Personal VR Hackathon

Author: Eric Silberstein

Claps: 99

Date: Feb 24, 2022

Over the long weekend I hacked together a VR demo. Iâ€™d never used a 3D gaming engine or developed for Android before. My goal was to figure out just enough to get my demo working. This blog post is about the in-the-weeds stuff I learned along the way that could help other VR newbies. But first some background on the demo.

I lead data science at Klaviyo. Vinicius Aurichio on our team was researching if, using only information up to and around the time of a first order, we could predict which of our customerâ€™s customers would place a second order. Our customers have a lot of data in Klaviyo. For each profile (one of their individual customers or potential customers) there is a timeline of events. These events are things like the person received an email, the person clicked an email, the person looked at a product details page, and the person placed an order.

You can review all of the events on the timeline of a single profile, but itâ€™s difficult to see the big picture. If you look only at aggregate information, though, itâ€™s easy to miss trends that get swallowed up in the aggregate statistics. It was a problem Vinicius faced while coming up with ideas for his machine learning models. To solve it, he built a visualization tool that sampled a few hundred profiles at a time, lined up their events according to elapsed time from some anchor event such as first order, and then plotted each event using color and shapes to signify event type.

As soon as I looked at Viniciusâ€™s visualization things popped out â€” not related to predicting if a profile would place a second order, but more basic things, such as profiles that should have but were not receiving regular campaigns. The visualization was like an X-ray for a Klaviyo account.

A doctor takes one look at an X-ray and can tell if you have broken bones. With Viniciusâ€™s plot, you could take one look and spot weird things â€” such as a flow not triggering in the way you thought, one message blocking another in a way you didnâ€™t intend due to smart sending, realizing that youâ€™re messaging people too often, or noticing that youâ€™re messaging people long after theyâ€™ve stopped engaging.

I donâ€™t know how many Quest 2s sold over the holidays but it must have been a big number, probably millions. (The companion iPhone and Android apps hit first place in the app stores.) I bought one and so did some of my colleagues. In January I used my Quest 2 for a bunch of 1:1s. The experience was a nice change from Zoom. Sure, you couldnâ€™t see the other personâ€™s actual face, but that was made up for by the feeling of being in a room with the other person, sitting at a virtual table, looking into the eyes of their avatar, and seeing their mouth move in sync with their voice. At one point three of us were in a meeting and I felt uncomfortably close to the person next to me, even though he was in London and I was in Boston!

I started wondering if our Klaviyo account X-ray visualization could be interesting in VR. Even a large monitor couldnâ€™t really do justice to Viniciusâ€™s visualization. In VR, the visualization could take up an entire virtual wall so you could â€œseeâ€ much more at a time. A third dimension could help â€” rather than using color and shape to represent event types, events could be spaced out over one axis by type. That would make it easier to see the marketing funnel in action, and allow simultaneous visualization of events spaced out over years (the long-term

customer relationship) and minutes (opening, clicking, browsing in a single session). â€œFlyingâ€ through your data also held potential. You could fly far away to get the big picture or fly close in along the timeline of a few profiles. VR could be good for collaboration. You and another avatar could both be in the data discussing why a profile was doing certain things, clicking in to see what products they looked at, and checking that the timing, copy and design of messages are all as intended.

Or maybe that all sounded cool but in practice would be useless. Thatâ€™s why I wanted to create a demo. Most of the time in this blog we talk about making things scale, and in our data science podcast we talk about being careful and drawing valid conclusions. I wasnâ€™t concerned with any of that. I wasnâ€™t trying to learn best practices, write idiomatic code, or make sure I understood edge cases. I just wanted to get something working by the end of the weekend.

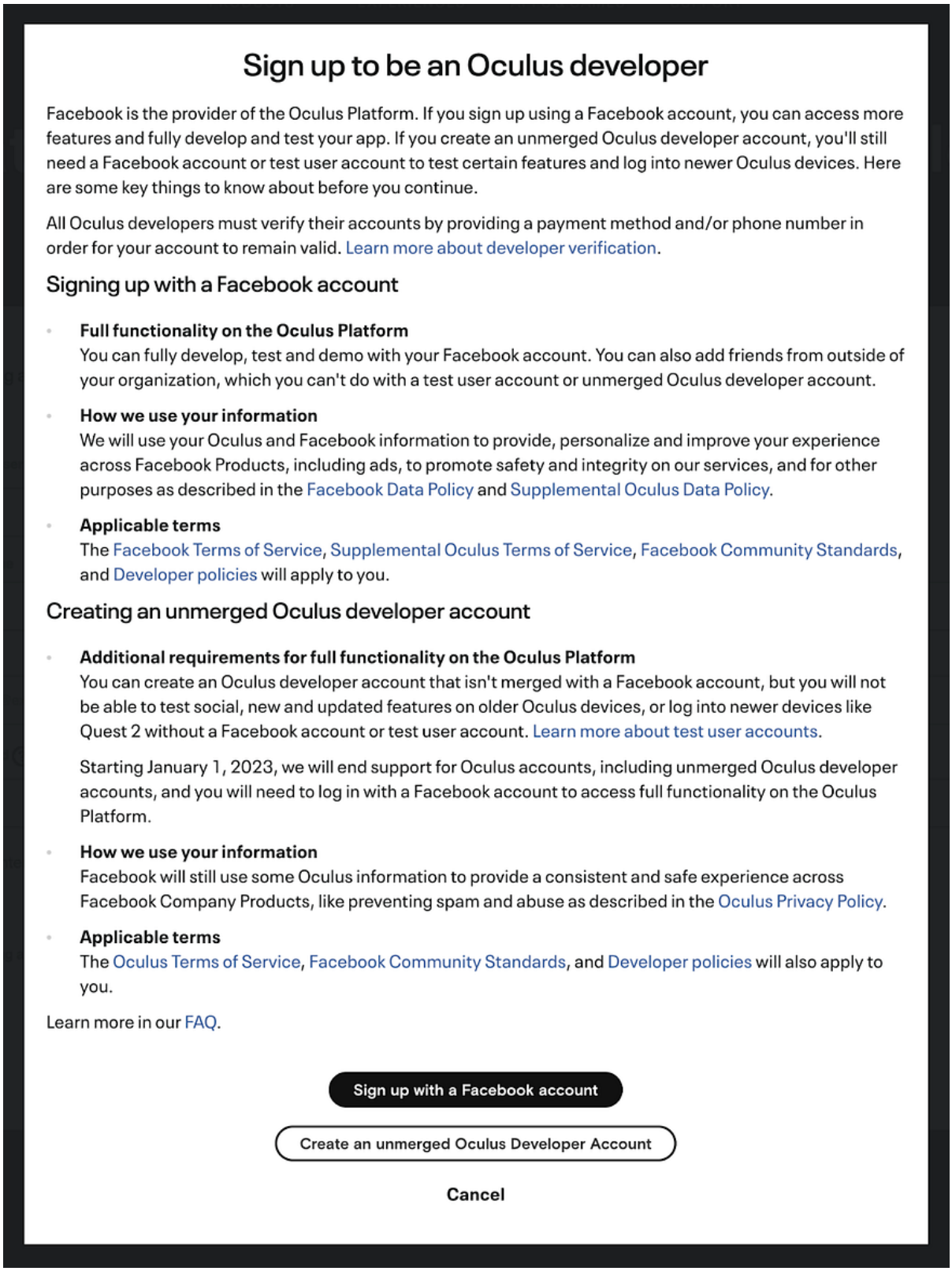
Unity, Unreal, Native, Web, Godot?

Unity was listed first on the Oculus for developers choose platform page. Showing points representing data in three dimensions didnâ€™t seem that different from showing objects in a game so a game engine felt appropriate. I installed Unity and didnâ€™t try any of the other options.

Turning on Developer Mode

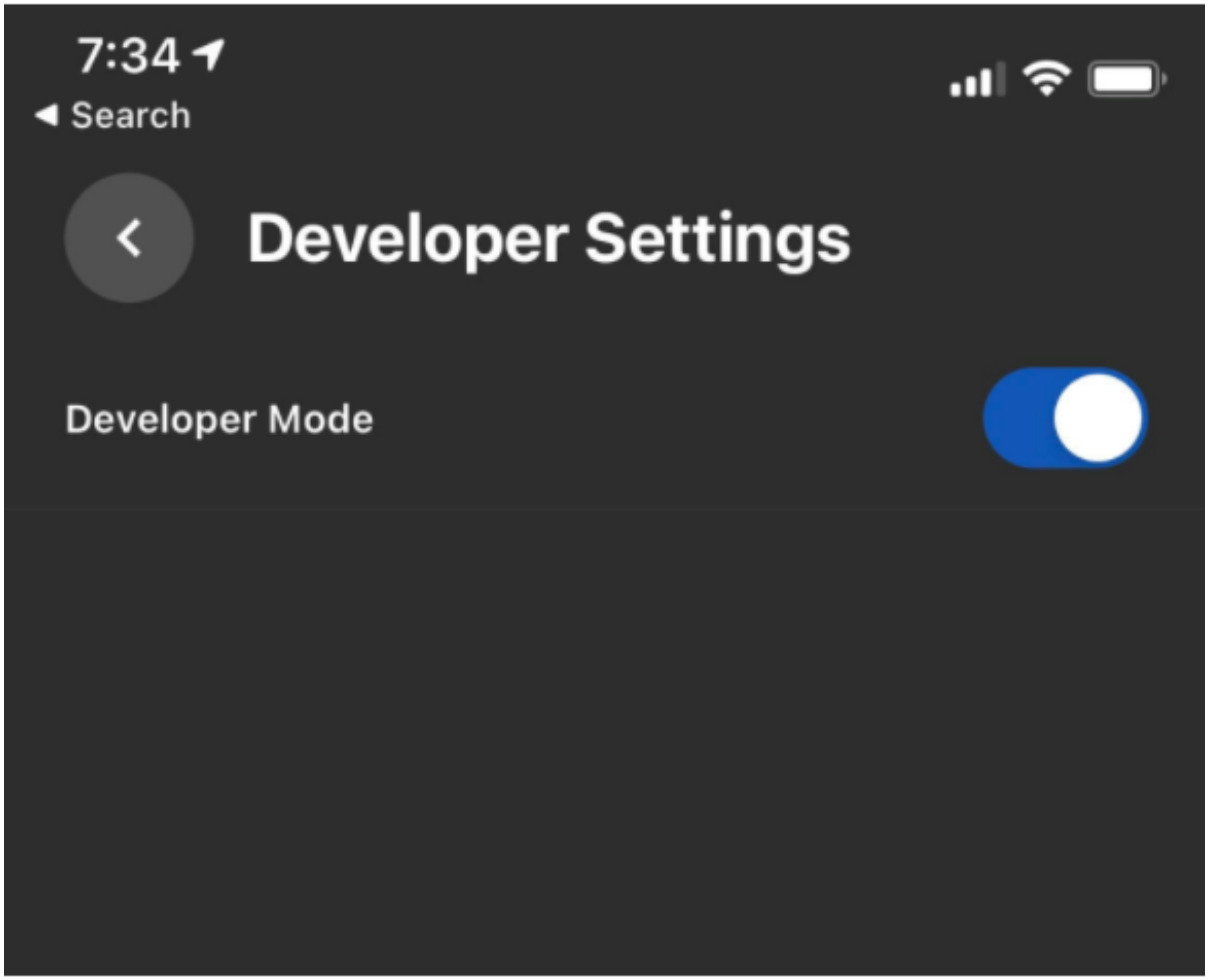
The first place I got stuck was turning on developer mode.

I didnâ€™t want to use my regular Facebook account for development. I went to the Oculus Developer Center, clicked the profile icon at top right, chose sign up, and without reading carefully, created an *unmerged Oculus developer account* which sounded like what I wanted.

