“If I'm just starting out with learning augmented, virtual, or mixed reality development, how do I decide which headset to build for?”

This is one of the most frequently asked questions that I get around developing for immersive technologies, and with good reason - the space around today's AR, VR, and mixed reality devices is growing rapidly, with new platforms and devices released regularly as we start to see new players entering the arena. New developers or programmers who are just beginning to explore 3D development often see the number of headsets available to them, and wonder: how do I pick one? What if I choose "wrong", since we don't really know where the market will fall?

The answer will hopefully reassure you - truthfully, the biggest hurdle to starting down the path of immersive development is in understanding the new paradigms in designing and developing for 3D environments in general. When you are just starting out with development, you will be spending much more of your time learning the basics of graphics programming than a particular platform or SDK - after all, the first thing that you'll need to consider is **what** you're building in the truest sense of an application.

One of the first things that I built when I was learning Unity was an experience called Allegro, a simple application for the Google Cardboard that combined my love of music with lighting and a virtual environment. The mechanics of Allegro were simple: put on the headset, look at the differently colored notes floating around your head, and pull the magnet on the side of your headset while gazing at a particular one to play a short musical interlude. Stack the different notes to create different sound combinations, and enjoy the peacefulness of a quiet, whimsical experience.

I decided recently to put Allegro, and my dev skills, to a challenge: converting my Google Cardboard application to run on the Windows Universal platform, targeting Windows Holographic and the HoloLens device. This wasn't my first time building for the HoloLens, but it was the first time that I was doing a free-form exercise.

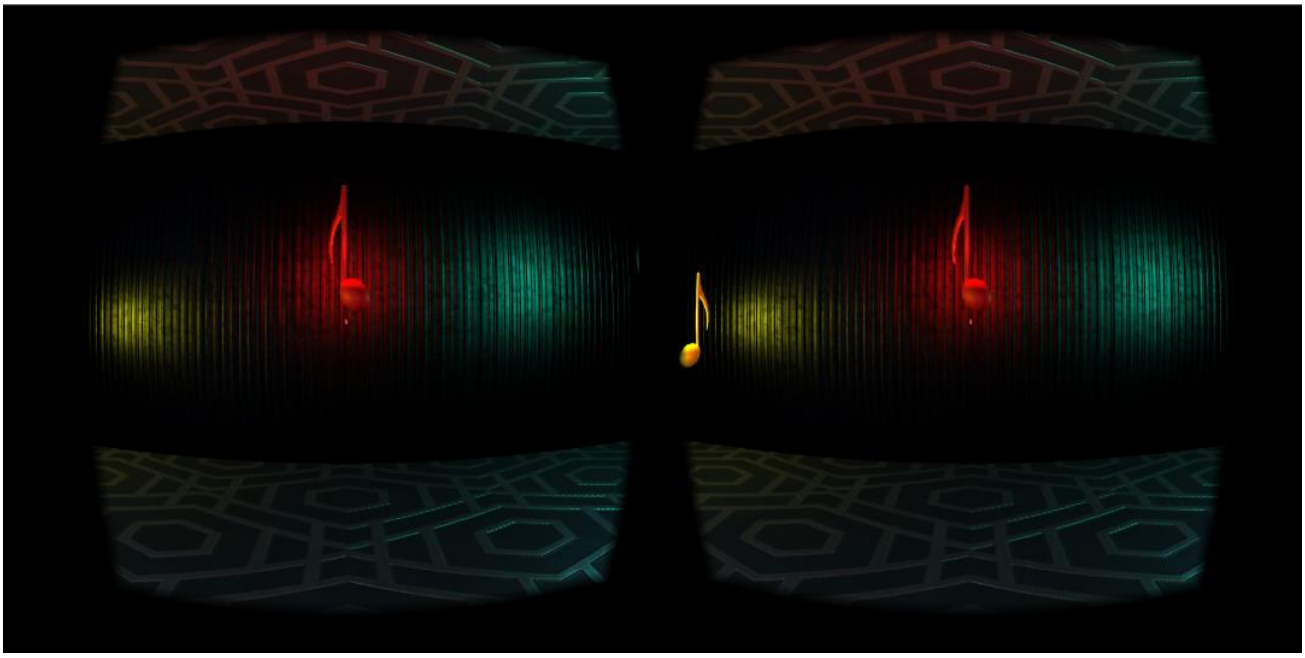
Setting Up

The first thing that I did was prepare my development machine for HoloLens development. If you don't have a developer device, the emulator tools for the Hololens do a great job of giving you the required resources to build and test holographic apps, and they're available on the development portal. I did a write up of my development environment for HoloLens a few weeks ago, so I won't dive into the details of setting up the prerequisites here, but you'll want to be prepped and ready to go before you get started. I also went through all of the tutorials on the Holographic Academy site, which give an excellent overview of using Unity to build a holographic application.

