# CPD Report: VR Development and XR Hand Gesture Recognition

## 1 INTRODUCTION

For my CPD, proposed earlier in the semester, I planned to improve my programming skills by learning VR development and implementing various emerging technologies like “hand tracking” using Unity’s OpenXR suite. My initial learning goals were:

1. To research and understand programming implementation methods
2. Design and execute effective testing cases.
3. Develop structured programming methodologies.
4. Implement debugging and maintenance strategies.
5. Create a functional VR demonstration that showcases acquired knowledge.

VR development and programming skills are vital to my personal development as a software engineer as they incorporate all aspects I wish to progress in my career later down the road. VR development skills are also important as a transferrable skill in design, implementation, and testing as using the Unity game engine encourages strict processes to ensure that all aspects of the program are functional and capable of compiling before any program can be run.

Additionally, VR is an emerging technology that I foresee as an untapped potential for software engineering as aspects of VR development aren’t widely taught in classes (being a niche subject) and learning more about this technology makes me excited about what’s to come. I also intended for this project to lay the groundwork for a potential project for accessible VR for Auslan (Australian Sign Language) users which I plan to continue to work on later down the road.

## 2 STRUCTURE OF OBJECTIVES

To improve my VR programming and design capabilities, my primary learning objective was to build upon my existing Unity knowledge and focus specifically on VR development—a topic not extensively covered in my university coursework, particularly only accessible through elective selection as a software engineer.

Before this CPD, I had completed two academic units focused on Unity that covered:

- C# scripting
- Asset management
- UI design
- Scene development

However, VR introduces unique challenges – such as 3D interaction fidelity, hardware compatibility, and user immersion – that go beyond what I had previously studied.

To address this gap, I enrolled in a structured VR development course designed to build foundational XR programming skills. This course is designed to implement a VR app using the XR toolkit in Unity, using the headset and controllers of the relevant device, with hand implementation being implemented later outside the learning environment. This course covered:

- Building interactive VR applications using Unity's XR Interaction Toolkit.
- Evaluating and refining VR experiences for usability.
- Choosing appropriate XR hardware for project goals.
- Optimising VR apps to meet framerate requirements.
- Programming custom VR interactions.
- Deploying VR apps to support head-mounted displays (HMDs).

This learning phase was instrumental in preparing me for my second major objective: implementing hand gesture recognition.

At the outset, my understanding of gesture tracking—particularly using Meta Quest 3—was minimal. I approached this section as a research-heavy, experimental stage. My initial focus was on static hand gestures, to learn how Unity interprets hand positions and how these can be mapped to interactive outcomes (e.g., displaying text for a wave gesture).

Once static gestures were understood, I began exploring dynamic gesture recognition. Given the emerging nature of this technology and limited out-of-the-box support in current XR packages, this phase required significant experimentation and adaptability. My goal was not necessarily full implementation, but rather to understand the potential and limitations of current hand-tracking tools.

## 3 WHAT WAS DONE

Throughout the course of this CPD, I followed my proposed project plan closely while setting smaller, achievable goals along the way to maintain progress. These goals were established as I developed the project, providing focus and helping me manage challenges as they arose.

### 3.1 WEEK 1: ENVIRONMENT SETUP, RESEARCH

The first step was to set up a stable development environment. I created a Git repository for the project, which allowed me to keep versions of my work saved at different stages. This meant that if something went wrong (such as a broken feature or file corruption), I could safely roll back to a previous working version without losing progress—something essential for complex development like VR.

Next, I began a structured VR development course to guide the early stages of my learning. [1] Before I could dive into development, I needed to connect my Meta Quest 3 headset to Unity (my development software). This setup was more difficult than expected, as the available instructions were outdated.

A requirement from Meta involved registering a developer account under an organization to activate "developer mode" on the headset—this led to the spontaneous creation of my fictional development studio: "NotTheFish".

Once developer mode was active, I linked the headset to my PC, adjusted Unity's build settings, and configured the project to be tested directly on the headset, wirelessly. This process was essential to test features as I built them in a real VR environment.

By the end of the week, I had completed the first section of the VR course and implemented the foundation of my VR room – a small, traditional room complete with a sofa, table, fireplace, and other decorative items. This space would later be used as the setting for the VR interactions further in the course.

