolationTarget is also required by fusion. Within Unity, this is just an empty GameObject placed as the child of the GameObject that is being moved.

The actual movement then requires a modification of the OVR scripts that define how an object is modified on a grab. These changes include the addition of a couple network specific components as seen in the code snippet below.

```
private NetworkRunner runner;
public InteractableController interactableController;
private IGrabbable _grabbable;
private Pose _grabDeltaInLocalSpace;


public void Initialize(IGrabbable grabbable)
{
    _grabbable = grabbable;
    runner = GameObject.Find("/NetworkManager").GetComponent<NetworkRu
}
```

The other major change then happens in the "OnGrab" function. Within this section, any changes that are made to the objects transform have to be made on the interactableController instead.

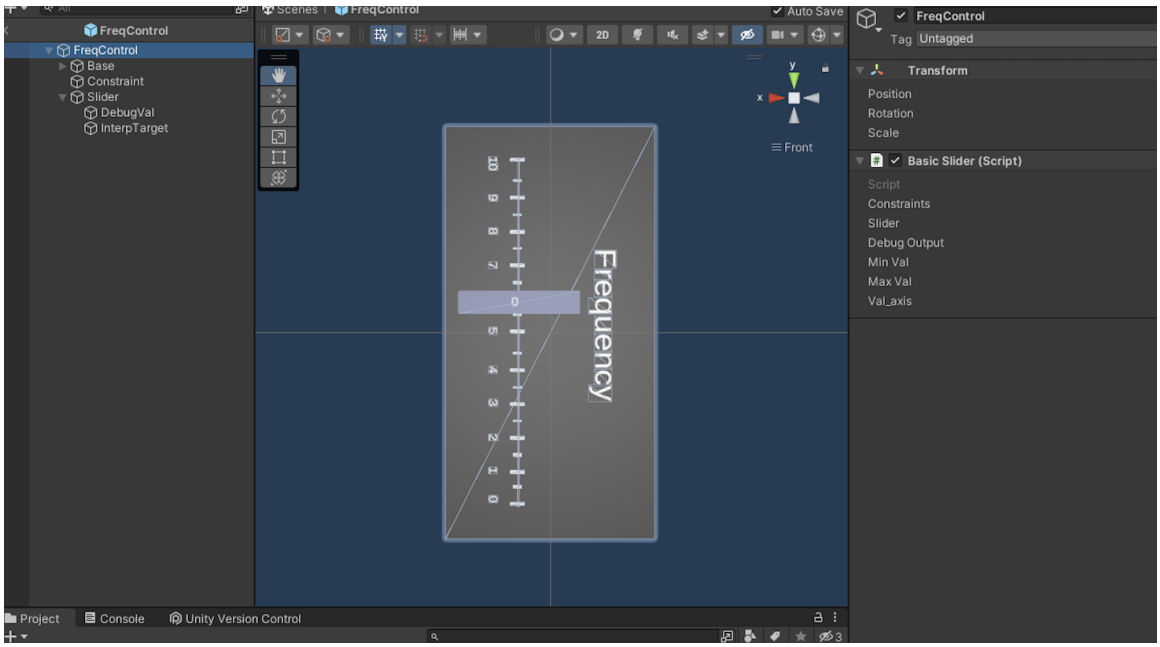
```
if (runner.IsConnectedToServer) {
    interactableController.position = # new position
    interactableController.rotation = # new rotation
} else {
    transform.position = # new position
    transform.rotation = # new rotation
}
```

The next component we began to adapt was the two hand grab. When using two control points you gain the ability to scale an object. This creates issues when using the default fusion setup since it does not automatically network scale like it does for

transforms. However, once the fusion side of the technical work is able to handle scale the changes would be the same as for the one hand transform.

6.3 Interaction Modifications

6.3.1 Translation Constraint



When adapting the OVR script that handles constrained translations there were differences in coordinate systems that were not handled. In OVR, the constraints are represented as simple values within the local or global space. This representation is slightly unintuitive when setting up within Unity since you have to figure out these values. These values then require additional calculations within the OVR code to handle the object's motion, scaling, etc. To circumvent these issues I fully adjusted the constrained translations script so that it used the constraints of another GameObject. In setup this allows you to visualize the object's potential motions in 3D. For interactions, this provides a simpler check for constraints utilizing the pre-existing Unity functions. The constraint object, while not visualized, does require a Render component in addition to a Mesh. This is because the Unity MeshFilter

handles its bounds locally, while the Render.bounds provides the global constraints. This modified script is then used in the new version of the Slider.

6.3.2 Scaling Uniformity

The default OVR script only allows scaling to be performed uniformly across the axes. The change in local scale is calculated using the same proportional change as the distance between the two grab points. This script also rotates and translates the object by aligning the center of the object with the center of the grab points.

To allow for more subtle adjustments. I created the framework for a new script to allow for two options. (1) Scale the object along each axis according to the distance between your hands along that axis. (2) Only scale the object along the axis with the greatest distance between the two hands.

These components were then used to build out the final WIT Demo discussed in the final chapter.