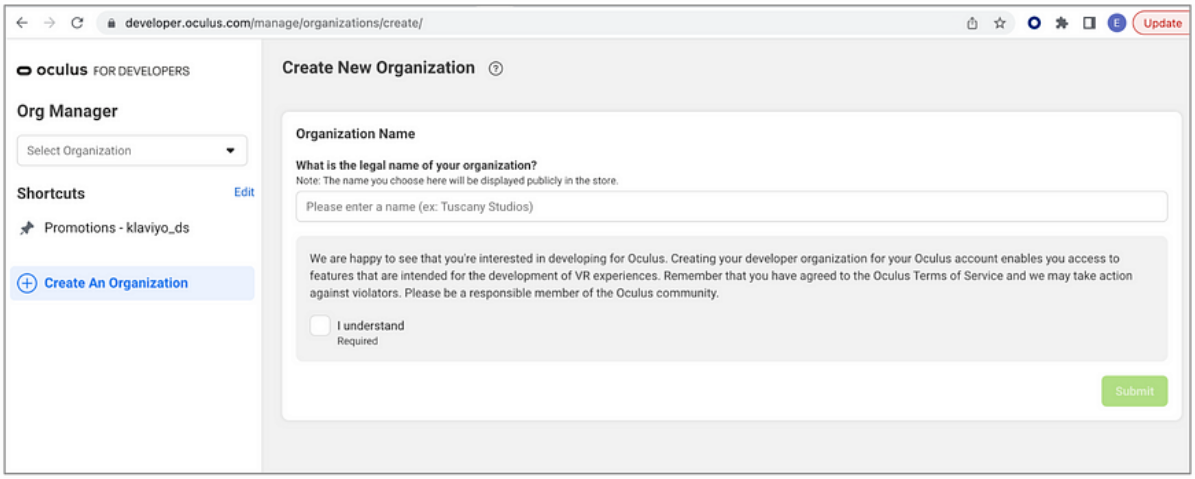


When I actually went to sign into my Quest 2 with the new account, it forced me to merge it with a Facebook account. (Now, in writing this blog post and reading the text above more carefully, I see it says: *You can create an Oculus developer account that isnâ€™t merged with a Facebook account, but you will not be able to test social, new and updated features on older Oculus devices, or log into newer devices like Quest 2 without a Facebook account or test user account.* So maybe a test user account could have helped. But thatâ€™s not the path I took. Instead, I created a new Facebook account and merged my new Oculus account with that.

So now I was signed into both my Quest 2 and the Oculus app on my phone with my new Facebook account. I started following the instructions to Enable Device for Development and Testing. In the Oculus phone app, I navigated to Menu -> Devices, confirmed I was connected to my Quest 2, tapped Developer Mode, but could not switch the toggle to on.



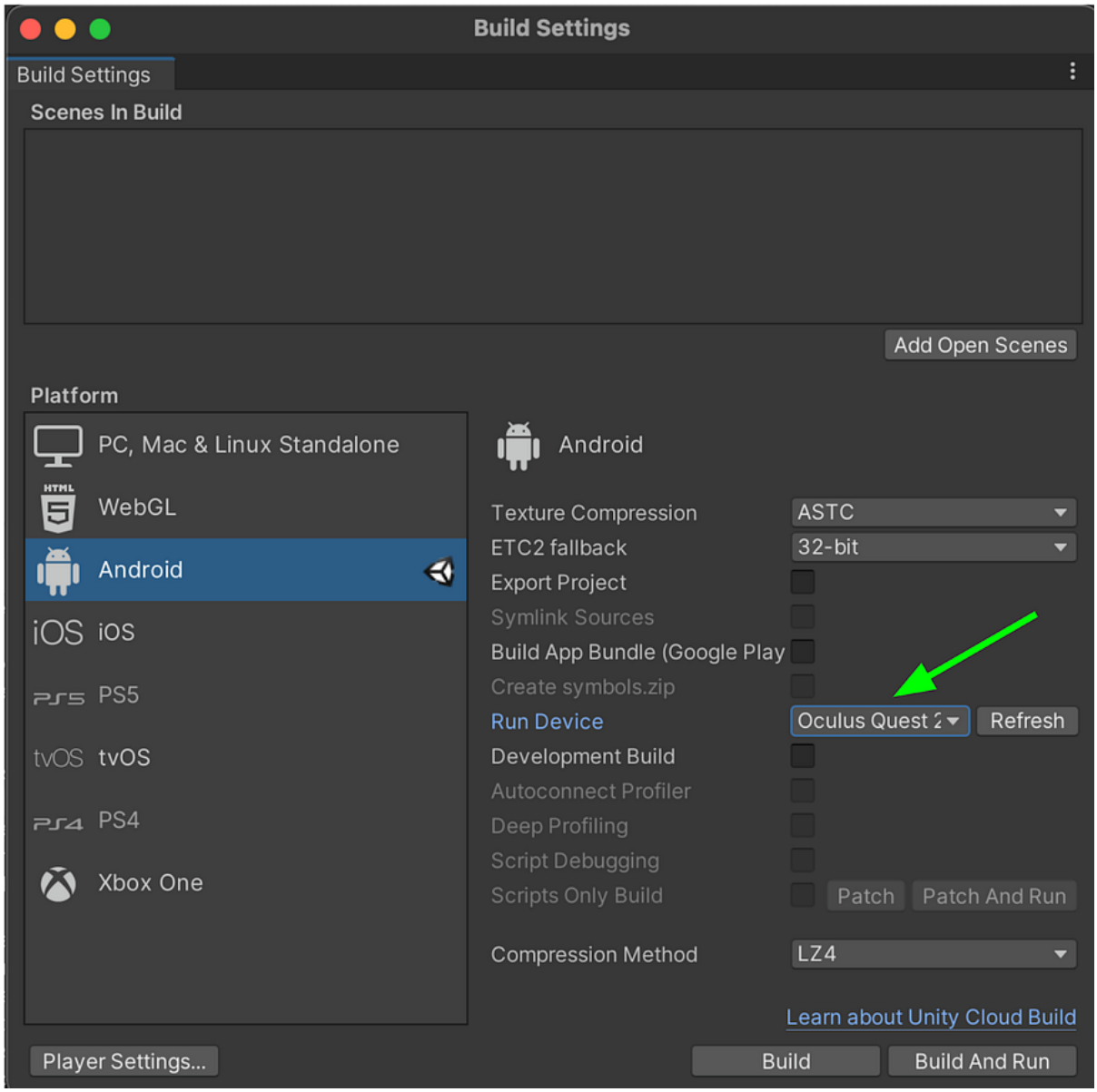
I tried many things and donâ€™t remember them all. One was creating a new organization on the Oculus developer site and then creating a new app within that organization. I now doubt that was necessary at this point, but I did need it later.



Very frustrated, I started thinking, maybe the developer account must be admin on the Quest 2. I did a factory reset of my Quest 2 and this time set things up so that the primary (admin) account

on the device was my developer Facebook account, and my â€œrealâ€ Facebook account was a secondary user. Andâ€¦voilaâ€¦I was able to toggle on Developer Mode. (So does the developer account need to be admin on the device? Or was it the factory reset? Or something else? Unsure.)

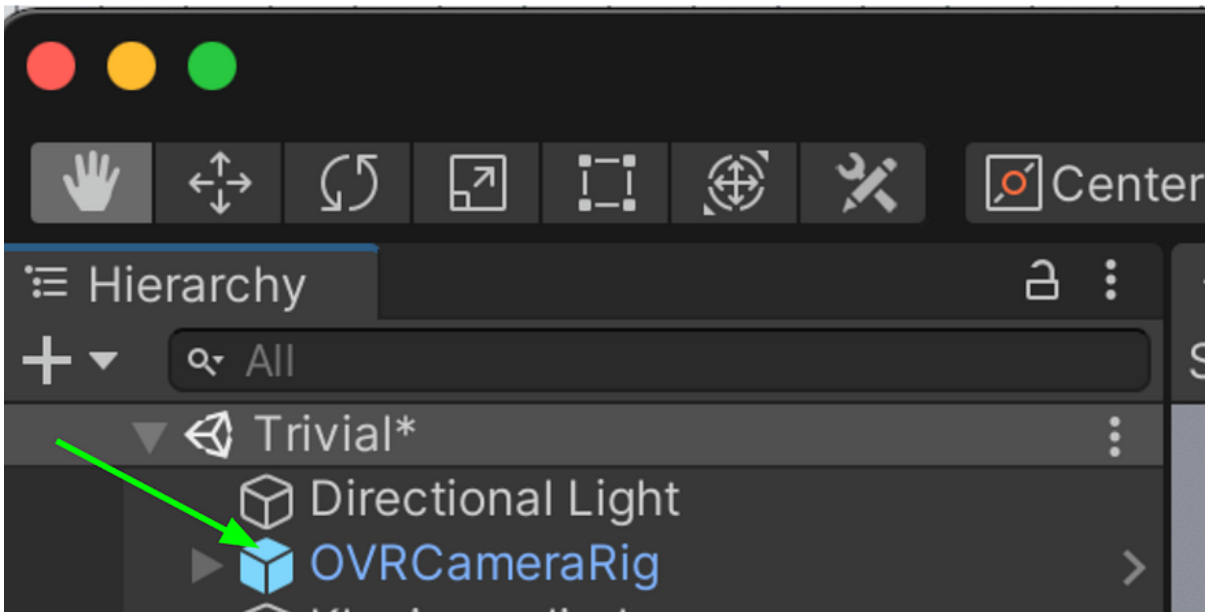
In Unity, I was then able to go to File -> Build Settings -> Android and see this:

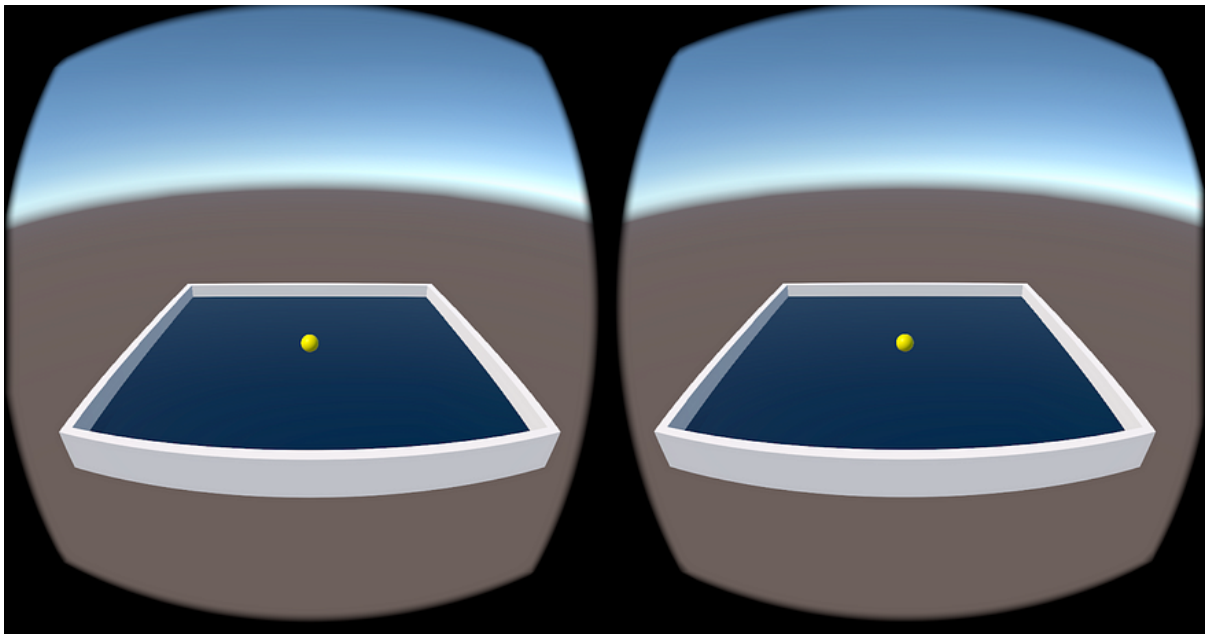


â€œBuild Your First Appâ€

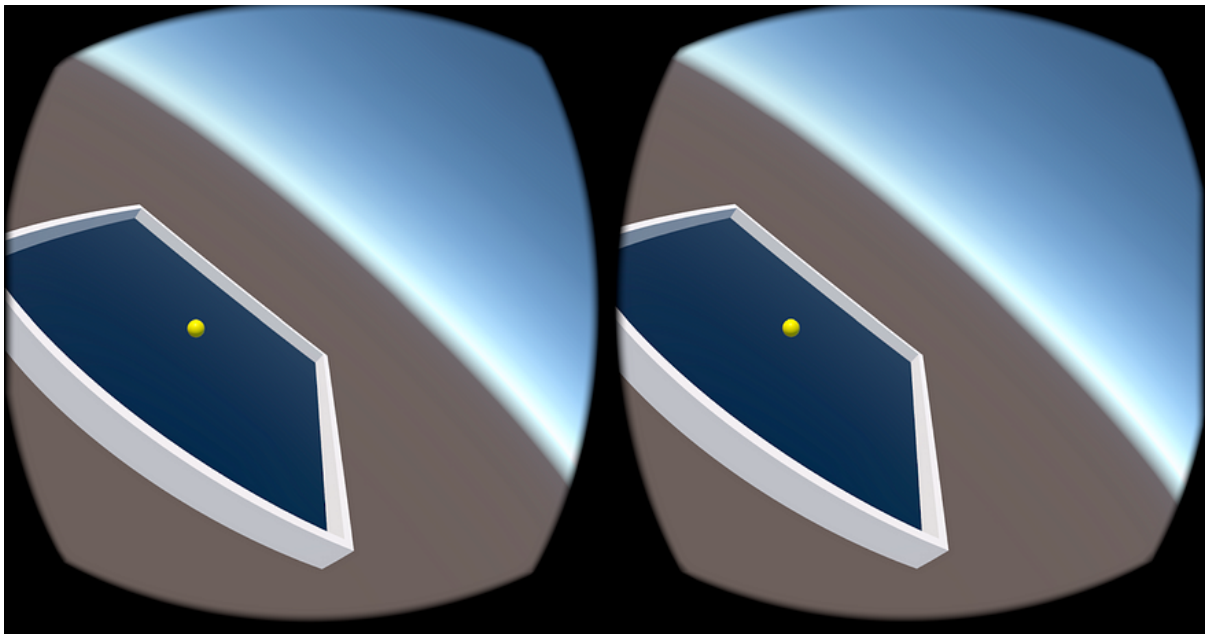
I next followed the instructions to Import Oculus Integration Package and Configure Unity Settings. Then I was ready to Build Your First VR App. As a Unity newbie I found the tutorial helpful. What confused me is that when I deployed it to my Quest 2 it didnâ€™t look like a VR app â€" when I moved my head, the whole image remained static. Now in writing this blog post I see they actually explain that at the top: *It does not use Oculus Integration package as the objective of this tutorial is to get you started with Unityâ€™s basic concepts and interface.*

However, the nice thing is that if you remove the Main Camera and drag in the OVRCameraRig, and then build and run again, now itâ€™s VR. Youâ€™re actually in the scene and as you move your head you can look around the 3D world.