## 3.2 Week 2–3: Tutorial completion, basic VR scene creation, introduction of user interaction

Continuing through the course, I focused on building user interaction features in the virtual room.

- Snap Turning: This allows the user to rotate their view left or right given by a set degree within the VR environment, making it easier to navigate without needing to physically turn. This is a common accessibility feature in VR to prevent discomfort.
- Teleportation and Anchors: I added areas in the room where the user could "teleport" by pointing to a controller. For example, users could aim at a rug in the room and instantly move their viewpoint there. I also implemented "anchors" which control exactly where and how a player lands when they teleport, including their direction of view.
- Grabbable Objects: I introduced basic interaction with objects by making items like a tennis racket and ball "grabbable" using the VR controllers. To ensure these items behaved correctly, I added invisible attachment points so they would align properly with the user's hand when picked up.
- Sockets: These are points where objects snap into place when released—for example, a hat is placed perfectly on a hook or the user's head. This ensures the virtual environment remains tidy and realistic rather than having objects float or fall awkwardly.

At this point, I experienced an illness that temporarily delayed my progress. Once recovered, I resumed the course and completed the final two units.

In Unit 2, I focused on events and responses in the environment. I created:

- A radio that plays music when turned on.
- A TV that displays video when activated using a remote-control object.
- Objects that give feedback—like vibrations—when interacted with.

These features created a deeper sense of realism. For example, the position of the sound of the radio changes based on the user's location, using Meta XR's spatial audio, which makes the music seem to come from one side of the head or quieter or louder depending on the location in the room.

I also explored user interface (UI) elements, such as menus the user can interact with using VR controllers. I even tried to create a wrist-mounted menu, which unfortunately didn't work as intended, but was a valuable experiment in pushing the limits of what I was learning.

In Unit 3, I tackled performance and comfort, which are crucial in VR:

- I added a fade effect during teleportation to reduce motion sickness.
- Gave users a choice between smooth and snap turning and used visual effects like motion blur to make turning feel more natural.
- Implemented distance grabbing, which lets users pull objects toward them without walking across the room.

I also learned to optimise the environment so it would run smoothly even on less powerful devices. This included reducing the number of visual details in some objects and using baked (pre-rendered) lighting for things like sunlight or lamp glow—saving the device from having to calculate lighting in real time.

By the end of this phase, I had created a fully functional and immersive VR environment. I received a course certificate as evidence of completion, and I was ready to move on to the next challenge: implementing hand gesture recognition.

## 3.3 Weeks 4–5: Hand Gesture Recognition

With the foundation built, I updated my project to work with the latest version of Unity XR, which included improved support for hand tracking. After resolving some compatibility issues, I successfully merged my existing work with the updated toolkit.

Initially, I was worried hand-tracking would be complicated to implement. However, the XR Toolkit had built-in support for hand models. Once configured, users could place their controllers down, and their real hands (captured by the headset's cameras) would appear in the virtual environment.

I then explored static gesture recognition—getting the program to recognise specific hand shapes. For example:

- A "thumbs up" or "OK" sign.
- An open palm.

Using Meta's API, I learned how to detect: [3]

- Hand shape (the relative position of fingers),
- Pose (how the hand is oriented in space),
- And set conditions (e.g., fingers curled a certain amount) that trigger events when matched.

These were tested in a built-in debug environment provided by Meta, which helped me collect and verify values for different gestures. Once a gesture was recognized, I could trigger visual feedback, such as a text message confirming the gesture was detected.

Encouraged by this progress, I attempted dynamic gesture recognition, which means detecting movement over time—like waving. I tracked wrist movement and tried to combine that with static poses to create a dual-condition gesture system. For instance, to recognise a wave, the program needs:

- An open hand (static shape),
- Moving side-to-side (dynamic motion).

While I had some limited success getting this to work—triggering a brief output of text when the gesture was performed correctly—it was inconsistent, and I didn't have enough time to fully debug it before the deadline. Still, it laid the groundwork for further development.

## 3.4 Week 6: Debugging, project polish, final documentation and media creation.

With the core project complete, I shifted focus to creating documentation (including this report) and preparing for the final presentation.

In a breakout session, I gave a practice presentation to receive early feedback. I was advised to reduce technical jargon and make the content accessible by focusing more on my personal development and learning journey. This helped shape my final delivery.