Chapter 7

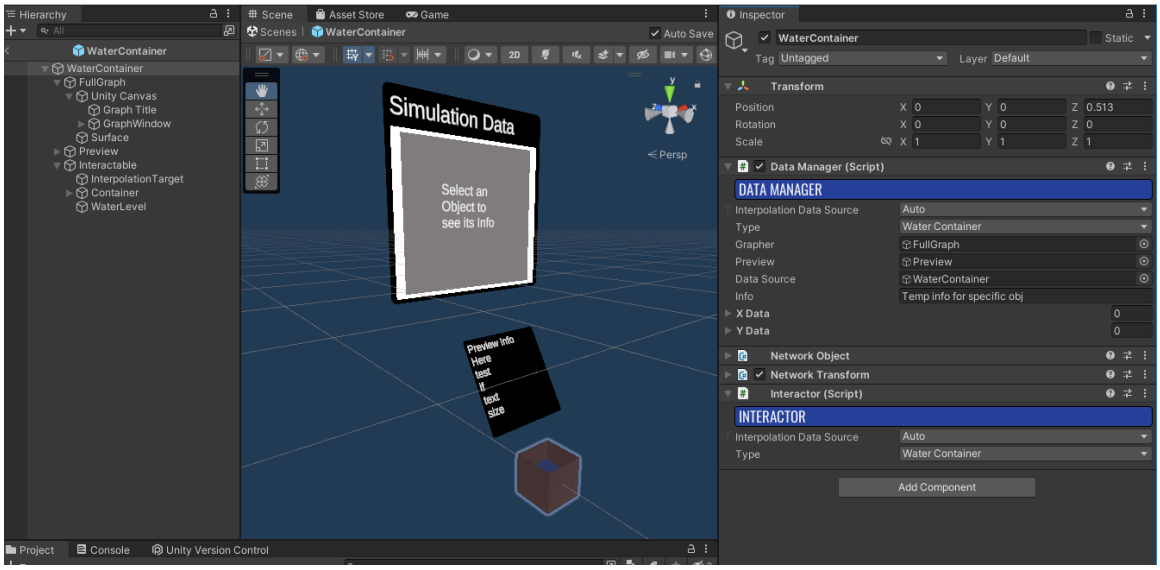
Results and Discussion

7.1 Final WIT Demo

For the final phase of this project, some of the interactions presented in the previous chapters were fine tuned to align more closely with a toy example. We chose to model a simple ecosystem with variable water and two separate agents within the system. Most of the modifications and adjustments to this system did not heavily impact the work presented in this paper. However, there were two game objects that were refined specifically for the final demo that are interesting to examine in more detail.

- WaterContainer: This GameObject is the visual representation of a well. On the system side it is represented as a Collector which has a quantitative amount of a resource - in this case water - that changes over time. The water container represents this data in three different ways.

  First, the water level within the container directly reflects the amount of water on the system side relative to the volume of the container surrounding it. An additional object is shown as a puddle of water around the container if the water volume is greater than 99% of the container volume. This provides additional feedback without breaking physics by overfilling the container. Ideally, interactions with this container could then change the amount of water the well is able to hold as well as the rate of evaporation by changing the volume and the



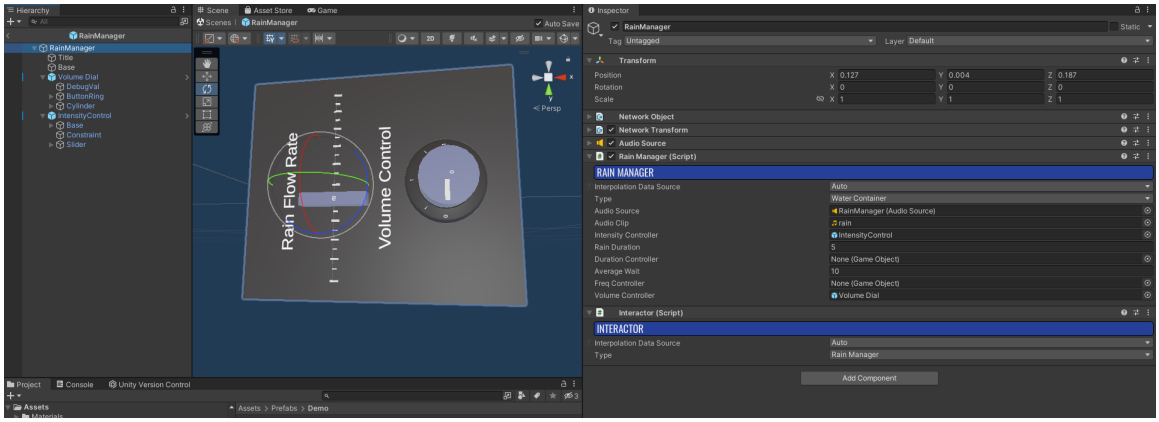
surface size. Due to the limitations of fusion this does not work while networked but can be seen with local interactions.

Second, the well shows a simple window above it with it's current water when the well is being pointed to. This allows a quick numerical inspection as well as a visual cue that there is more information tied to this object.

Finally, the well displays a graph of it's water level over time when the object is pinch selected on ray cast. This graph, could then be used to examine the behaviour of the system over time. In this case, the graph is a child of the WaterContainer, but the scripts are built to allow it to be connected in a more general way.