Before OVRCameraRig, no matter how I turned my head, it always looked like this



After OVRCameraRig it started behaving as expected when youâ€™re in VR

Another thing I tried at this point was adding the ability to make the ball fly off the surface (i.e. along the y axis) using the thumbstick on the other controller. (3D physics simulation is incredible. I had no idea how much these game engines handle for you.)

Getting debug lines

I installed adb using brew (this was on an Intel Mac). There was a ton of log data coming off the device. I used the prefix â€œERICâ€ in all my debug messages. For example, in my C# code I added:

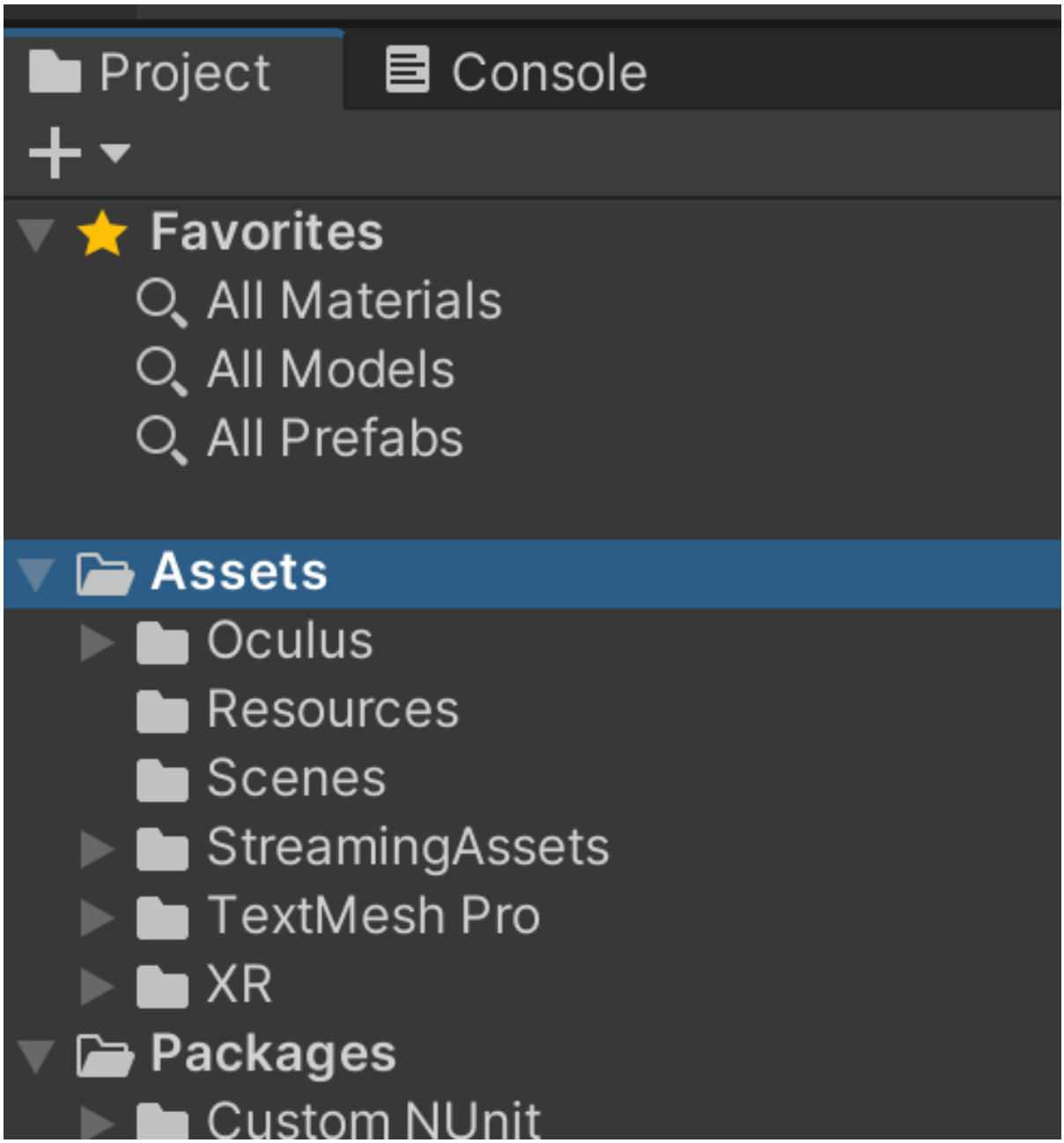
```
Debug.Log("ERIC: ON POINTER CLICK, customerNumber is " + customerNumber);
```

And ran this in my terminal:

```
/usr/local/Caskroom/android-sdk/4333796/platform-tools/adb logcat | grep -
```

Reading CSV files

I created CSV files with the event data to show in my demo. (In a real implementation this information would be downloaded or streamed from a web service.) At first I copied the CSV files into the Assets folder.



When I ran my app locally in Unity on my laptop I could read the CSV files using normal file reading. However, that didnâ€™t work when I deployed to the Quest 2. Iâ€™m not sure how all the packaging works, what files get copied, do they get copied as files or are they compressed/ bundled, if paths get moved around, what permissions code has running on the device, etc. I found this solution:

I dragged the CSV files to the Assets/Resources folder. Then within my C# code:

```
string fileData = Resources.Load<TextAsset>("events").text;
```

Which set fileData to the contents of the `events.csv` file. Note that even though the file was named `events.csv`, I passed `events` to `Load()`. My guess is when you drag the CSV file into the Resources folder in Unity it creates a text object that gets named by the name of the file minus the extension.

Dynamically attach a script to a Game Object

I represented each data point (e.g. a profile clicking on an email) as a sphere. I needed to programmatically attach my custom script to each sphere to handle when the user pointed their laser at the sphere and pulled the trigger (more on that later).

This turned out to be easy:

```
GameObject gameObject = GameObject.CreatePrimitive(PrimitiveType.Sphere);g
```

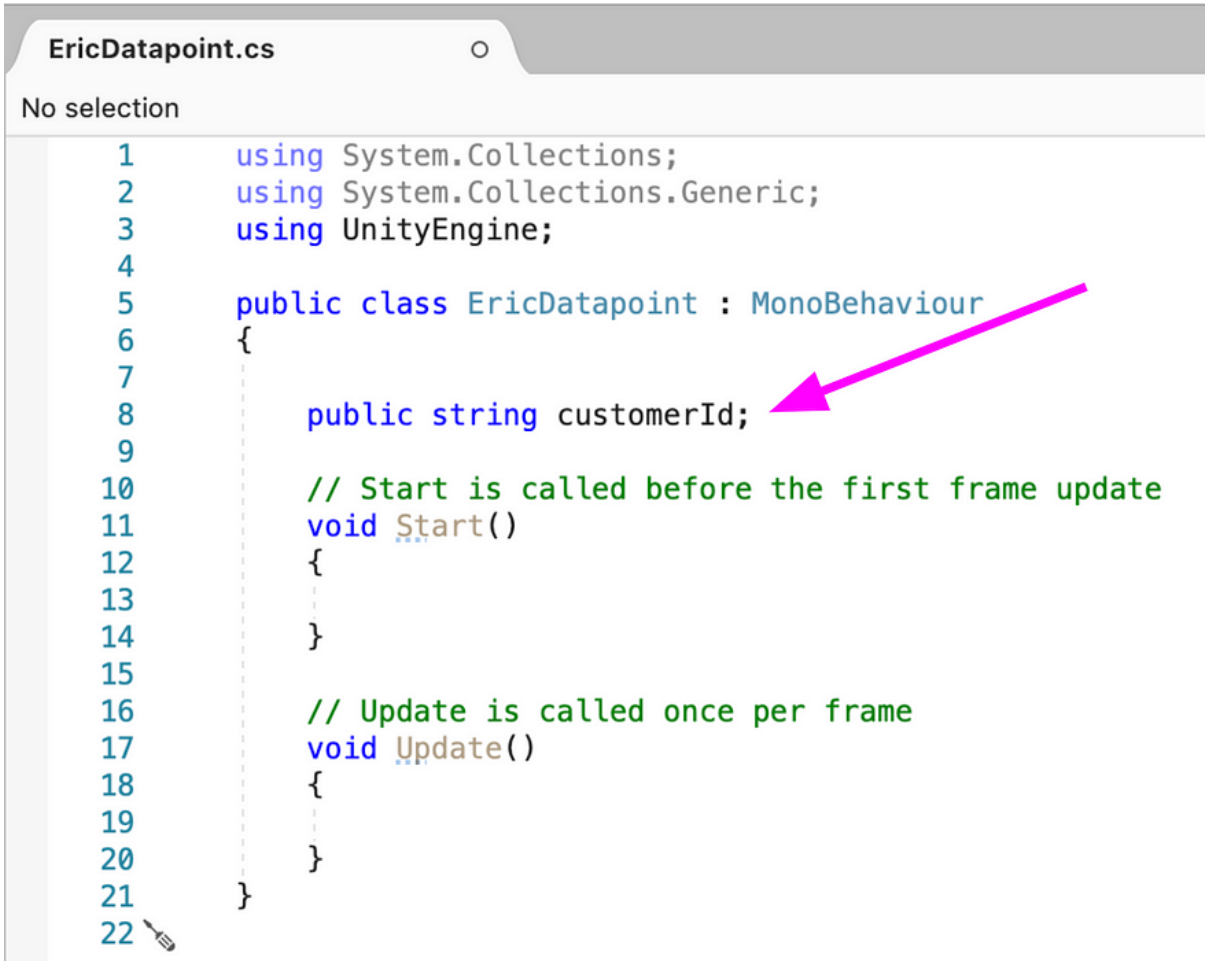
Here, `EricDatapoint` is the name of my class that was created when I added the script `EricDatapoint` with Assets -> Create -> C# Script.

I assume the angle bracket notation is a C# thing that means something like pass this type as an argument to the function and require the function to return an object of that type. So just by creating the `EricDatapoint` class in Unity it became available to other scripts in the project, and `gameObject.AddComponent<EricDatapoint>()` instantiated a new object of type `EricDatapoint` and associated it with my new GameObject. (Or maybe the type gets associated with the GameObject and then an object is instantiated?) Either way, I could then access the instance as follows:

```
EricDatapoint datapoint = gameObject.GetComponent<EricDatapoint>();
```

Associate extra info with a Game Object

Building on the above, for each data point I wanted to track extra information such as the associated customer ID. I did that with member variables on the script associated with the Game Object.