The Process

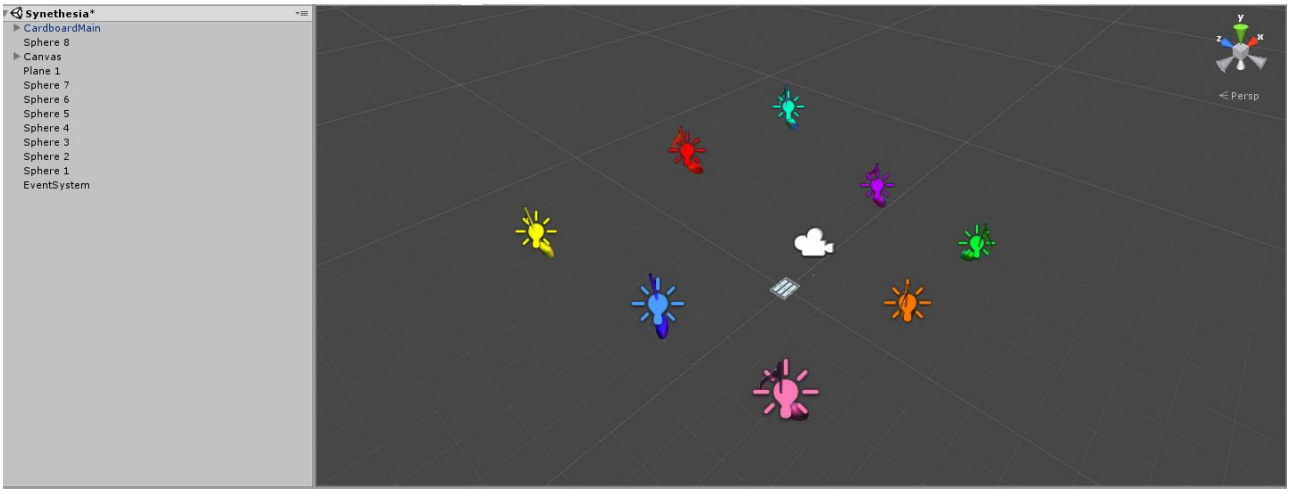
I had last worked on Allegro VR last June - over a year ago - and I wasn't entirely sure what to expect when I downloaded the code onto my computer and started up Unity.

After converting the project to the 5.4 Unity HoloLens beta, I was pleasantly surprised to find that everything upgraded cleanly and I was able to start the process of changing it over to Windows Holographic!



Before I got started, I made a new branch in my AllegroVR project for HoloLens support. Then, it was back into Unity for some modifications! Allegro currently has one fully functional scene (Synethesia) so I decided to start there.

The first change that I made to the scene was to take out the walls and the floors in my environment. Since the HoloLens is a *mixed reality* device, there wasn't really too much of a reason to keep a fake environment around the music notes when I could have them interact with the physical world instead. To do this, I selected the walls and floor in the Hierarchy and hit the 'delete' key.



When we delete our environment, our Unity Scene may look a little bare, but that's okay - we're going to be spicing it up with our physical world soon enough! To do this, I brought in the help of a great open source library from Microsoft, the HoloToolkit, which provides a lot of really powerful resources for creating holographic applications in Unity. Download the HoloToolkit from Github at: https://github.com/Microsoft/HoloToolkit-Unity. Once the package is downloaded and built, it can be integrated into a Unity project just like any other asset package.