- RainManager:

  This GameObject is the auditory representation and physical modifier of the rain within the simulation. The system side handles whether or not it is raining as well and modifies the flow of water into the well according to the intensity of the rain. This flow rate is able to modified by using a slider. The rain manager then plays rain if it is raining and modifies the volume of the rain according to it's intensity. While developing this, I also found the need to add an additional volume control dial which caps the max loudness which corresponds to the max



rain. This dial could be used to modify the duration or likelihood of rain. Or switched out for a different control if there were more aspects to modify on the system side.

7.2 Discussion

Overall, this project was successful in building the ground work for future development of classroom scale AR activity.

From the theoretical exploration, we learned the importance of signaling the availability of interaction as well as the awareness of the user and the importance of physical and social components. These considerations were seen within the technical exploration, where maximizing audio and visual feedback made the virtual world more satisfying, engaging, and intuitive. Most surprisingly, we saw the importance of physical feedback and how the constraints of virtual objects impact our perception of actions.

Specifically, we found that smaller objects work better for one handed grabs since they result in the smallest discrepancy between object and hand shape. However, if dynamic object tracking is made available in the future this suggestion may change.

The biggest takeaway from the final demo development was the importance of having a targeted design. When developing future projects, stepping through the following questions is recommended.

- Decide what the goal of the activity is. Do you want to focus on the underlying

structures of a system and identifying the agents, flows, and other components? Do you want to focus on emergent behaviours and equilibrium?

- Decide what information and levers are important to access within the system. Do you care about information about specific agents or about the connection between generalized groups? Do you want to modify the capacity, quantity, flow, or other aspects within the system?

- Decide how the environment is best visualized within the room and where interactions will be localized relative to students. Are interactions concentrated within a small space? Are interactions spread out across large distances?

- Decide how information will be communicated within this space. Do students need to share graphs or other information in a non verbal manner?

This understanding, can then inform the decisions that go into the technical development. Similar, to any UI design, the goal of each control should be reflected in it's physical affordances. This work, shows promises steps towards this end goal and will hopefully become more and more accessible to casual developers as the available technology provides more and more support for these types of projects.