Unfortunately, some last-minute issues—like illness and difficulty getting my headset to connect—slowed progress. However, I completed and submitted the final presentation and supporting materials the following week.

# 4 Challenges and Changes

Despite my successes, I did run into some challenges and roadblocks throughout my project, this can be seen in the final product as well as some more underlying situations that followed me through the project, including difficulty with the pre-built hand-tracking with the older system used during the learning stage, the time-intensive debugging that was necessary for static/dynamic gesture recognition, and scope limitations.

## 4.1 Initial Setup and Headset Problems

During the initial setup of the VR learning course, the requirements for getting the headset to pair with my computer was the first hurdle that I had to overcome, it took quite a bit of time to research and uncover why the connection wouldn't be allowed before realising the "developer mode" was required. The process of setting the "developer mode" itself was difficult as well, as the documentation doesn't give the clearest instructions and seems to change from device to device.

Also, when trying to set up developer mode, which required a mobile device to complete, the headset would be listed as offline, then not connect to the mobile device, and then the headset needed affiliation to an organisation which I had to also set up. Overall, the experience showed me that not all answers are readily available, and it strengthened my patience for technology just not working as intended.

## 4.2 Compatibility with Learning System to Final Implementation

Another major challenge was the difference in compatibility between the XR system used in the Unity VR course and the newer XR toolkit I needed to use for the hand implementation. The learning modules depended on an older interaction system, which did not support modern hand-tracking. While the course gave me a solid framework, I had to rebuild and reconfigure many aspects of the environment—including teleportation systems, UI interaction, audio sources, and sockets—just to make them work with the updated hand-tracking system.

This process was time-consuming, as I had to manually test, remove, or rewrite features from the initial learning course to work with the newer version of Unity's XR Interaction Toolkit. In hindsight, a clearer understanding of the final build environment from the start might have allowed me to avoid some of this duplication of effort.

## 4.3 Gesture Recognition Complexity and Debugging Time

Another significant challenge was the technical complexity of gesture recognition, particularly when it came to implementing dynamic gestures. While the Meta XR package provided a foundation for static gestures, interpreting finger positions and pose constraints accurately required trial-and-error testing and considerable research.

The debugging process became especially time-intensive when I began experimenting with dynamic gesture recognition. Creating a system that could detect both a static gesture and movement over time (e.g., waving) introduced a host of new problems. Getting both conditions to reliably trigger an output often was unsuccessful, and the system worked inconsistently even when I believed the gestures were being performed correctly.

As a result, while I was able to produce a solid functional prototype of a static gesture system but an early and mostly dysfunctional version of a dynamic gesture (a wave), these failures I attribute to some theories:

- The API cannot handle registering the static gestures while movement is performed and thus the system cannot function with the combination of conditions
- The movement recognition was either too specific or too vague for the condition to be registered properly, requiring further refinement.
- Some aspect of the programming was overlooked or missed in the scope of time given to produce a successful program.

These dynamic gesture features weren't polished or reliable enough for broader use. This remains an area I would like to revisit in the future with more time and resources.

### 4.4 Scope and Time Constraints

Time management became a limiting factor as well. Midway through the project, I experienced an illness that slowed progress, particularly during project Weeks 3 and 6. This, combined with ongoing university assignments and the unexpected complexity of gesture recognition, forced me to scale back my initial scoping goals.

Originally, I hoped to begin early work on a full Auslan-to-text communication framework. However, given the challenges with even basic gesture recognition, I was forced to acknowledge that this was beyond the projected scope of what could be achieved in a single CPD cycle. Instead, I shifted focus to ensuring the gesture system I had developed was functional at a foundational level, with the intent to build upon it later.

## 5 What I'd Do Differently

There are some small things, upon reflection, which I would either choose to do differently or avoid altogether, mainly because of time constraints or skill gaps that needed to be addressed before the project. These lessons not only apply to this specific project but to my broader practice as a developer.

### 5.1 Narrow the Project Scope Earlier

Initially, my CPD plan included the plan to explore a full Auslan-to-text translation framework. While this was caused by a passionate interest in accessible VR solutions, it quickly became clear that the technical and developmental scope was too large for the timeframe and my current level of experience.