Note: If you're following along with this in your own project, you may see a few console errors in Unity about unsafe code. Check out this forum post for some solutions - I didn't need the sharing capabilities, so I simply removed that folder to get rid of the issues

The next step was to change the camera prefab that we were using. The Cardboard camera (which is now all-up the Google VR/GVR) provides a different set of parameters than you would use for other devices, and changing this will be one of the big differences in our project. It's worth noting that for this particular project, the only custom script that I added to my CardboardMain camera was a physics raycaster for detecting 3D object collisions - if you're converting a larger project, **make a note of any custom scripts that you've put on your camera!**

Once I deleted the CardboardMain camera from my scene, I went into HoloToolkit > Utilities > Prefabs and dragged the Main Camera object into the root of my hierarchy. This prefab sets up the camera for us nicely, putting it at 0,0,0 (the player's head is the origin) and has all of the components set up to use the proper clear flags and culling. My music notes were floating a bit too high up in the air at this point, so I went in and moved them to be eye level with the camera.

Replacing the Gaze & Gesture Components

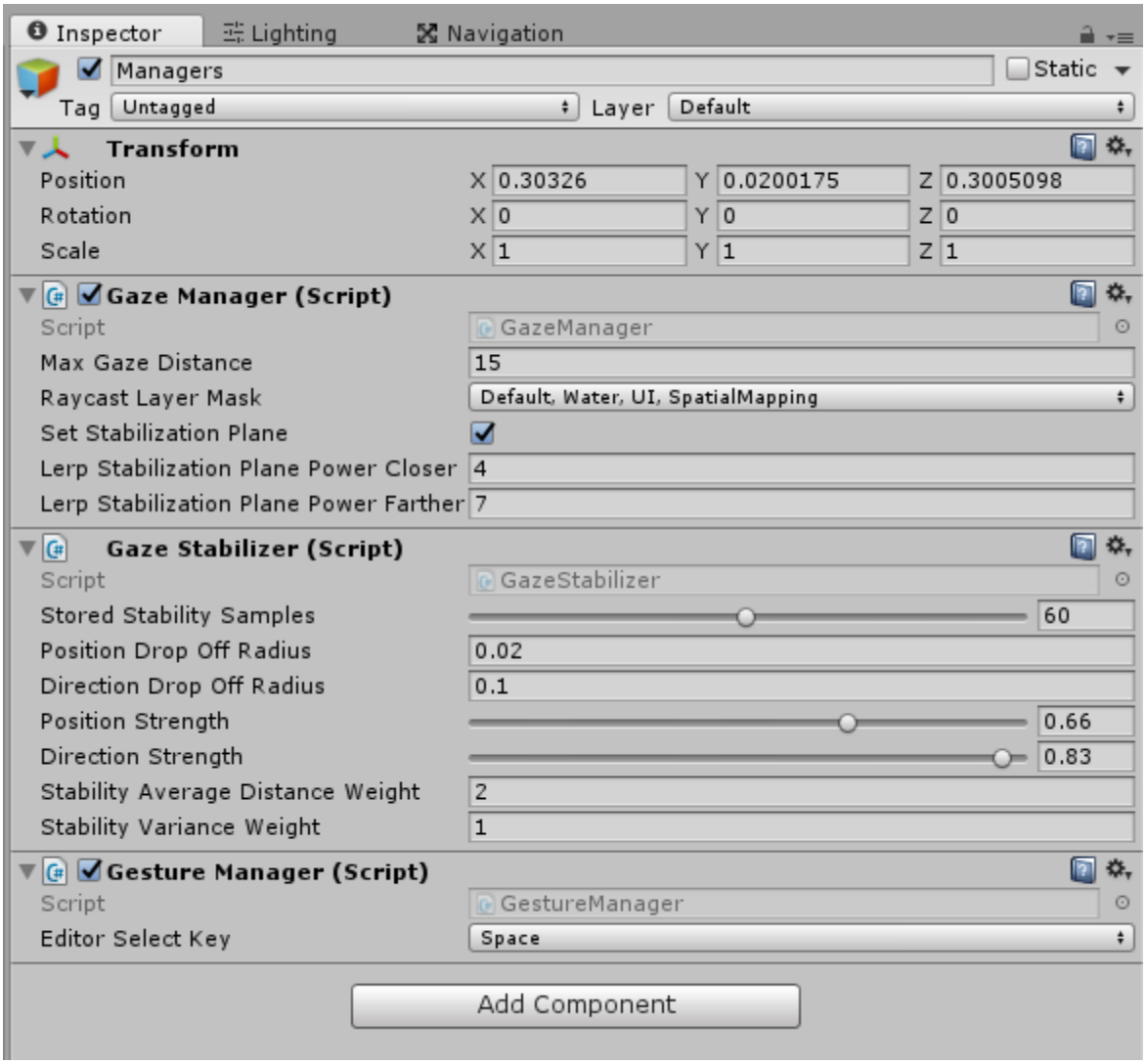
The next step is to replace the Cardboard GazeInput mechanism with the input system that the HoloLens uses. With Cardboard, Allegro triggered musical experiences when