Appendix A

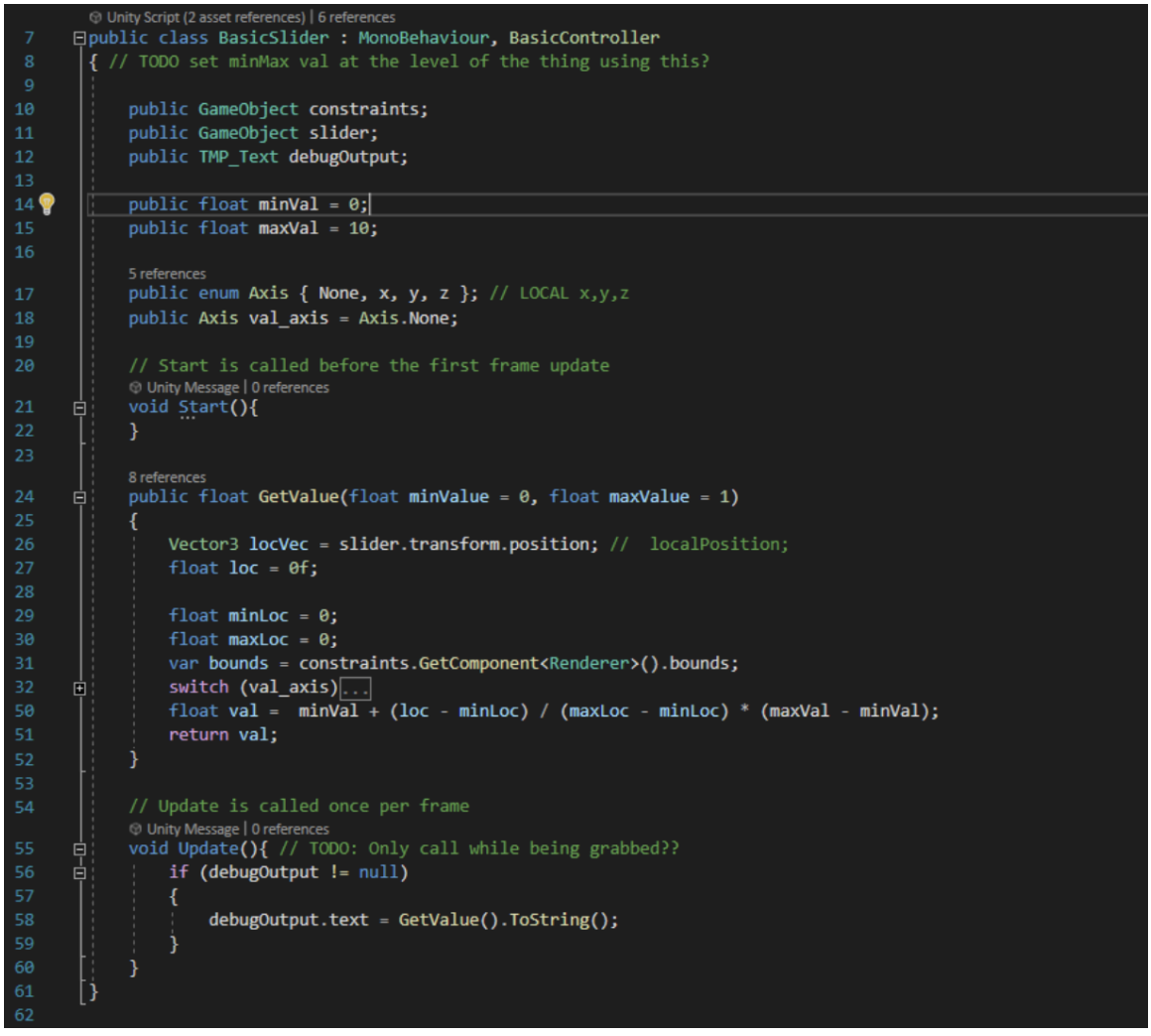
Headset Options

	Anchors	Pass-through View	Input	Other
Quest 2	Spatial Anchors	Black and White Stereo	Controllers Hand Tracking Voice	Weight: 503g Battery Life: 2-3 hr
Quest Pro	Spatial Anchors	Color Stereo	Controllers Hand Tracking Voice	Weight: 722g Battery Life: 1-2 hr Peripheral Vision
Pico 4	Player's Initial Location	Color	Controllers Hand Tracking	Android Support
Lynx		Color	Controllers	Peripheral Vision
HTC Vive: Cosmos Elite		None	Controllers	Uses Vive Discontinued
Lenovo: ThinkReality		Color Stereo		Android Support (Unreleased at time of project)
HoloLens 2	Spatial Anchors	Color Stereo	Controls Hand Tracking	Uses MRTK
Magic Leap 2		Color	Controls Hand Tracking	

Appendix B

Basic Controller Code

```
5   public class BasicDial : MonoBehaviour, BasicController {
6       private float dialValue = 0;
7
8       [SerializeField] private GameObject dial;
9       private float minRot = 0;
10      private float maxRot = 180;
11
12
13      // Start is called before the first frame update
            Unity Message | 0 references
14      void Start()
15      {
18          minRot = dial.GetComponentInChildren<Oculus.Interaction.NetworkOneGrabRotateTransformer>().Constraints.MinAngle.Value;
19          maxRot = dial.GetComponentInChildren<Oculus.Interaction.NetworkOneGrabRotateTransformer>().Constraints.MaxAngle.Value;
20      }
21
            2 references
22      public float GetValue(float minValue = 0, float maxValue = 1)
23      {
24          var rot = dial.GetComponentInChildren<Oculus.Interaction.NetworkOneGrabRotateTransformer>().GetAngle();
25          dialValue = minValue + (rot - minRot) / (maxRot - minRot) * (maxValue - minValue);
26          return dialValue;
27      }
28
29      // Update is called once per frame
            Unity Message | 0 references
30      void Update()
31      {
32          if (dial.transform.Find("DebugVal").GetComponent<TMP_Text>() != null)
33          {
34              dial.transform.Find("DebugVal").GetComponent<TMP_Text>().text = GetValue().ToString();
35          }
36
37      }
38
39  }
38
39  }
```



```
Unity Script (2 asset references) | 6 references
public class BasicSlider : MonoBehaviour, BasicController
{ // TODO set minMax val at the level of the thing using this?

    public GameObject constraints;
    public GameObject slider;
    public TMP_Text debugOutput;

    public float minVal = 0;
    public float maxVal = 10;

    5 references
    public enum Axis { None, x, y, z }; // LOCAL x,y,z
    public Axis val_axis = Axis.None;

    // Start is called before the first frame update
    Unity Message | 0 references
    void Start(){
    }

    8 references
    public float GetValue(float minValue = 0, float maxValue = 1)
    {
        Vector3 locVec = slider.transform.position; //  localPosition;
        float loc = 0f;

        float minLoc = 0;
        float maxLoc = 0;
        var bounds = constraints.GetComponent<Renderer>().bounds;
        switch (val_axis)...
        float val =  minVal + (loc - minLoc) / (maxLoc - minLoc) * (maxVal - minVal);
        return val;
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update(){ // TODO: Only call while being grabbed??
        if (debugOutput != null)
        {
            debugOutput.text = GetValue().ToString();
        }
    }
}
```

