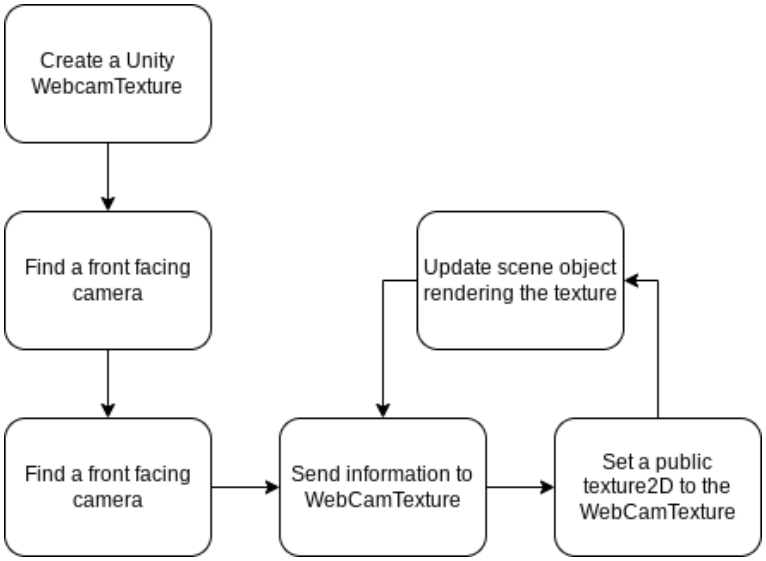
Improving Professional Training and Education Engagement with Accessible Augmented Reality

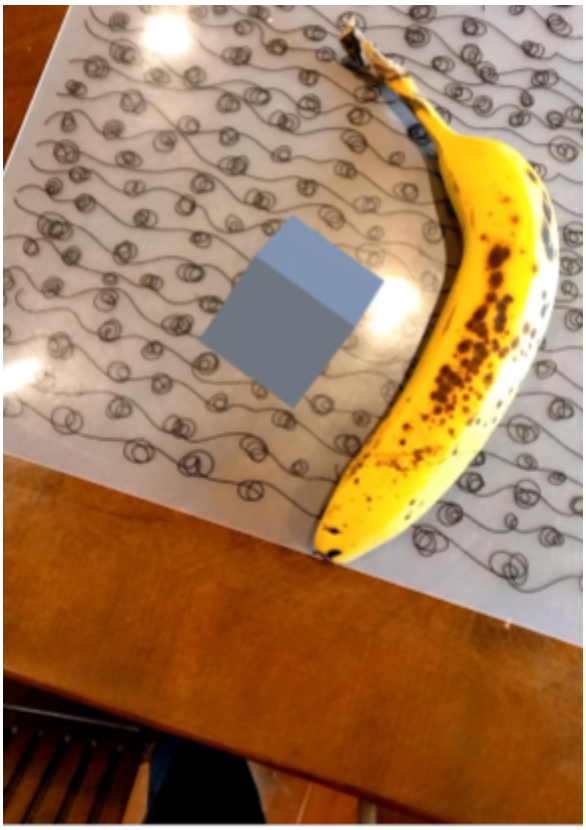
Project F16 by Aidan L and Jeremy F

[The following milestones are summaries of important entries in our engineering journal]

Milestone #2: (Started 11.12.21) Camera Implementation and Augmented Reality

We settled on using Unity to handle three dimensional interactions. We used a script to find a front facing camera. Once done so, a Unity **WebCamTexture** is updated every frame by that camera's input. This texture is then transferred onto a Unity scene object for the final camera to render as a UI overlay element.