the magnet trigger was pulled, but with HoloLens, I changed this to use a combination of Gaze and Air Tap (Holographic Academy 211 has a great overview on implementation of Gesture controls in Unity for HoloLens). To start out, I added an empty game object into my scene, which I named 'Managers', to house the scene-wide scripts for managing gaze and gesture.

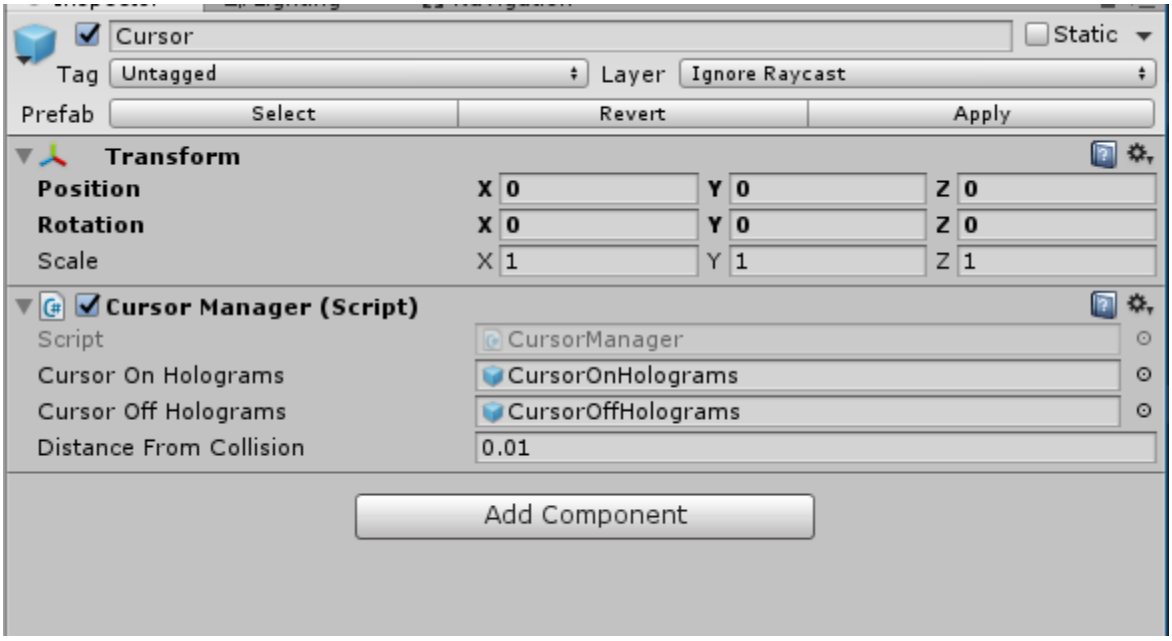
When developing for HoloLens, the basic structure to check for input is something like the following and uses Gaze + Gesture together:

- Where am I looking (gaze) in the world? This is handled by the Gaze Manager, which will return information about where a player is looking
- Am I looking at something that can be interacted with? If so, the Gaze Manager will send a message to the object.
    - What should I do when I look at something that can be interacted with? When the object receives a message from the Gaze Manager, the object should respond
- Is the user trying to do a gesture? This is handled by the Gesture Manager, which will send messages when a player is performing an air tap or tap+hold
    - If so, what happens? The object in focus (determined by the Gaze Manager) will respond to messages from the Gesture Manager