Appendix C

Data Graphing

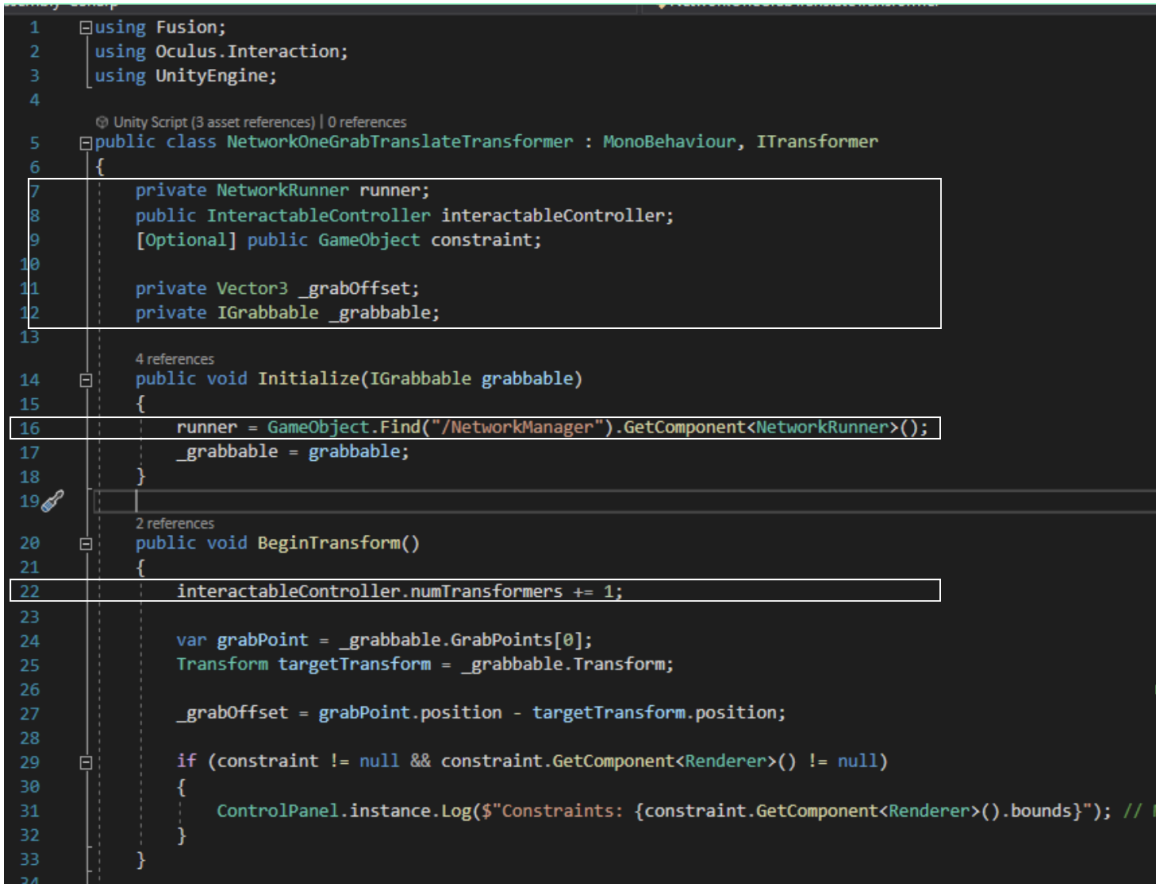
The graph visualization combines the Unity Line GameObject as well as simple Circle sprites to create the line and data point visualization. The coordinate of each point in the line and the dot are then determined by a linear mapping onto the local coordinate space of the graph window.

```
6   public class WindowGraph : MonoBehaviour{
7
8       [SerializeField] GameObject dataPointPrefab; // Replace with image - sprites allow for animation?
9       [SerializeField] GameObject graphBase; // TODO: Automate this connection
10    [SerializeField] Material lineMaterial;
11      // Start is called before the first frame update
12
13      // TODO: Make a data object: x/y axis values as well as min/max values - if non provided then set max and min to max&min rang diff times 1.5
14      // TODO: maybe this is even an array with multiple attributes you can choose to plot
15      float[] _yData;
16      float[] _xData;
17
18      Vector2 xRange; // min, max order
19      Vector2 yRange; // min, max order
20
21      List<GameObject> points;
22      GameObject line;
23
24      void Start() { // called when application / script initalized
25          points = new List<GameObject>();
26          CreateLine();
27      }
28      public void updateGraph(float[] xData, float[] yData)
29      {
30          _xData = xData;
31          _yData = yData;
32          int data_length = Mathf.Min(xData.Length, yData.Length);
33          xRange = new Vector2(0f, Mathf.Max(xData));
34          yRange = new Vector2(Mathf.Min(yData), Mathf.Max(yData) * 1.1f);
35
36          for (int i = points.Count-1; i >= data_length; i--)
37          {
38              points[i].transform.localPosition = GetLocalCoord(xData[0], yData[0]);
39              //Destroy(points[i]);
40          }
41
42          //points.RemoveRange(data_length, points.Count - data_length);
43          LineRenderer l = line.GetComponent<LineRenderer>();
44          l.positionCount = data_length;
45          for (int i = 0; i < data_length; i++)
46          {
47              if (i < points.Count)
48              {
49                  points[i].transform.localPosition = GetLocalCoord(xData[i], yData[i]);
50              }
51              else
52              {
53                  CreatePoint(xData[i], yData[i]);
54              }
55              l.SetPosition(i, GetLocalCoord(xData[i], yData[i]));
56          }
57      }
58
```

```
58
59      // Takes in data value not data position!
60      void CreatePoint(float x, float y){
61          Vector3 pos = GetLocalCoord(x, y);
62          GameObject newPoint = Instantiate(dataPointPrefab);
63          points.Add(newPoint);
64          newPoint.transform.localPosition = pos;
65          newPoint.transform.SetParent(graphBase.transform, false);
66          newPoint.transform.localScale = new Vector3(10, 10, 10);
67      }
68
69
70
71      void CreateLine() {
72          line = new GameObject("Line");
73          line.transform.SetParent(graphBase.transform, false);
74          LineRenderer l = line.AddComponent<LineRenderer>();
75          l.useWorldSpace = false;
76          l.startWidth = 0.01f;
77          l.endWidth = 0.01f;
78          l.positionCount = 0;
79          l.material = lineMaterial;
80          line.transform.localPosition = Vector3.zero;
81      }
82
83      Vector3 GetLocalCoord(float xVal, float yVal){
84          // Assumes x and y range start at 0
85      float xMult = graphBase.GetComponent<RectTransform>().sizeDelta.x/(xRange[1] - xRange[0]);
86      float yMult = graphBase.GetComponent<RectTransform>().sizeDelta.y/(yRange[1] - yRange[0]);
87      return new Vector3((xVal-xRange[0])*xMult, (yVal-yRange[0])*yMult, 0.0f);
88    }
89
90
91      // Update is called once per frame
92      void Update()
93      {
94
95      }
96  }
97
```