So then in my code that created a Game Object with a sphere I could do:

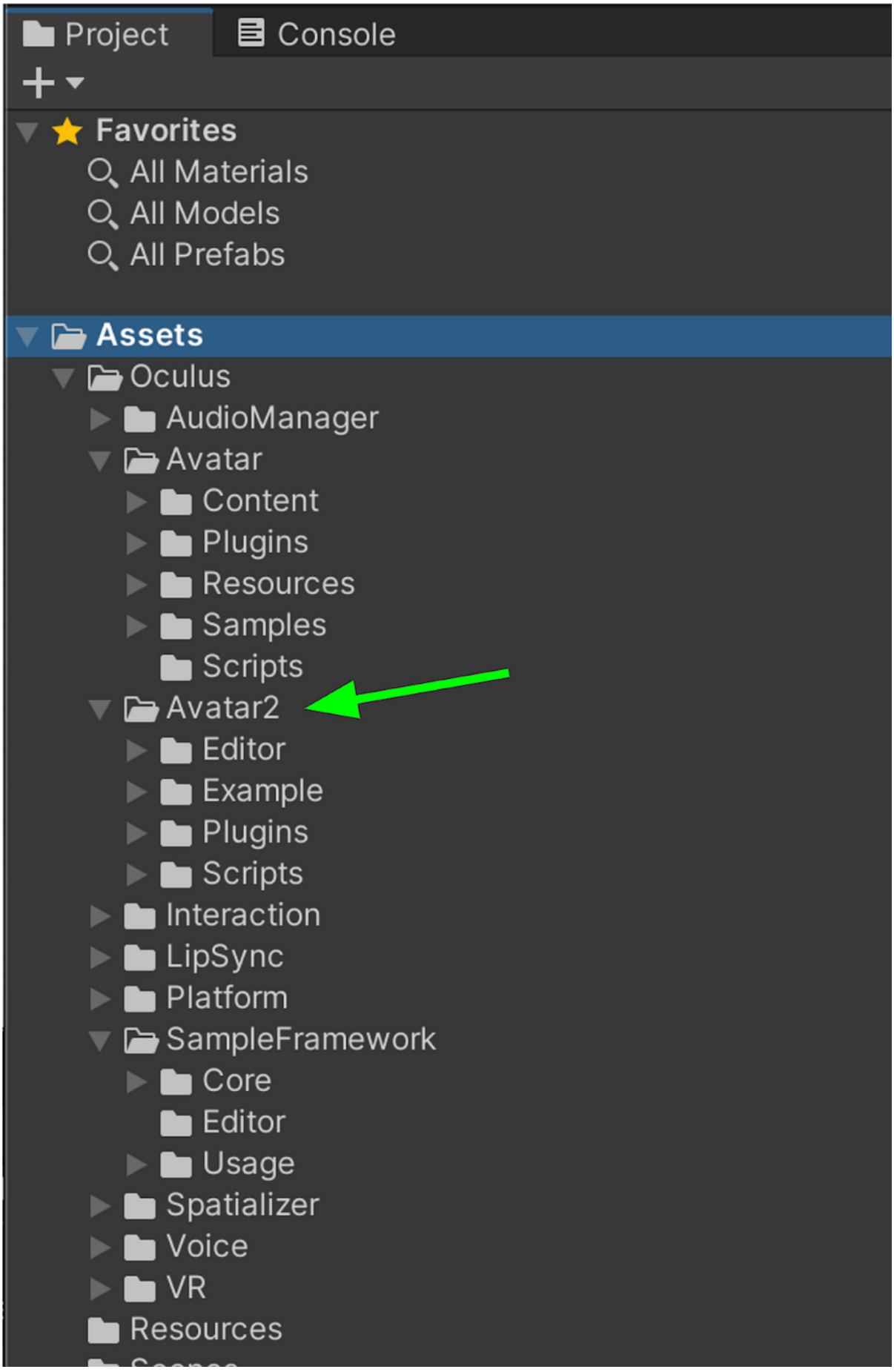
```
GameObject gameObject = GameObject.CreatePrimitive(PrimitiveType.Sphere);g
```

Code to execute when scene loads

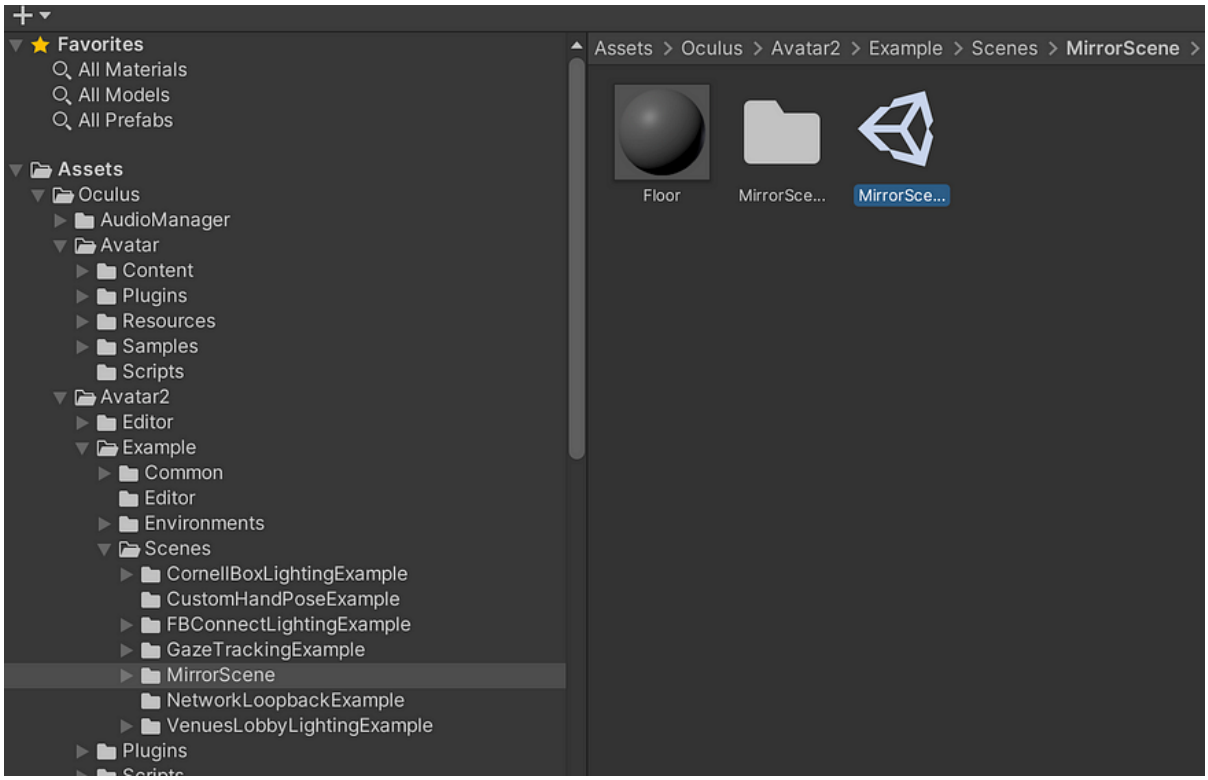
I needed code that would execute upfront to read in the CSV files and create the spheres. It looked like the `Start()` method on any script associated with any active game object would work. To keep things organized, I made a new game object called *Grapher* (GameObject -> Create Empty) and added a single component, the `EricDatapoint` script.

Meta Avatars, Meta Avatars SDK, AvatarSDK2

I spent a lot of time trying to understand this. I wanted to get an avatar in my scene. I had an *Avatar* folder in Assets from whatever steps I followed above, but read somewhere that that was out of date and to instead use â€œMeta Avatars.â€ That was confusing, because wasnâ€™t all of this from Meta? Eventually I downloaded Oculus Developer Hub and used that to install the Meta Avatars SDK. I believe thatâ€™s what added the *Avatar2* folder shown below and the *Avatar2SDK* menu to Unity.

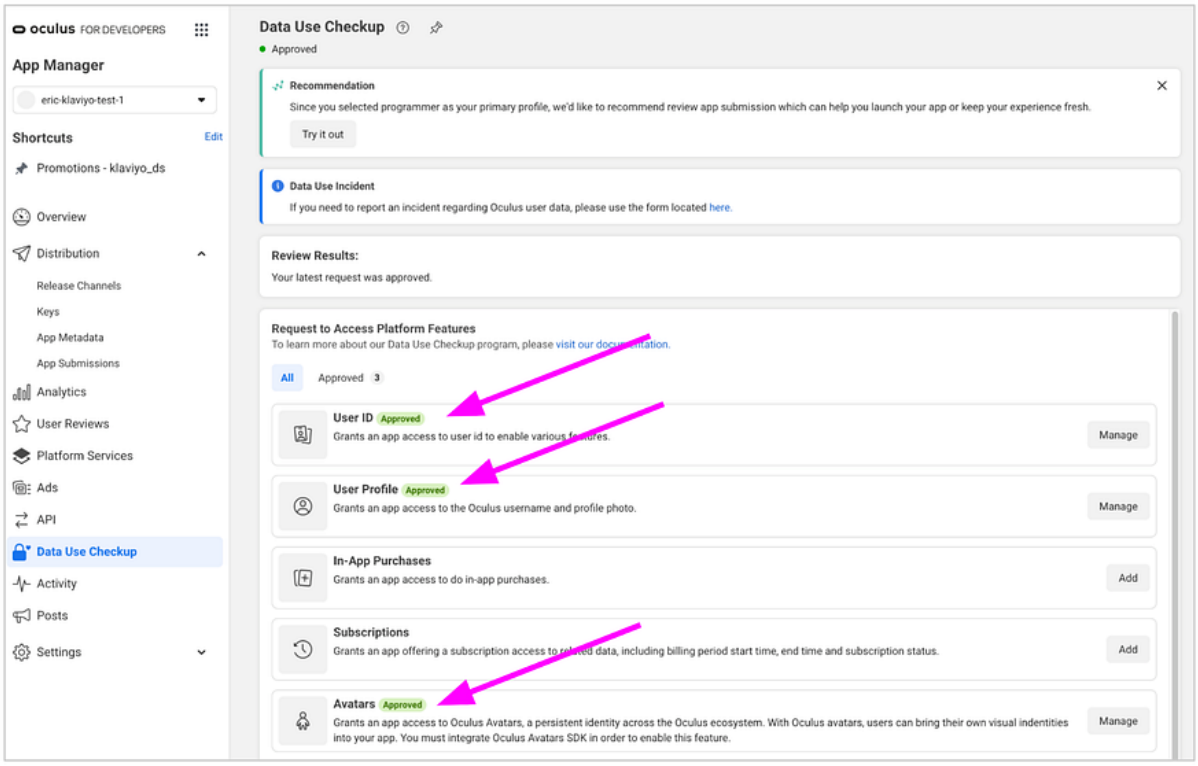

Project
Console
Favorites
All Materials
All Models
All Prefabs
Assets
Oculus
AudioManager
Avatar
Content
Plugins
Resources
Samples
Scripts
Avatar2
Editor
Example
Plugins
Scripts
Interaction
LipSync
Platform
SampleFramework
Core
Editor
Usage
Spatializer
Voice
VR
Resources

I dragged the example MirrorScene into my hierarchy and was able to see an avatar mirroring my movements, but it wasnâ€™t the avatar I had configured in my account on the Quest 2.

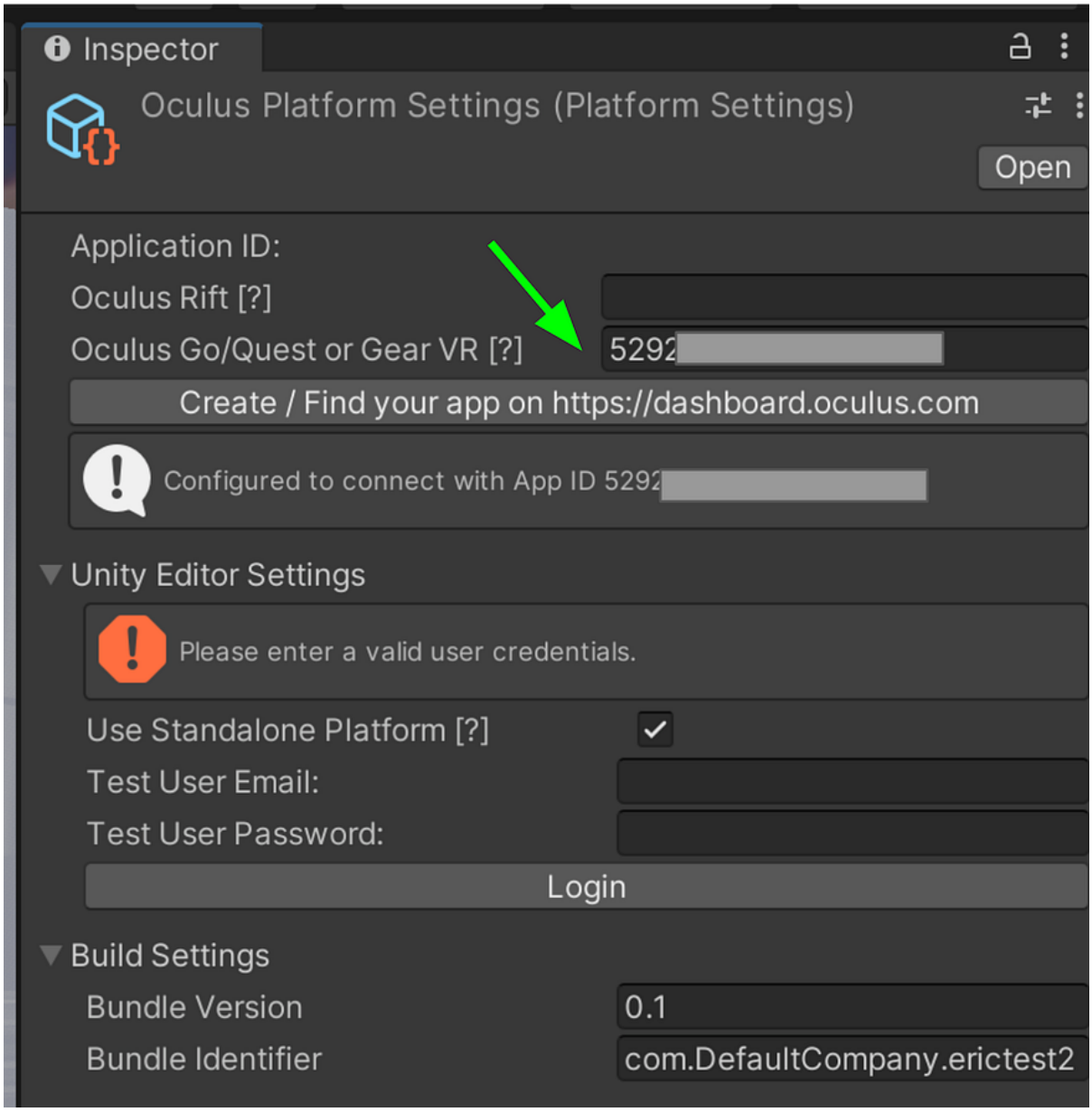


I added debugging, tried a bunch of things, and Iâ€™m not sure which made the difference, but in the end I got it working. Here are the things I did:

a) I requested to use certain features for my app in the developer portal:



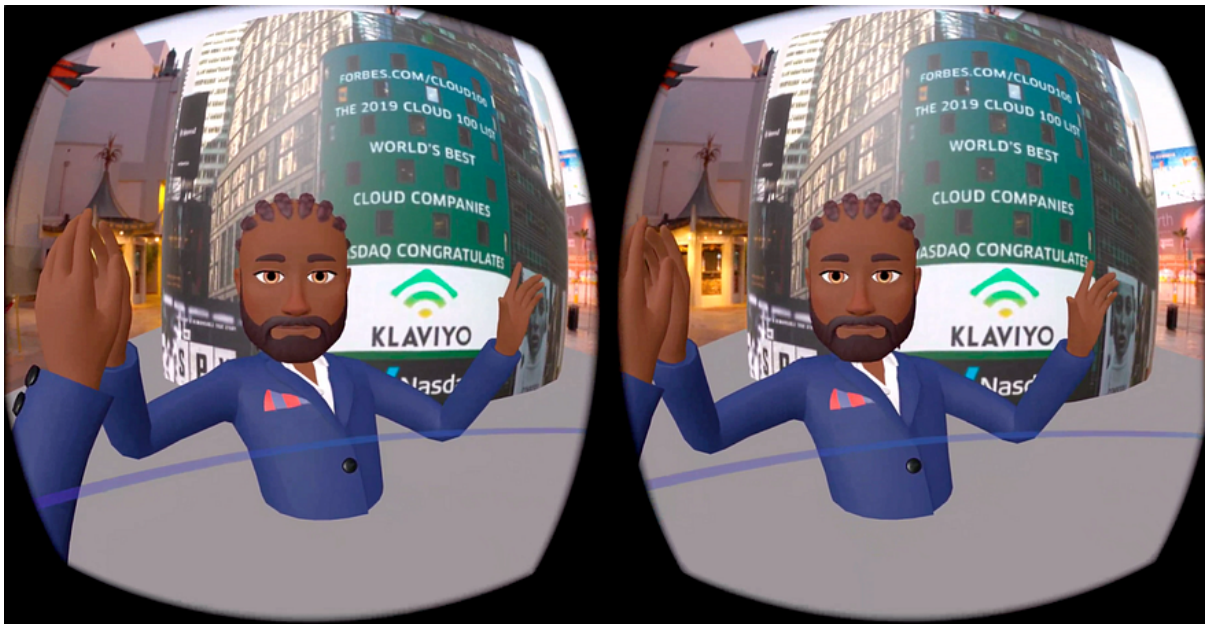
b) In that same screen, I went to API, copied the App ID, then went into Unity, selected Oculus -> Platform -> Edit Settings and pasted the ID in as the Oculus Go/Quest or Gear VR Application ID:



c) Restarted the Quest 2.

After that my correct avatar showed up. This code to launch the avatar editor also started working:

```
AvatarEditorDeeplink.LaunchAvatarEditor();
```

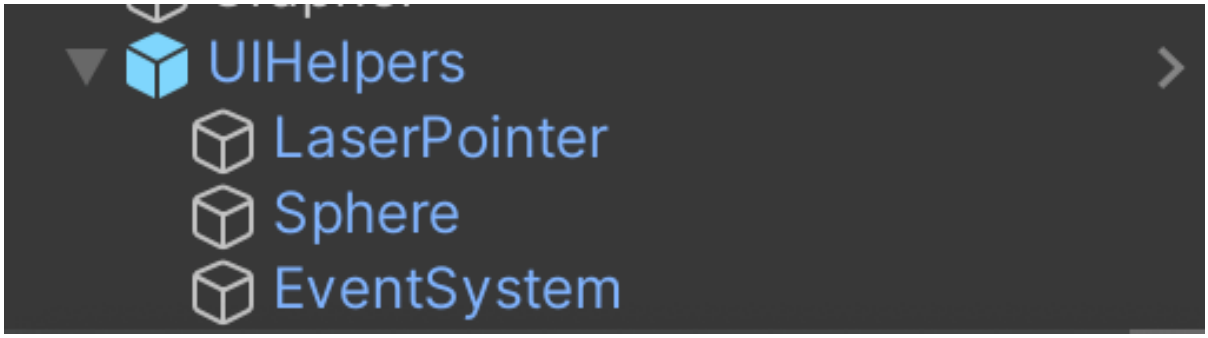


Finally! The avatar I chose for my development account on the Quest 2 showed in the example MirrorScene!

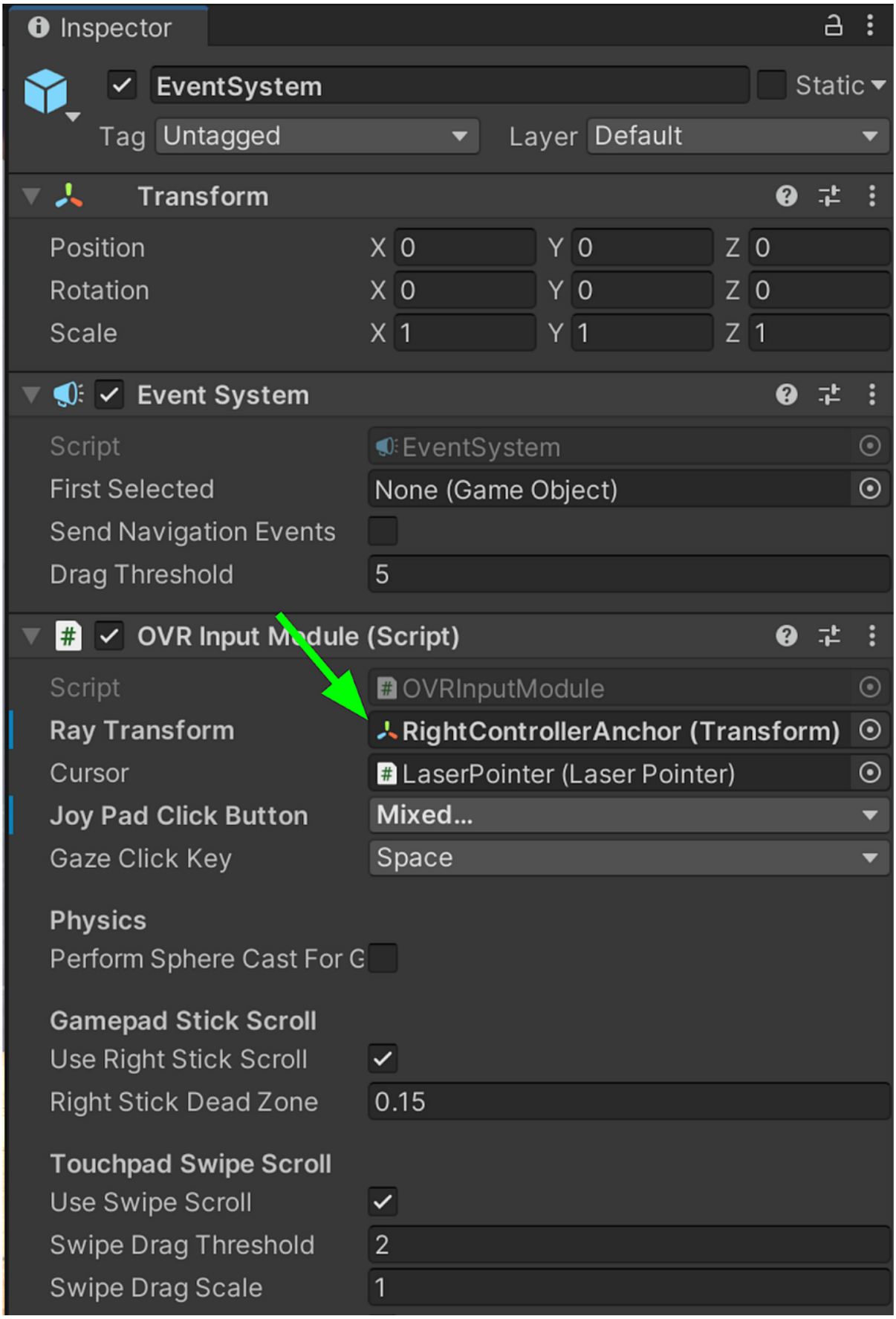
Using a laser to point and click

I wanted to make it so my avatar (the hand of the first-person avatar, not the mirror avatar facing us in the scene above) could use a laser to point and click any of the spheres representing the data points. I couldnâ€™t figure out how to get this working with the Meta Avatar I used above (more on that below) and so instead made it work in what I imagine is the â€œold wayâ€ like this:

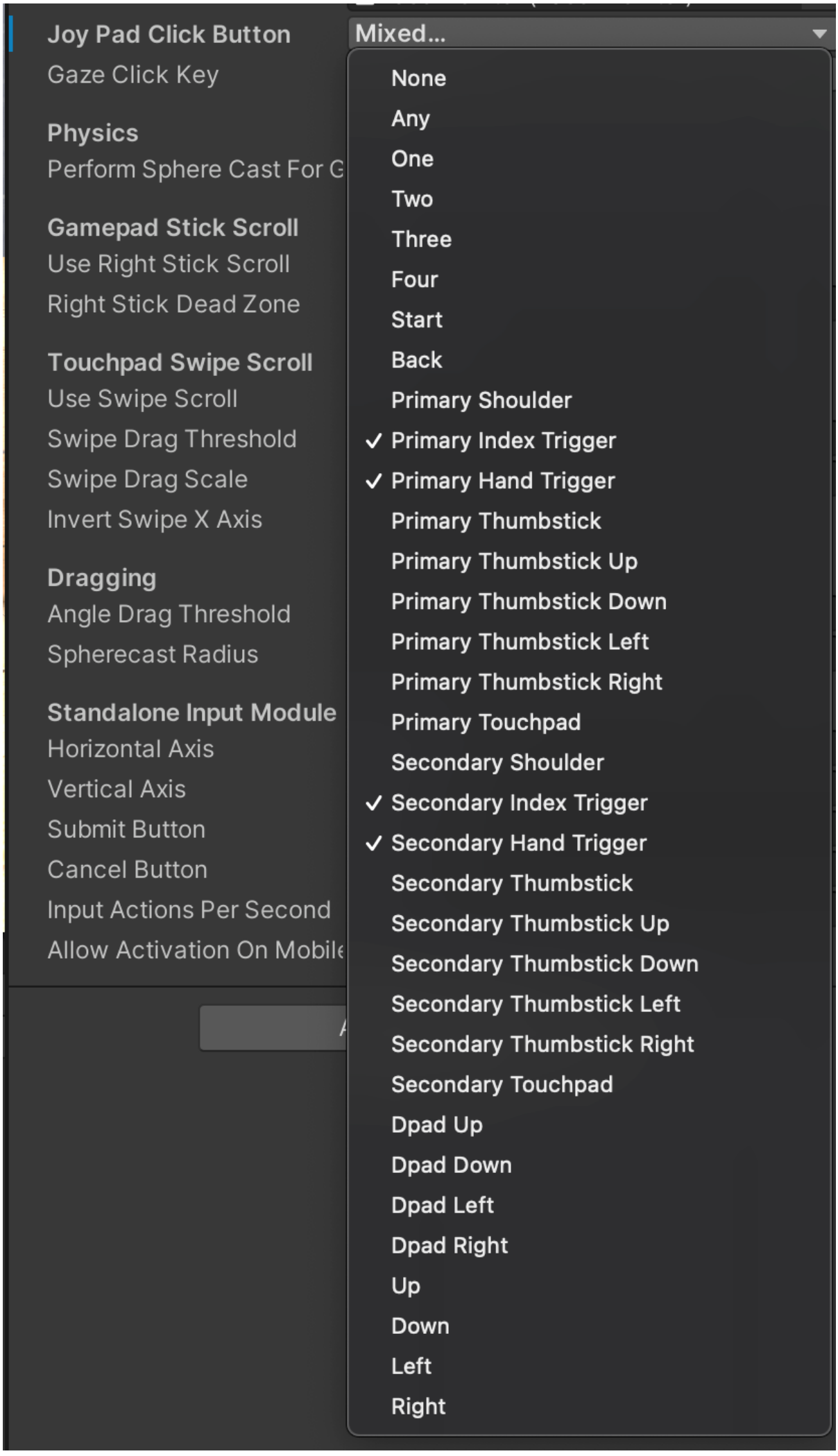
First I brought the UIHelpers prefab into my scene:



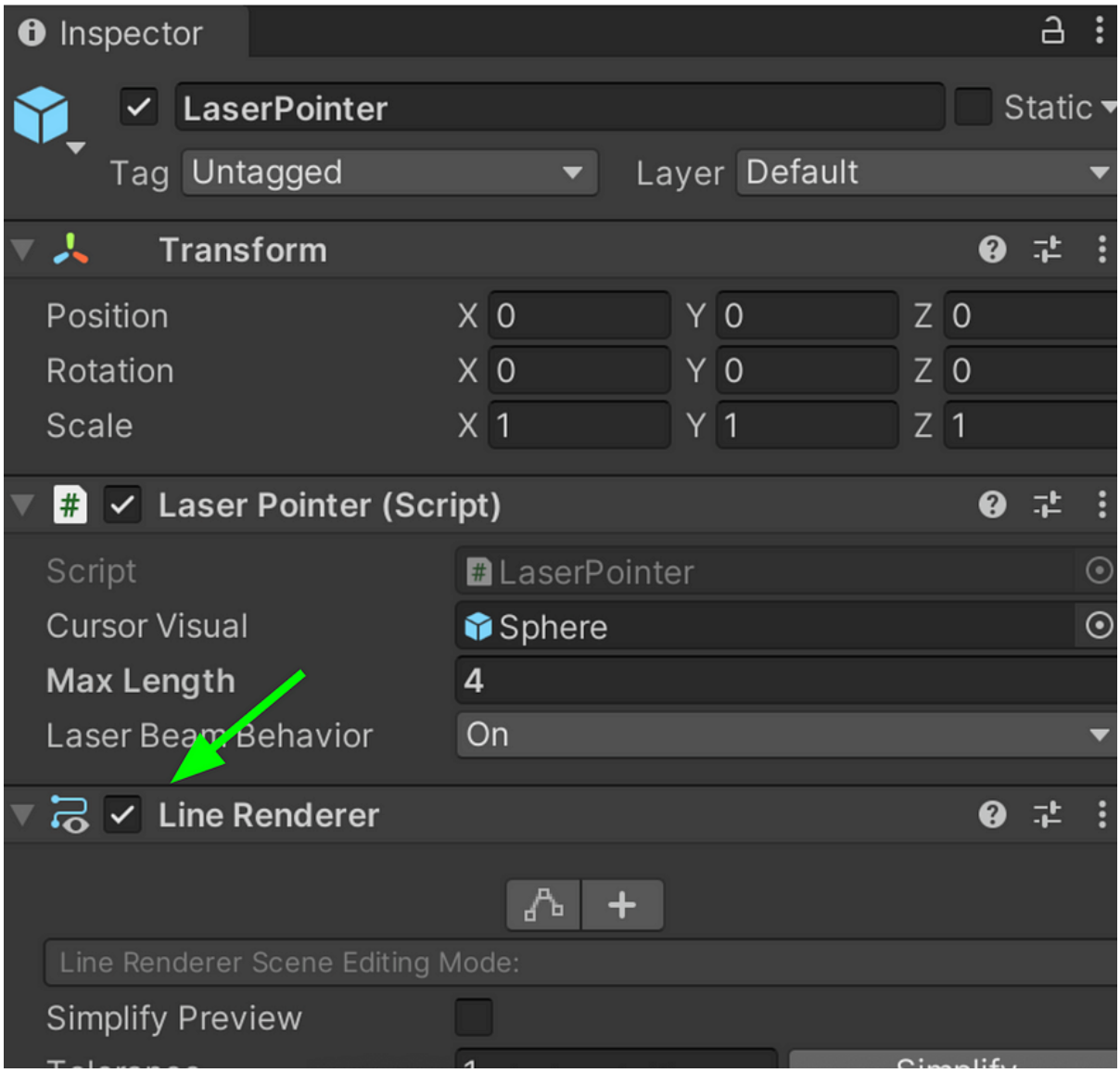
Then I clicked on EventSystem and under OVRInput Module (Script) set the Ray Transform to the RightControllerAnchor (a Game Object in the hierarchy). I understood this to mean have the laser start at the right controller.



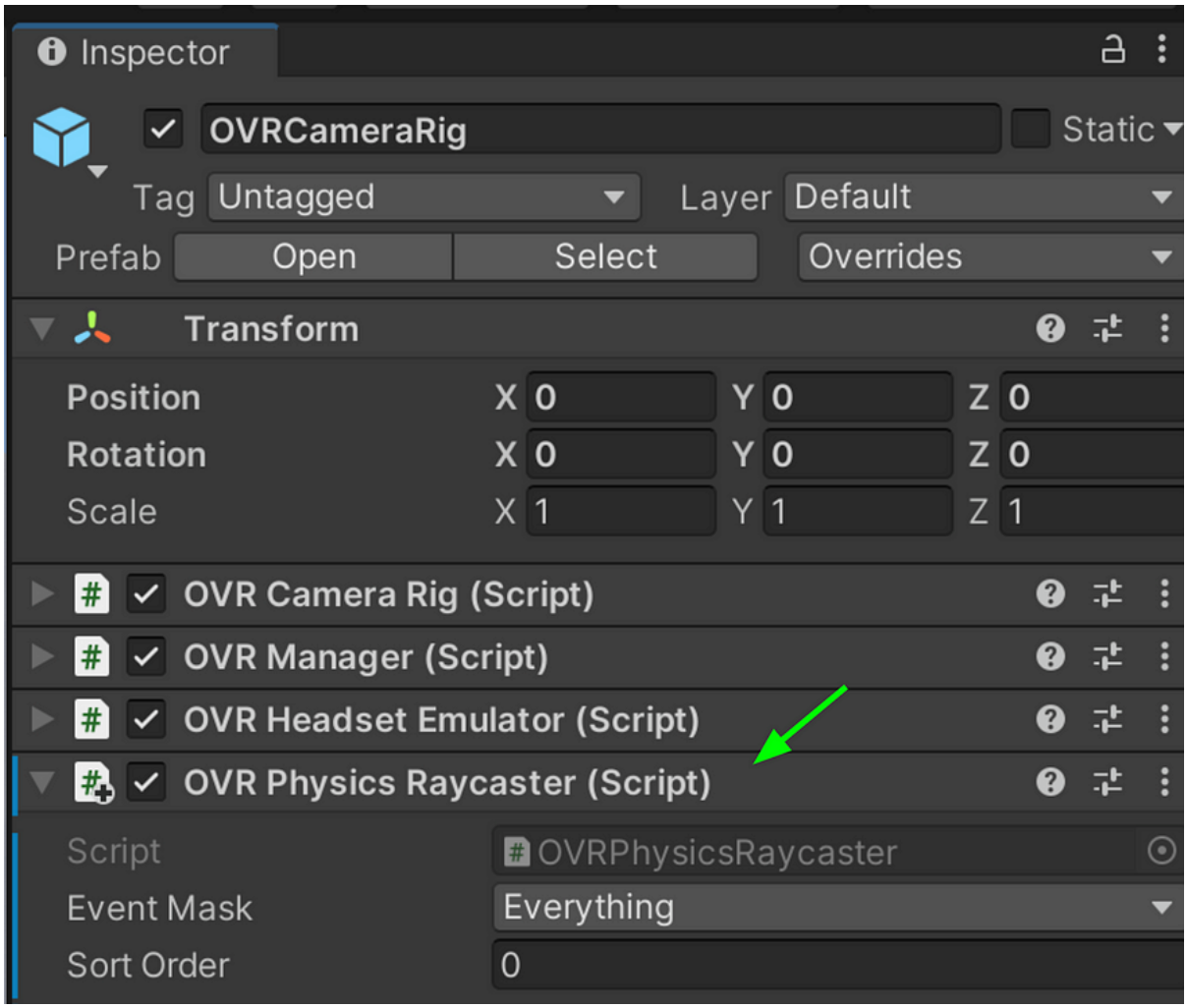
I changed the Joy Pad Click Button to only the two trigger buttons because I wanted the other buttons to do other things in my demo.


Joy Pad Click Button
Gaze Click Key
Physics
Perform Sphere Cast For G
Gamepad Stick Scroll
Use Right Stick Scroll
Right Stick Dead Zone
Touchpad Swipe Scroll
Use Swipe Scroll
Swipe Drag Threshold
Swipe Drag Scale
Invert Swipe X Axis
Dragging
Angle Drag Threshold
Spherecast Radius
Standalone Input Module
Horizontal Axis
Vertical Axis
Submit Button
Cancel Button
Input Actions Per Second
Allow Activation On Mobile
Mixed...
None
Any
One
Two
Three
Four
Start
Back
Primary Shoulder
✓ Primary Index Trigger
✓ Primary Hand Trigger
Primary Thumbstick
Primary Thumbstick Up
Primary Thumbstick Down
Primary Thumbstick Left
Primary Thumbstick Right
Primary Touchpad
Secondary Shoulder
✓ Secondary Index Trigger
✓ Secondary Hand Trigger
Secondary Thumbstick
Secondary Thumbstick Up
Secondary Thumbstick Down
Secondary Thumbstick Left
Secondary Thumbstick Right
Secondary Touchpad
Dpad Up
Dpad Down
Dpad Left
Dpad Right
Up
Down
Left
Right

I also checked the Line Renderer on LaserPointer:



Added OVR Physics Raycaster as a component of IVRCamera rig:



Thenâ€¦and this a bit to figure outâ€¦I went back to the `EricDatapoint` C# script that I was attaching to every sphere and made it implement `IPointerClickHandler`. To do that I also needed to add `using UnityEngine.EventSystems;` Then I implemented `OnPointerClick()` and saw the expected debug output when I clicked with the laser pointer.