AR demo, virtual cube is displayed over the camera input

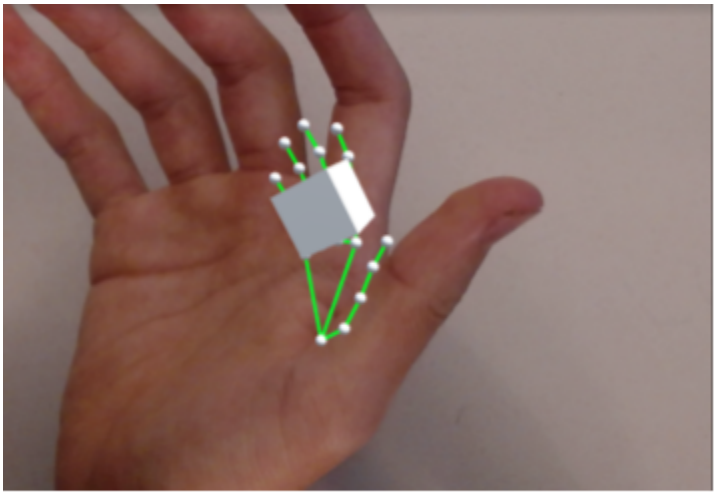
Once creating a live display, we moved on to the integration of virtual objects. Initially, we created a layer filter on the Unity scene camera so it could only see objects of that layer's tag. This filtered camera would then be rendered over the camera UI camera for the augmented reality effect. However this was more cumbersome and hardware intensive, so we pursued a simpler method. By switching the canvas to a screen space type relative to the camera, the UI overlay to be displayed behind the 3D scene instead of in front of it (when rendered as a screen space overlay type, the UI is rendered last and therefore behind objects in the scene).

Milestone #4: (Started 12.17.21) Integration Between Mediapipe and Unity

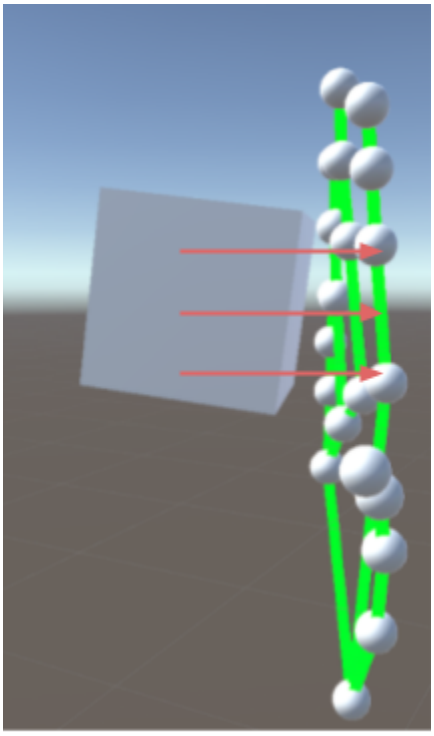
Mediapipe is a versatile and effective hand tracking algorithm that utilizes a combination of machine learning and advanced algorithms to track components of a hand. However, it only had direct support for languages like Python, Javascript, and C++. Although there were also standalone implementations for Android and iOS, they also meant that we wouldn't be capable of computing virtual interactions unless we built a 3D rendering system, physics system, and more, from scratch.

The first alternative was creating a Unity plugin using the .dll filetype in order to implement local computing calculations independent from Unity before sending it over. However, creating a .dll plugin involves a higher understanding of native structures and connections that we decided we wouldn't be able to learn in time for this project.

Instead, we used NatML, a free software and structure with support for Mediapipe to Unity connections. It allows any instance of our app to download the machine learning model from NatML servers. This model is then utilized locally, saving memory space and optimizing calculation times since the model is simulated on-device. As a result, we were able to create virtual interactions with our new hand model.



Although incorrectly scaled, this is an early screenshot of our hand tracking and interaction



Different perspective, this shows the hand-cube collision (Arrows indicate gravity)

Milestone #5: (Started 1.3.22) Implementation on mobile device

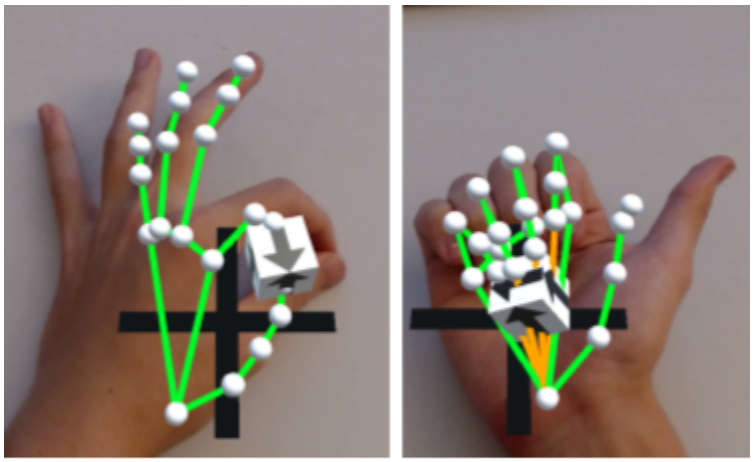
Testing until this point was performed on the Unity Editor. Once we built this for an android device, the app didn't track the hand and crashed nearly instantly. We implemented Android Logcat to debug the error and found a **System.Net.WebConnection** error. By increasing the API level from 24 to 27, we were able to track, but the app was still crashing. Again logcat showed us the error stemmed from **UnityEngine.Texture2D.ReadPixels** which originated in a file type

conversion method we created that left large artifacts every frame. Once fixed, the app ran correctly.