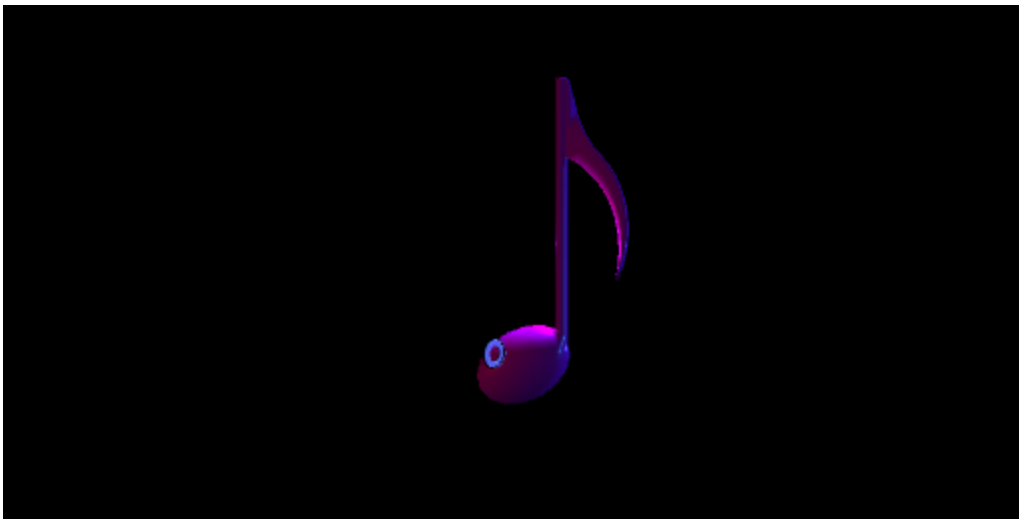
The HoloToolkit Unity package contains a few scripts to get us started with implementation of gaze and gesture in our applications, so I went ahead and attached GazeManager, GestureManager, and GazeStabilizer scripts to my Managers object. These can all be found in HoloToolkit > Input > Scripts.



I wanted to be able to see where I was looking in the scene, so I decided to use the included Cursor object to show a difference when the gaze manager detected whether or not an active hologram was selected. To do this, I dragged the Cursor prefab into my scene. The HoloToolkit cursor has both an On and Off Hologram child, which are triggered accordingly when they intersect with a hologram. Confirm that the cursor script has the On/Off Hologram children set properly - your Cursor prefab should have a script called CursorManager attached:



When we run our app, we should see a cursor appear when the camera is looking at one of our music notes:



Next, we want to develop the code that will handle the interactions from our Gaze and Gesture manager. Notice that there wasn't anything specific that I had to do for the cursor to detect whether or not it was on a hologram: our Gaze Manager handled that for us! We'll now take that a step further and create a few functions for our music notes to respond to the gaze and air tap.

Since this was already written for Cardboard, my music notes already had a script that defined their behavior when they were selected. I had a function called "SetGazedAt" that would be called when the object was clicked on - a bit of a naming misnomer on my

part, but the function itself simply recognized that when a click happened, we'd play a sound.

```
public void SetGazedAt(bool gazedAt) {
        if (gazedAt) {
                Debug.Log ("Entered Gaze on " + this.gameObject.tag);
                this.GetComponent<AudioSource> ().Play ();

        }

        //GetComponent<Renderer>().material.color = gazedAt ? Color.green : Color.red;
}
```

The original SetGazedAt script for Cardboard

With the HoloToolkit, our Gesture Manager handles the detection and collision for our holograms, so when it detects that we've performed the air tap gesture on an object, we call SendMessage on the selected hologram and pass it in an "OnSelect" method. To easily support this in the script on the music note, I simply added a new, public function that calls SetGazedAt(true) when the OnSelect method is called from the Gesture Manager.