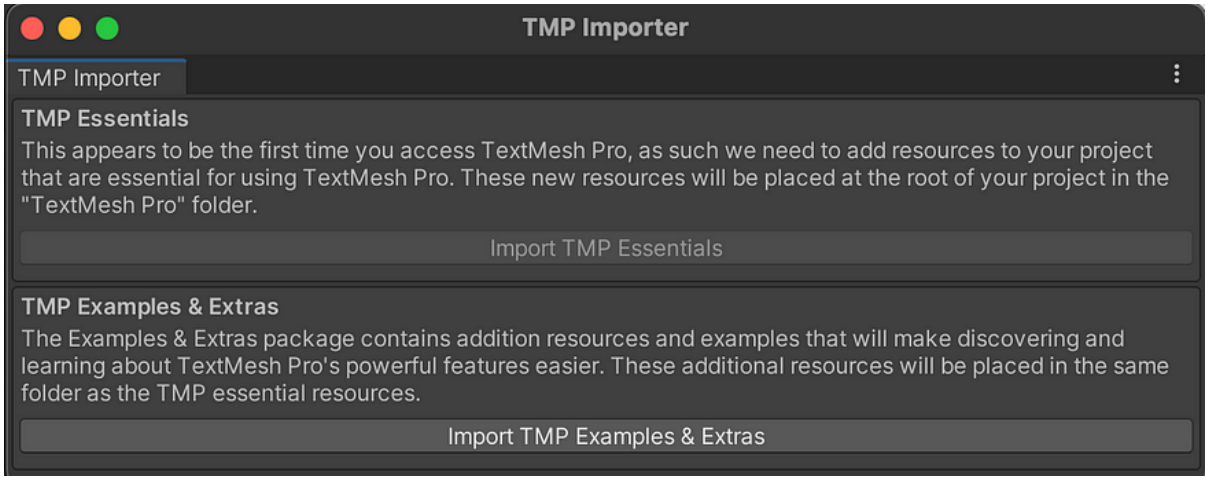
EricDatapoint.cs

No selection

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.EventSystems;
5
6   public class EricDatapoint : MonoBehaviour, IPointerClickHandler
7   {
8
9       public string customerId;
10
11      // Start is called before the first frame update
12      void Start()
13      {
14
15      }
16
17      // Update is called once per frame
18      void Update()
19      {
20
21      }
22
23      public void OnPointerClick(PointerEventData pointerEventData)
24      {
25          Debug.Log("ERIC: OnPointerClick() called with customerId " + customerId);
26      }
27  }
```

Inserting Text

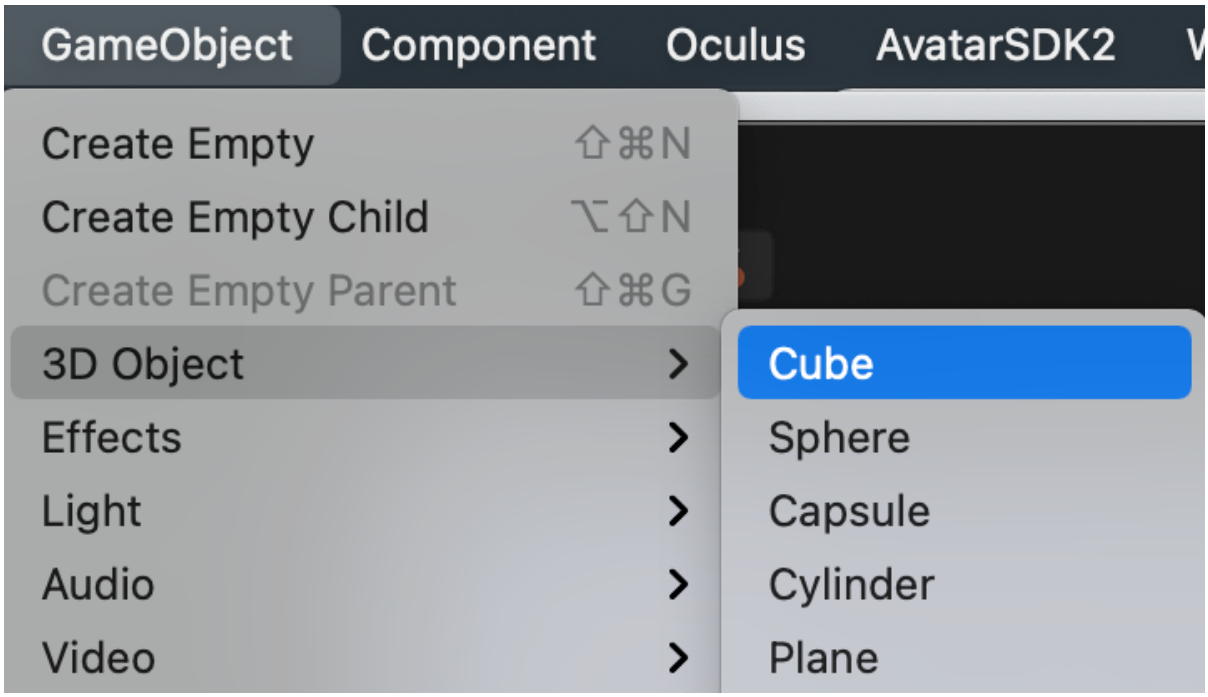
I needed text in a few places, for example, in the interface that shows details when you click on a sphere and as date labels along the horizontal floor. I wasnâ€™t sure if I should use GameObject -> 3D Object -> 3D Text, GameObject -> 3D Object -> Text - TextMeshPro, or GameObject -> UI -> Text. I tried â€œText - TextMeshProâ€¦ and it showed me this dialog and I thought â€” I must be going in the wrong direction, I just want to insert basic text. However, after playing around a bit, it does seem like TextMeshPro works best and you donâ€™t futz with font size and scaling to get it to look good.



Misc things that kept confusing me

What type of Game Object is this?

One thing that kept throwing me off is I would look at one of the example scenes to see how something worked, look at one of the Game Objects in the hierarchy, and be frustrated because I couldnâ€™t see where in the inspector it said what type of Game Object it was (e.g. a cube, a sphere). Once I realized that Game Objects are mostly containers for components things started to make more sense.



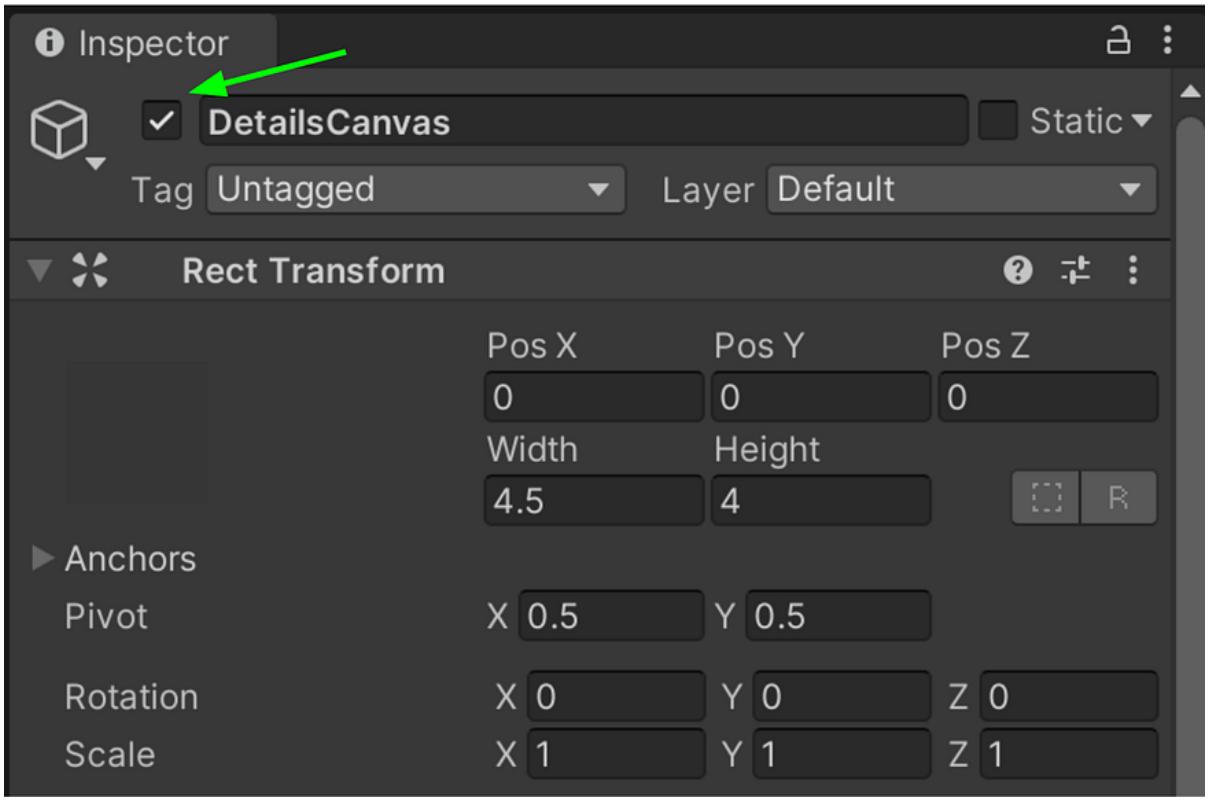
So when you choose GameObject -> 3D Object -> Cube, youâ€™re not creating a â€œCubeâ€ Game Object. Itâ€™s more like a shortcut that creates a Game Object and adds a Mesh Filter component for a cube, a Mesh Renderer component, and a Box Collider component. You could do all of that manually. It seems though like the transform, tags, and a few other things are part of the Game Object itself.

Why canâ€™t I programmatically get that (inactive) Game Object?

I created a UI that shows up when you click with the laser pointer. In some cases, for example, if I didnâ€™t have an image to show, I made the Game Object containing my Raw Image component inactive. But then I couldnâ€™t â€œfindâ€ it from my script using `Find()` like this:

```
detailsImage = GameObject.Find("DetailsImage");
```

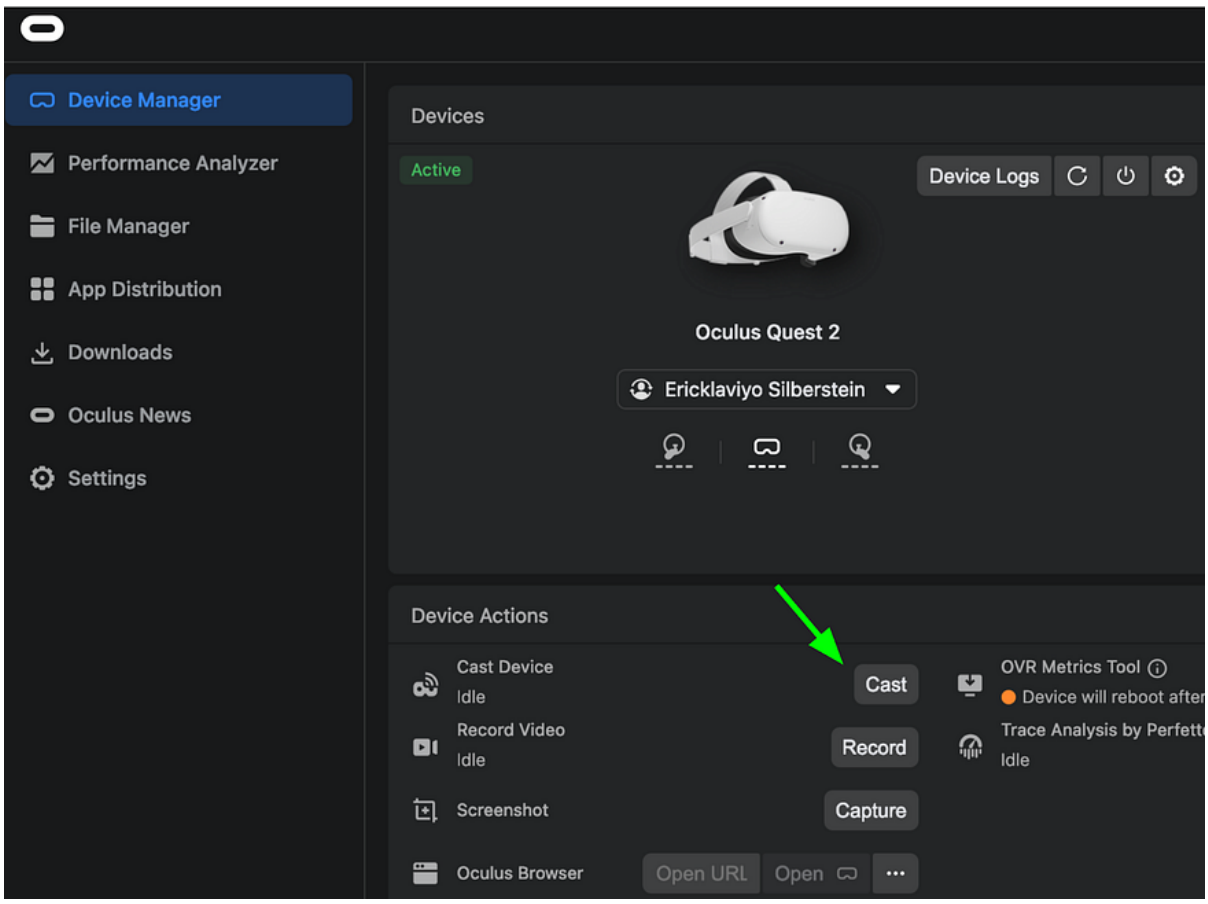
It seemed like Game Objects could be active or inactive, and when inactive, they werenâ€™t findable. I also realized this checkbox means active. (Why is there no tooltip? Maybe itâ€™s such a basic part of Unity and that everyone just knows it?)

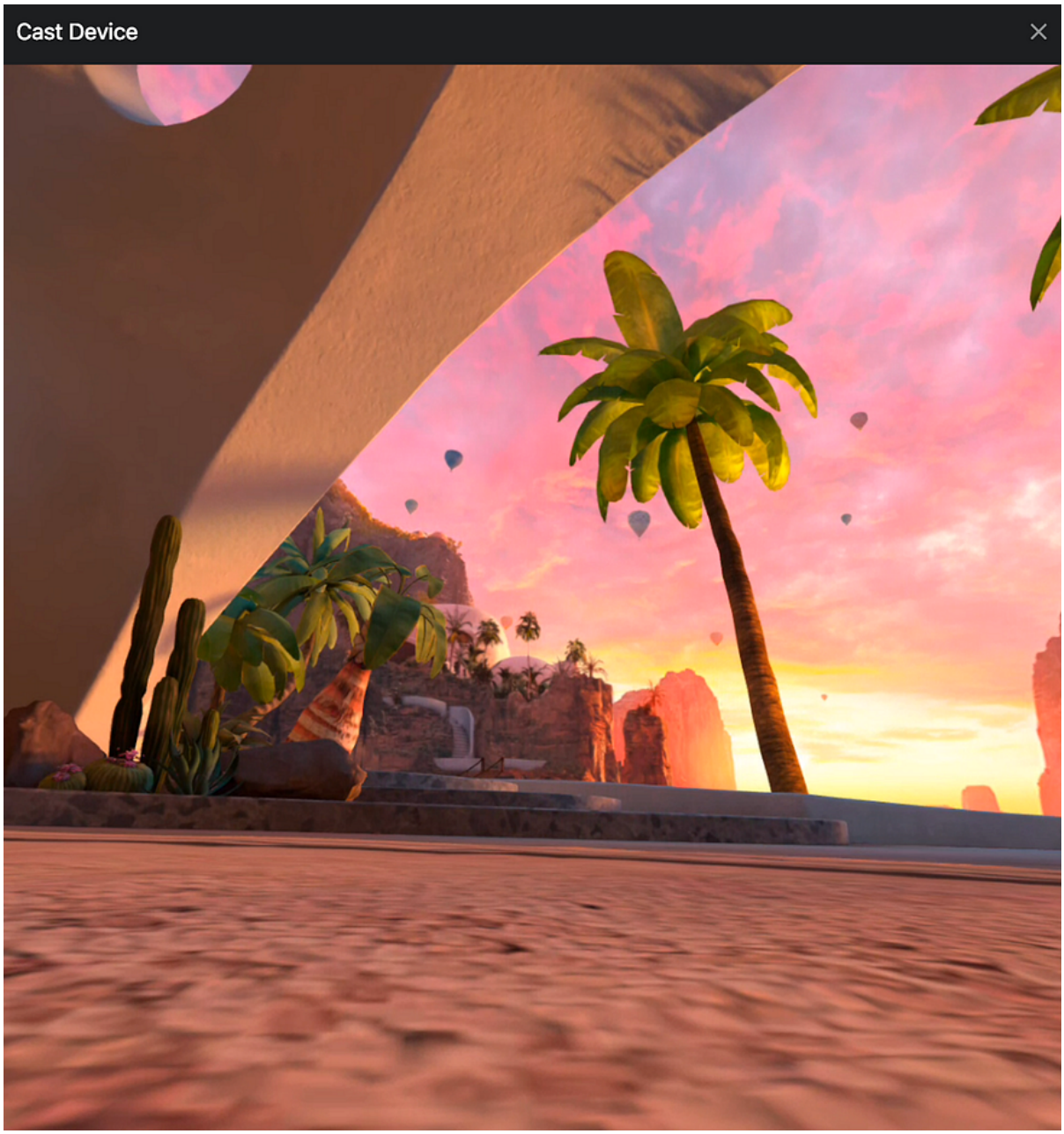


Casting to Oculus Developer Hub

To show my demo over Zoom, I wanted to cast what I was seeing in the Quest 2 to my laptop. I had no problem recording videos and taking screenshots, but cast wasnâ€™t working. A red dot indicating casting appeared in the Quest 2 but in the Oculus Developer Hub it hung on â€œWaiting for Connectionâ€¦â€ Perhaps video and screenshots use developer mode functions over the USB cable, but cast is using the same technology you use to cast to the Oculus iPhone app?

Casting to my phone wasnâ€™t working either. (Last week my daughter was showing a friend Beat Saber and casting to her phone so I knew that should work!) In troubleshooting it mentioned being sure to be on the same wifi which I wasnâ€™t. I switched my phone and laptop to the same wifi network as my Quest 2. It still didnâ€™t work. I then restarted the Quest 2 and then was able to cast to the phone and Oculus Developer Hub. So who knowsâ€¦


Device Manager
Performance Analyzer
File Manager
App Distribution
Downloads
Oculus News
Settings
Devices
Active
Device Logs
Oculus Quest 2
Ericklaviyo Silberstein
Device Actions
Cast Device
Idle
Cast
OVR Metrics Tool
Device will reboot after
Record Video
Idle
Record
Trace Analysis by Perfett
Idle
Screenshot
Capture
Oculus Browser
Open URL
Open

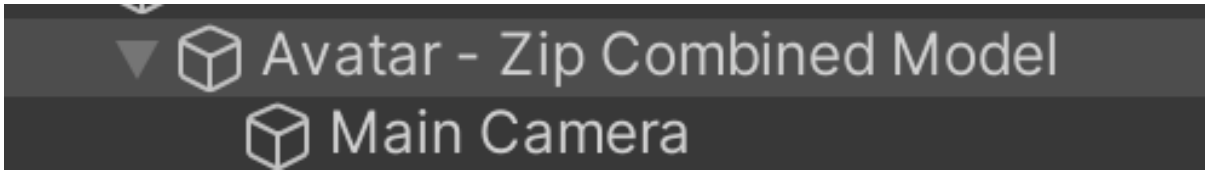


Yes! Itâ€™s casting.

Laser pointer with Meta avatars

I couldnâ€™t get this working by the end of the weekend and gave up. I wanted the nice Meta avatar that took so long to get figured out above to be able to point at my spheres with the laser pointer. It seemed like that should be simple, one of the basic things these avatars can do. I bet Iâ€™m missing something obvious.

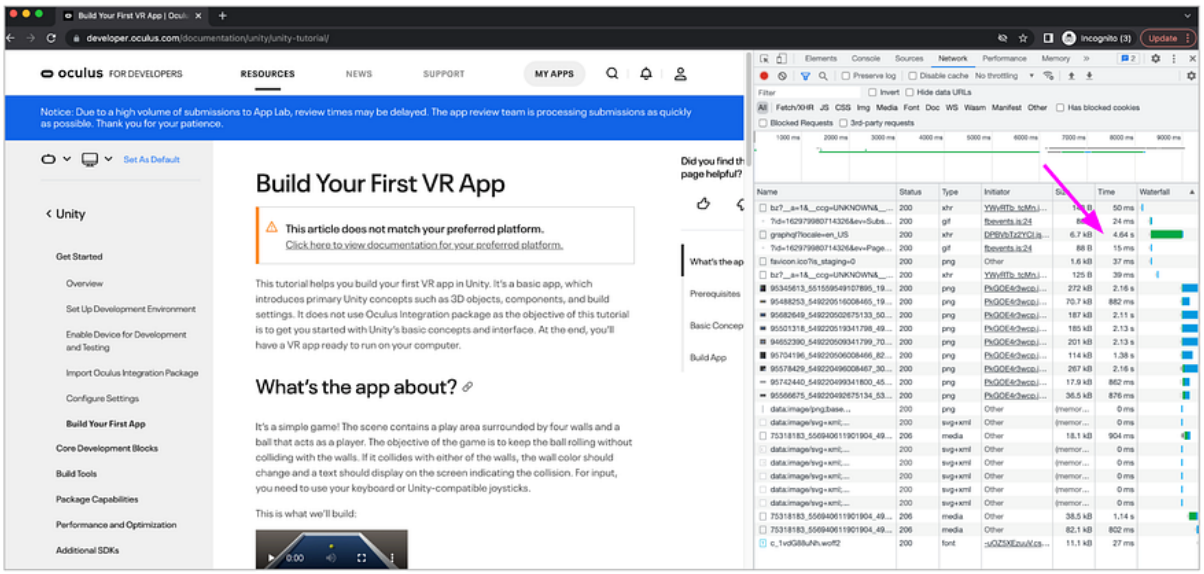
The Meta avatar, at least in the sample mirror scene, uses *Avatar - Zip Combined Model*:



I donâ€™t understand what that is or how to get the transform from it corresponding to the right hand or controller, which is where the laser pointer should start. I also donâ€™t understand why it has a â€œMain Cameraâ€ instead of an OVRCameraRig.

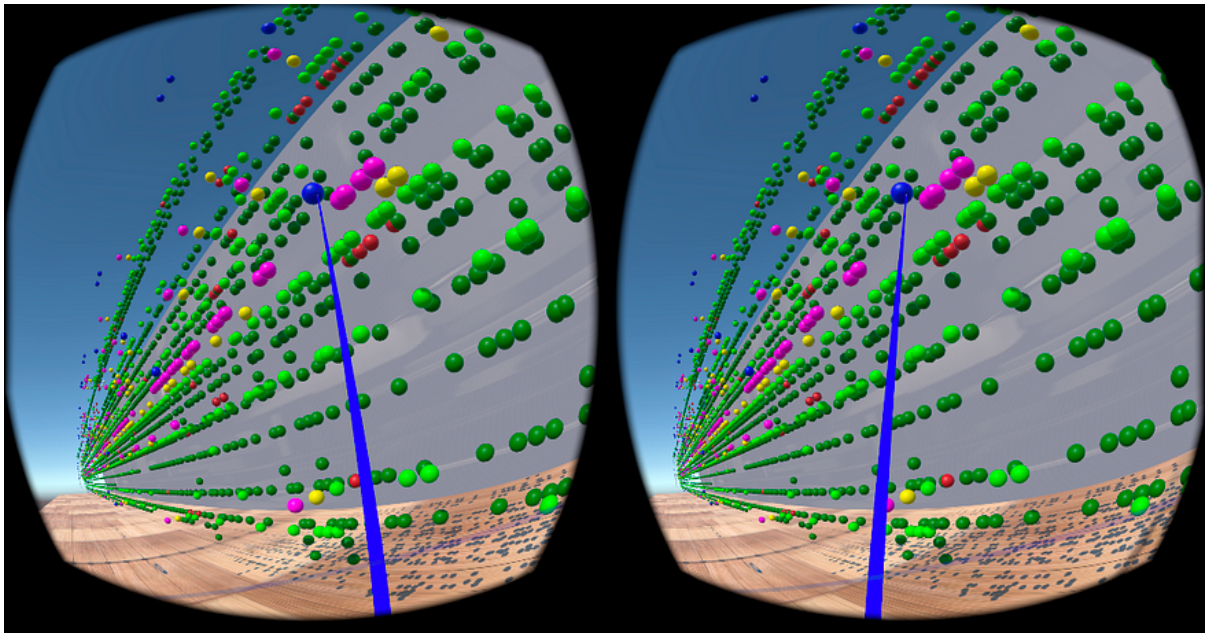
Some complaining

Câ€™mon Metaâ€¦make the developer site faster!

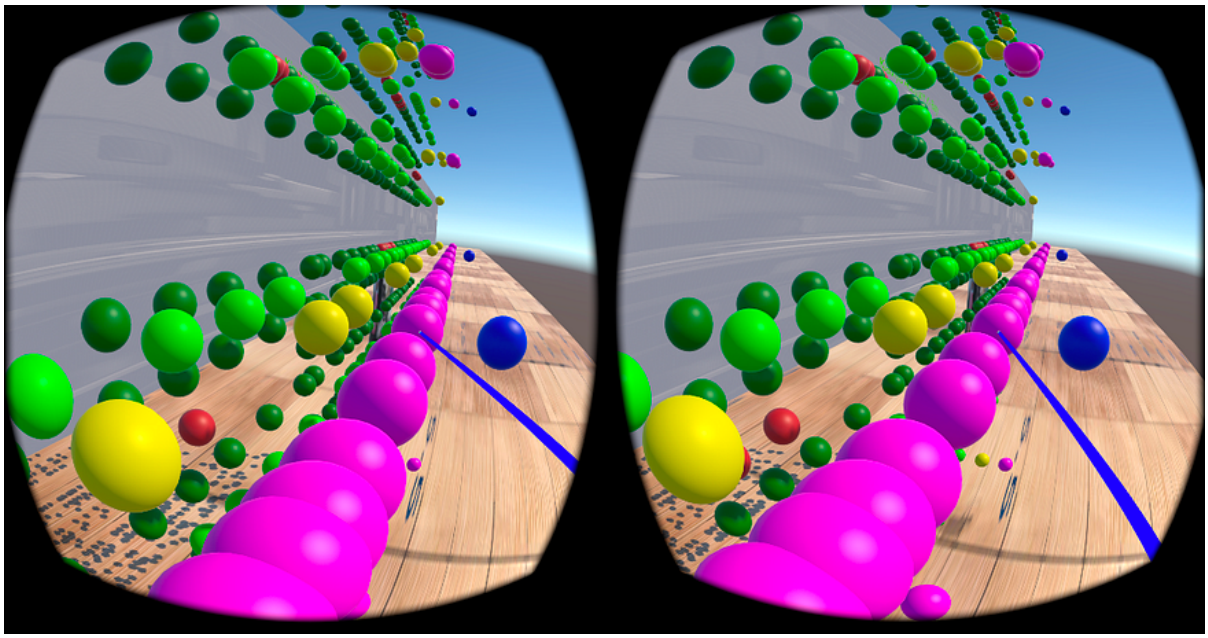


And finallyâ€¦two shots from my demo

Wellâ€¦it looks better in 3D!



Looking back toward older events, about to use my laser pointer to click on the blue sphere (placed order), which will then show what products the person ordered



Exploring a person who looked at a bunch of products (purple) after clicking on campaign emails (yellow)