In future projects, I would aim to define clearer scope objectives from the beginning—defining a core "minimum viable product" (MVP) first and treating more advanced features (like dynamic gestures or full language systems) as stretch goals. This would help manage time better and reduce the risk of important milestones being pushed back due to time-consuming experimentation.

### 5.2 Allocate More Time to Debugging and Testing

I underestimated how long testing and debugging would take particularly for dynamic gesture recognition. While I made a solid finished product with static gesture detection, combining it with motion-based conditions introduced complexity that I wasn't fully prepared for.

If I had scheduled more dedicated time for testing and further debugging, I could have pushed the dynamic recognition further. Going forward with other projects, I'll try to factor in buffer weeks or fallback plans in project timelines to accommodate the unexpected but inevitable time spent troubleshooting.

### 5.3 Seek Peer or Mentor Feedback Earlier

Although I received useful feedback during my breakout presentation later in the semester, I now recognise the benefit of sharing progress earlier with peers, tutors, or online communities. External feedback could have helped steer me around common pitfalls, particularly with Unity's XR Hands API or gesture tracking quirks. Next time, I'll aim to involve others more regularly, even during the prototyping phase.

## 6 Future Steps and Continuing Development

This CPD project was always intended to be the foundation for a larger, longer-term investigation into VR interaction and its uses in accessibility. Although I made meaningful progress, particularly with static gesture recognition and user interaction design, there are several areas I plan to continue working on beyond the CPD project cycle.

### 6.1 Refining Gesture Recognition Accuracy

One of the future goals is to improve the reliability and accuracy of the gesture recognition system. At present, gestures can be triggered unintentionally during other movements or interactions, which impacts usability.

A simple but effective short-term solution may be to implement a toggle system, for example, using an "activation" gesture (like a raised closed fist) to turn gesture recognition on or off. This would prevent unintended gestures from being registered when the user isn't actively trying to communicate.

### 6.2 Exploring Machine Learning for Dynamic Gestures

Dynamic gestures (which involve motion over time) were not fully functional by the end of this project, and I recognise that the current approach based on position and movement tracking alone has various limitations.

In future considerations, I plan to explore machine learning (ML) approaches to gesture recognition. These methods would allow the program to "learn" gesture patterns based on user-recorded data and accurately identify movements with more nuance and adaptability than rigid, hard-coded rules.

A possible implementation could involve:

- Using TensorFlow or PyTorch to train a gesture recognition model.
- Collecting movement data within Unity.
- Labelling common gestures (like "wave," "point," or "beckon") during training.

- Exporting the trained model into Unity for real-time gesture analysis.

While this approach is significantly more complex and time-consuming, it holds great potential for creating more responsive, reliable gesture recognition systems.

### 6.3 Developing Better Testing Tools

During development, I made use of a debugging scene provided by Meta to test static gestures (hand shape and pose). I plan – at some stage – to create a similar custom debugging scene for dynamic gestures, enabling more precise testing of motion-based recognition. This would not only assist with current functionality but also provide a flexible tool for future development and experimentation.

### 6.4 Future Vision: Auslan Integration and Accessibility

Ultimately, the long-term vision of this project is to contribute to more accessible VR experiences, particularly for the deaf and hard-of-hearing community. One goal is to develop a system where Auslan gestures are translated in real-time into text boxes within the VR environment – allowing users who use sign language to communicate more easily in online spaces.

This concept bridges the gap between users who communicate via Auslan and those who don't, offering a new avenue for inclusive digital interaction. It's a direction I'm deeply passionate about and one that I intend to continue pursuing particularly as the focus of my future thesis work.

## 7 References

[1]
"Unity Learn," *Unity Learn*, 2022. https://learn.unity.com/course/create-with-vr?version=2022.3

[2]
*Youtube.com*, 2025. https://www.youtube.com/watch?v=mJ3fygb9Aw0

[3]
"XR Hands | XR Hands | 1.5.1," *Unity3d.com*, 2025. https://docs.unity3d.com/Packages/com.unity.xr.hands@1.5/manual/index.html

[4]

*Youtube.com*, 2025. https://www.youtube.com/watch?v=Lc1PuEatrCA

[5]

*Youtube.com*, 2025. https://www.youtube.com/watch?v=gg1eiCrY0e8