Appendix D

Non Uniform Scaling on Two Hand Grab

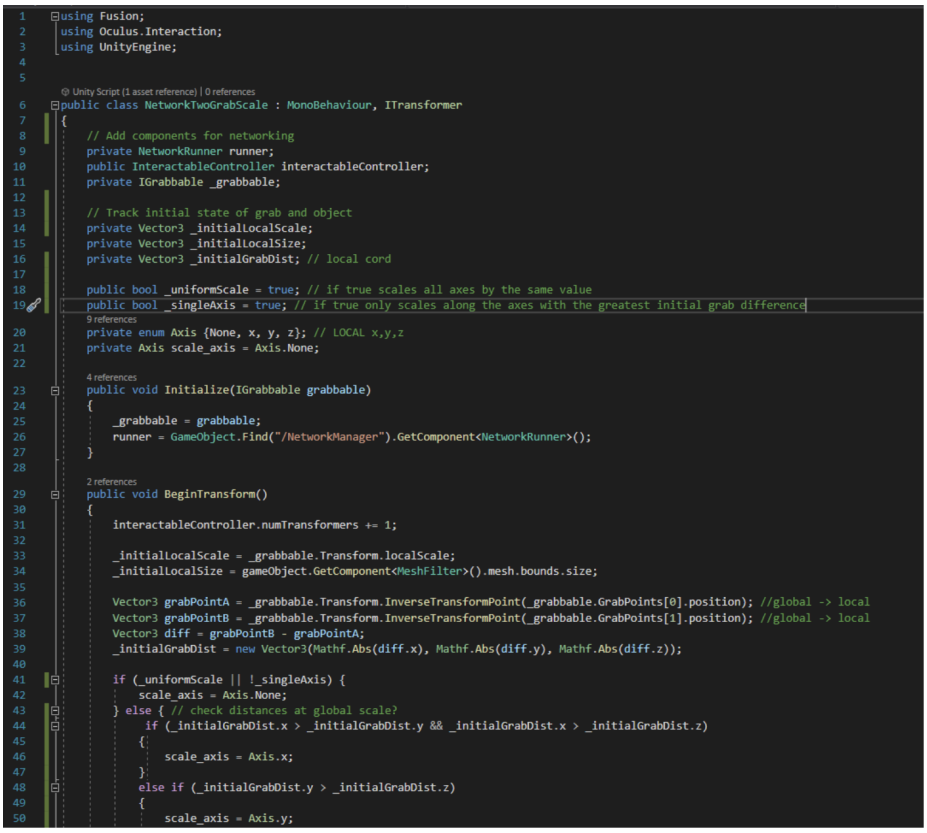


```
using Fusion;
using Oculus.Interaction;
using UnityEngine;

Unity Script (3 asset references) | 0 references
public class NetworkOneGrabTranslateTransformer : MonoBehaviour, ITransformer
{
    private NetworkRunner runner;
    public InteractableController interactableController;
    [Optional] public GameObject constraint;

    private Vector3 _grabOffset;
    private IGrabbable _grabbable;

    4 references
    public void Initialize(IGrabbable grabbable)
    {
        runner = GameObject.Find("/NetworkManager").GetComponent<NetworkRunner>();
        _grabbable = grabbable;
    }

    2 references
    public void BeginTransform()
    {
        interactableController.numTransformers += 1;

        var grabPoint = _grabbable.GrabPoints[0];
        Transform targetTransform = _grabbable.Transform;

        _grabOffset = grabPoint.position - targetTransform.position;

        if (constraint != null && constraint.GetComponent<Renderer>() != null)
        {
            ControlPanel.instance.Log($"Constraints: {constraint.GetComponent<Renderer>().bounds}"); //
        }
    }
```

```
19
                2 references
20              public void BeginTransform()
21              {
22                  interactableController.numTransformers += 1;
23
24                  var grabPoint = _grabbable.GrabPoints[0];
25                  Transform targetTransform = _grabbable.Transform;
26
27                  _grabOffset = grabPoint.position - targetTransform.position;
28
29                  if (constraint != null && constraint.GetComponent<Renderer>() != null)
30                  {
31                      ControlPanel.instance.Log($"Constraints: {constraint.GetComponent<Renderer>().bounds}"); //
32                  }
33              }
34
                2 references
35              public void UpdateTransform()
36              {
37                  Vector3 grabPoint = _grabbable.GrabPoints[0].position;
38                  var targetTransform = _grabbable.Transform;
39                  Vector3 newPos = grabPoint - _grabOffset;
40
41                  if (constraint != null && constraint.GetComponent<Renderer>() != null)
42                  {
43                      Bounds bounds = constraint.GetComponent<Renderer>().bounds;
44                      if (!bounds.Contains(newPos))
45                      {
46                          newPos = bounds.ClosestPoint(newPos);
47                      }
48                  }
49
50
51                  if (runner.IsConnectedToServer)
52                  {
53                      interactableController.position = newPos;
54                  }
55                  else
56                  {
57                      targetTransform.position = newPos;
58                  }
59
60              }
61
                2 references
62              public void EndTransform()
63              {
64                  interactableController.numTransformers -= 1;
65              }
66          }
67
```