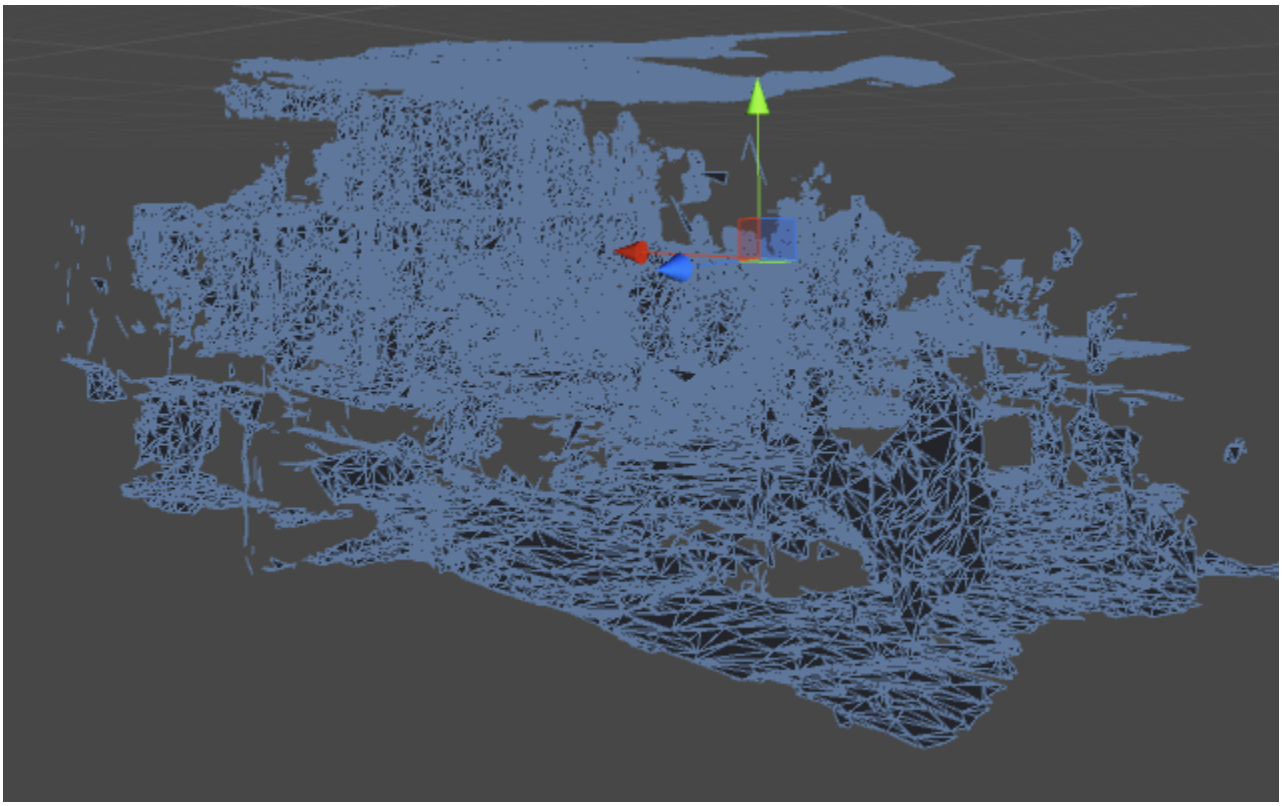
```
public void SetGazedAt(bool gazedAt) {
    if (gazedAt) {
        Debug.Log ("Entered Gaze on " + this.gameObject.tag);
        GetComponent<AudioSource> ().Play ();
        GetComponent<ParticleSystem>().Play();
    }

    //GetComponent<Renderer>().material.color = gazedAt ? Color.green : Color.red;
}

public void OnSelect()
{
    SetGazedAt(true);
}
```

The HoloLens version of SetGazedAt, with the accompanying OnSelect function that calls the behavior function

Being able to wrap my existing functionality in a public OnSelect method made the entire process incredibly straightforward. The code that controlled everything was reusable - instead of using an EventSystem with the GazeInput from the Cardboard SDK, we were able to use our Gaze Manager and Gesture Manager to take care of interaction detection. At this point, the entire project conversion up until now had taken about an hour to do, and resulted in a fully functional, spatial-sound enabled mixed reality musical sensation!

But let's not stop there! If you were paying attention, you may have noticed that our SetGazedAt function contains one extra line for HoloLens - a command to play a non-existent particle system from our music notes! Because of how quickly I was able to get this up and running on the HoloLens, I decided to spend a few extra hours beefing up the app to take advantage of one of the coolest features of the HoloLens: the spatial mapping feature!