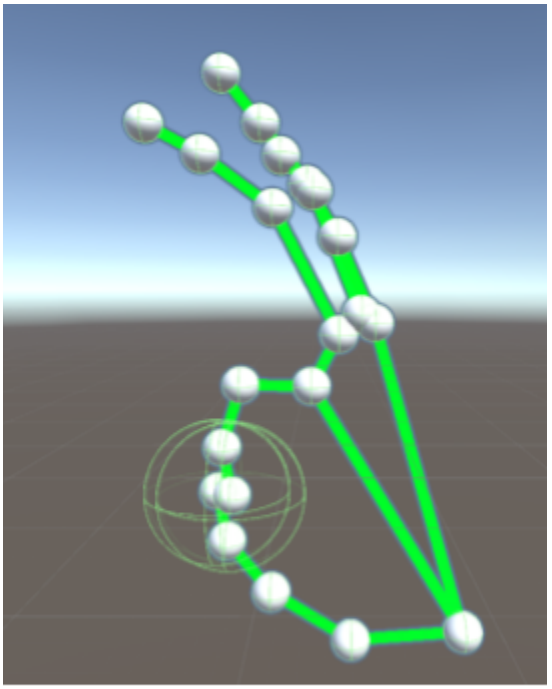
Milestone #6: (Started 1.15.22) Gesture Recognition

We originally created rigidbody colliders for the hand landmarks we wanted to track, however it was too hardware intensive and had multiple unnecessary dependencies, so we opted to manually calculate the distance between the fingertips we wanted to track. We tracked the Vector3 positions of each of these objects, then found their relative distance. This distance can then be measured to a threshold to determine when certain areas of the hand are within a 'pinching' or 'grabbing' threshold.

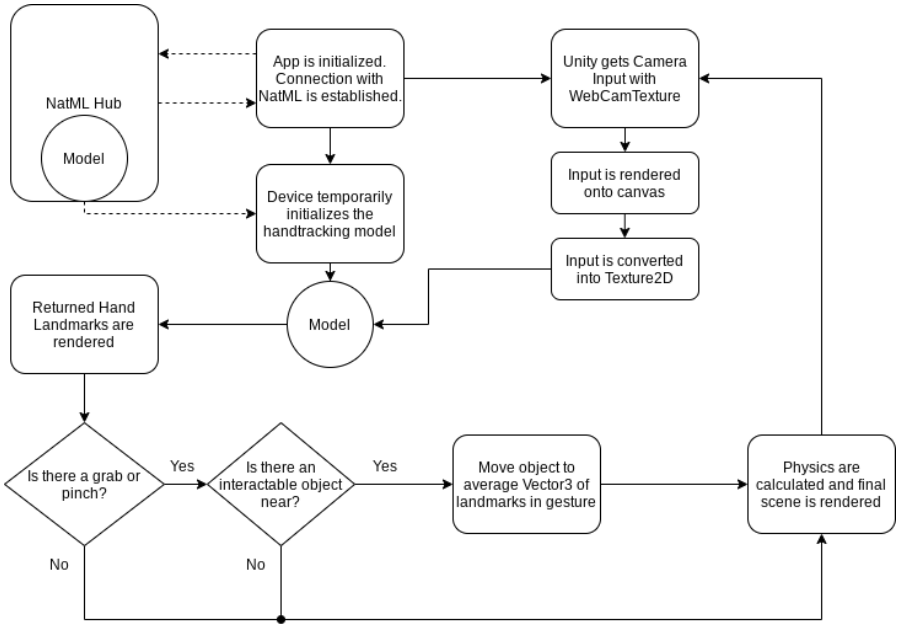
When this threshold is met, the nearest object tagged as **interactable** is 'grabbed' by the hand.