Appendix E

Translation Constraint



```
using Fusion;
using Oculus.Interaction;
using UnityEngine;


Unity Script (1 asset reference) | 0 references
public class NetworkTwoGrabScale : MonoBehaviour, ITransformer
{
    // Add components for networking
    private NetworkRunner runner;
    public InteractableController interactableController;
    private IGrabbable _grabbable;

    // Track initial state of grab and object
    private Vector3 _initialLocalScale;
    private Vector3 _initialLocalSize;
    private Vector3 _initialGrabDist; // local cord

    public bool _uniformScale = true; // if true scales all axes by the same value
    public bool _singleAxis = true; // if true only scales along the axes with the greatest initial grab difference
    9 references
    private enum Axis {None, x, y, z}; // LOCAL x,y,z
    private Axis scale_axis = Axis.None;

    4 references
    public void Initialize(IGrabbable grabbable)
    {
        _grabbable = grabbable;
        runner = GameObject.Find("/NetworkManager").GetComponent<NetworkRunner>();
    }

    2 references
    public void BeginTransform()
    {
        interactableController.numTransformers += 1;

        _initialLocalScale = _grabbable.Transform.localScale;
        _initialLocalSize = gameObject.GetComponent<MeshFilter>().mesh.bounds.size;

        Vector3 grabPointA = _grabbable.Transform.InverseTransformPoint(_grabbable.GrabPoints[0].position); //global -> local
        Vector3 grabPointB = _grabbable.Transform.InverseTransformPoint(_grabbable.GrabPoints[1].position); //global -> local
        Vector3 diff = grabPointB - grabPointA;
        _initialGrabDist = new Vector3(Mathf.Abs(diff.x), Mathf.Abs(diff.y), Mathf.Abs(diff.z));

        if (_uniformScale || !_singleAxis) {
            scale_axis = Axis.None;
        } else { // check distances at global scale?
            if (_initialGrabDist.x > _initialGrabDist.y && _initialGrabDist.x > _initialGrabDist.z)
            {
                scale_axis = Axis.x;
            }
            else if (_initialGrabDist.y > _initialGrabDist.z)
            {
                scale_axis = Axis.y;
```

```
2 references
public void UpdateTransform() {
    float minVal = 0.0001f;

    var targetTransform = _grabbable.Transform;
    Vector3 sizeChange = new Vector3(1f, 1f, 1f);

    // Get the new local grab points
    Vector3 grabPointA = _grabbable.Transform.InverseTransformPoint(_grabbable.GrabPoints[0].position); //global -> local
    Vector3 grabPointB = _grabbable.Transform.InverseTransformPoint(_grabbable.GrabPoints[1].position); // global -> local

    // Get the absolute distanec along each local axis
    Vector3 grabDiff = grabPointB - grabPointA;
    grabDiff = new Vector3(Mathf.Abs(grabDiff.x), Mathf.Abs(grabDiff.y), Mathf.Abs(grabDiff.z));

    // Calculate new size by adding the change in hand distance
    Vector3 newTargetSize = _initialLocalSize + (grabDiff - _initialGrabDist); // TODO just store this
    newTargetSize = new Vector3(Mathf.Max(minVal, newTargetSize.x), Mathf.Max(minVal, newTargetSize.y), Mathf.Max(minVal, newTargetSize.z));

    // Calculate scale that corresponds to new size
    float x_scale = _initialLocalScale.x * newTargetSize.x / _initialLocalSize.x;
    float y_scale = _initialLocalScale.y * newTargetSize.y / _initialLocalSize.y;
    float z_scale = _initialLocalScale.z * newTargetSize.z / _initialLocalSize.z;
    Vector3 newScale = new Vector3(1, 1, 1);

    // Select which set of scales to actually use for new scale
    if (_uniformScale){ // use average scale across axis
        newScale *= (x_scale+y_scale+z_scale)/3.0f; // Average across scales
    } else if (_singleAxis){ // scale all axis seperatly
        switch(scale_axis)...
    } else {// scale axis with greatest difference on grab
        newScale = new Vector3(x_scale, y_scale, z_scale);
    }

    // Update the scale directly or through network link
    if (runner.IsConnectedToServer) {
        ControlPanel.instance.Log($"Need to add scale componenet to network");
        //interactableController
    } else {
        targetTransform.localScale = newScale;

    }

}

2 references
public void EndTransform()
{
    interactableController.numTransformers -= 1;
}
}
```

Bibliography

Chiang Chiang, T. H. C., Yang, S. J. H., Hwang, G.-J. (2014). Students' online interactive patterns in augmented reality-based inquiry activities. Computers Education, 78, 97–108. https://doi.org/10.1016/j.compedu.2014.05.006

[0] [1] Diaz, M., Alencastre-Miranda, M., Munoz-Gomez, L., Rudomin, I. (2006). Multi-User Networked Interactive Augmented Reality Card Game. 2006 International Conference on Cyberworlds. https://doi.org/10.1109/cw.2006.29

[2] E. Bozgeyikli and L. L. Bozgeyikli, "Evaluating Object Manipulation Interaction Techniques in Mixed Reality: Tangible User Interfaces and Gesture," 2021 IEEE Virtual Reality and 3D User Interfaces (VR), Lisboa, Portugal, 2021, pp. 778-787, doi: 10.1109/VR50410.2021.00105.