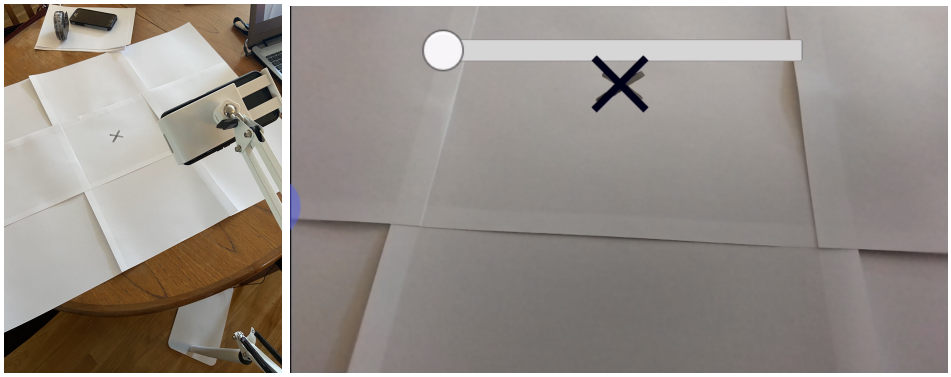
The cube's transform is calculated as an average between fingertips when it's picked up. The orange lines represent the fingertip's relative distance



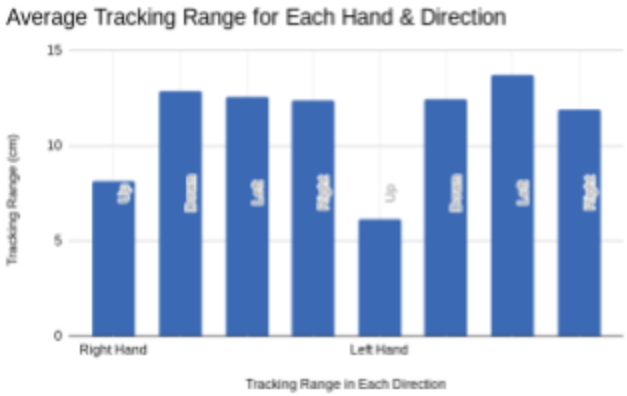
The larger green circles represent trigger colliders, whose collisions are detected by Unity's Physics system

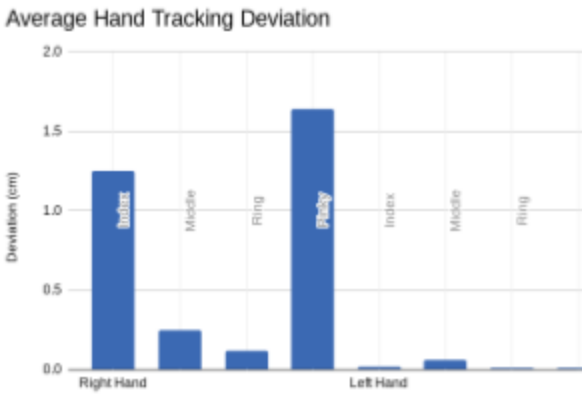


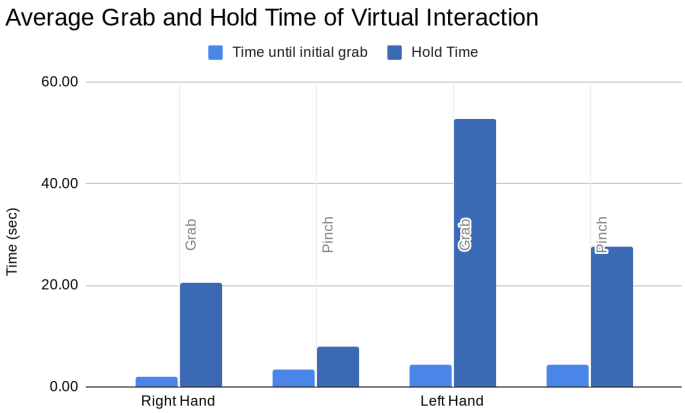
Milestone #7: (Started 1.20.22) Tracking Testing and Conclusion



This is our testing setup. The device is streaming to discord for us to view its screen. It's at a 60 degree angle of depression and is oriented 32.5 cm away from the real-life landmark. Using this correlation between a real-life and virtual landmark, we could test a variety of aspects of our design, including the range and accuracy of our hand tracking. We additionally tested the amount of time it took for our device to recognize gestures and interact with objects.







[Click for Testing Spreadsheet]

Conclusion:

Our testing showed that our app met many of the original criteria; tracking accuracy and interaction worked as expected, and tracking range, which was not part of t original criteria, proved to be the biggest issue in our design. Otherwise, it is practically functional, but can still be improved in certain aspects such as the maximum detection range and depth tracking.