[3] Fugate, Jennifer M., et al. "The Role of Embodied Cognition for Transforming Learning." International Journal of School Educational Psychology, vol. 7, no. 4, 2018, pp. 274–288., https://doi.org/10.1080/21683603.2018.1443856.

[4] "'Getting Nerdy' Cellular Biology Augmented Reality Card Deck Demo." YouTube, YouTube, 20 Aug. 2019, https://www.youtube.com/watch?v=OcftVepIoqc. Accessed 30 Apr. 2023.

[5] Görlich, D., Akincir, T., Meixner, G. (2022). (rep.). An Overview of User Interface and Interaction Design Patterns for VR, AR, and MR Applications.

[6] Hostetter, Carol. "Community Matters: Social Presence and Learning Outcomes." Journal of the Scholarship of Teaching and Learning, vol. 13, no. 1, Feb. 2013, pp. 77–86., https://files.eric.ed.gov/fulltext/EJ1011685.pdf.

[7] Huang, T.-C. (2017). Seeing creativity in an augmented experiential learning environment. Universal Access in the Information Society, 18(2), 301–313. https://doi.org/10.1007/s10209-017-0592-2

[8] Jeffri, N. F., Rambli, D. R. (2020). Guidelines for the interface design of AR systems for Manual Assembly. Proceedings of the 2020 4th International Conference on Virtual and Augmented Reality Simulations. https://doi.org/10.1145/3385378.3385389

[9] Johnson-Glenberg, M.C., Megowan-Romanowicz, C. Embodied science and mixed reality: How gesture and motion capture affect physics education. Cogn. Research 2, 24 (2017). https://doi.org/10.1186/s41235-017-0060-9

[10] Johnson-Glenberg, Mina C., et al. "Collaborative Embodied Learning in Mixed Reality Motion-Capture Environments: Two Science Studies." Journal of Educational Psychology, vol. 106, no. 1, 2014, pp. 86–104., https://doi.org/10.1037/a0034008.

[11] Johnson, D.W., Johnson, R.T., Holubec, E.J. (1984). Cooperation in the Classroom. Edina, Minnesota; USA. Interaction Book Co. publishing

[12] Kang, S., Tversky, B. From hands to minds: Gestures promote understanding. Cogn. Research 1, 4 (2016). https://doi.org/10.1186/s41235-016-0004-9 Learn science, master stem, be future ready.: AR/VR Learning Creation. Merge. (n.d.). Retrieved December 11, 2022, from https://mergeedu.com/

[13] Laal, Marjan. "Positive Interdependence in Collaborative Learning." Procedia - Social and Behavioral Sciences, vol. 93, 2013, pp. 1433–1437., https://doi.org/10.1016/j.sbspro.2013.10.058.

[14] Logan, Catherine, MEd, EdS. (2022, October 26). Education Guidelines. Personal.

[15] Nielsen, Jakob. "Enhancing the Explanatory Power of Usability Heuristics." Conference Companion on Human Factors in Computing Systems - CHI '94, 1994, https://doi.org/10.1145/259963.260333.

[16] Nokes-Malach, T.J., Richey, J.E. Gadgil, S. When Is It Better to Learn Together? Insights from Research on Collaborative Learning. Educ Psychol Rev 27, 645–656 (2015). https://doi.org/10.1007/s10648-015-9312-8

[17] Plass, J. L., O'Keefe, P. A., Homer, B. D., Case, J., Hayward, E. O., Stein, M., Perlin, K. (2013). The impact of individual, competitive, and collaborative mathematics game play on learning, performance, and motivation. Journal of Educational Psychology, 105(4), 1050–1066. https://doi.org/10.1037/a0032688

[18] Satriadi, K. A., Smiley, J., Ens, B., Cordeil, M., Czauderna, T., Lee, B., Yang, Y., Dwyer, T., Jenny, B. (2022). Tangible globes for Data Visualisation in augmented reality. CHI Conference on Human Factors in Computing Systems. https://doi.org/10.1145/3491102.3517715

[19] Schmitz, B., Klemke, R., Walhout, J., Specht, M. (2015). Attuning a mobile simulation game for school children using a design-based research approach. Computers Education, 81, 35–48. https://doi.org/10.1016/j.compedu.2014.09.001

[20] Yoon, Susan A., et al. "Teaching and Learning about Complex Systems in K–12 Science Education: A Review of Empirical Studies 1995–2015." Review of Educational Research, vol. 88, no. 2, 2017, pp. 285–325., https://doi.org/10.3102/0034654317746090.