Using Spatial Mapping



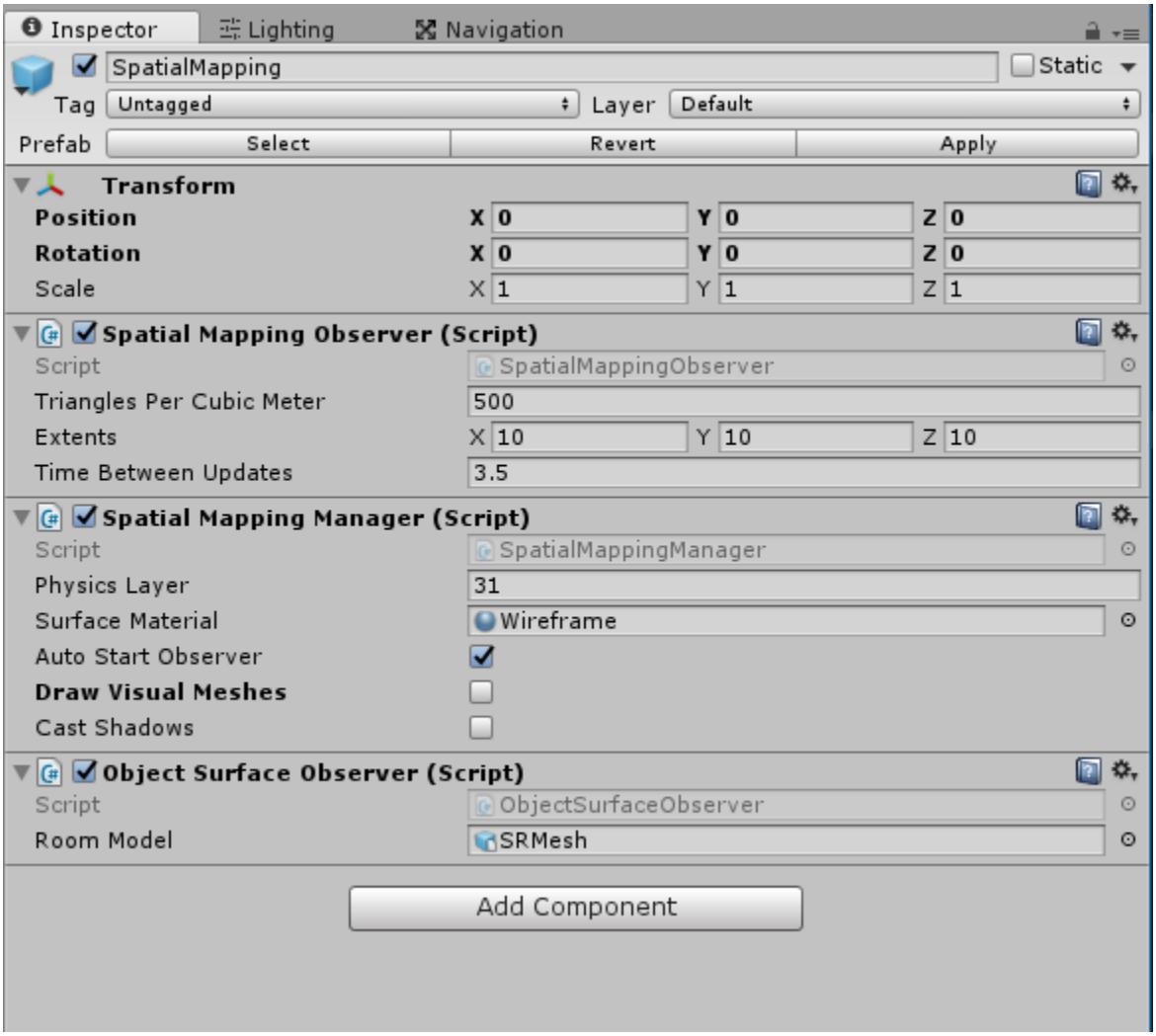
Part of a spatial map of a room, displayed in Unity

I decided that I wanted to add a feature to my app where the music notes would send out particles when they were played. I wanted the particles to bounce off of the physical environment I was in, so I decided to dive into the spatial mapping functionality that the HoloLens exposes to developers and add it into my application. If you need a primer on HoloLens spatial mapping, Holograms 230 has got you covered. There are some really cool ways to start poking around the spatial mapping capabilities, so before we dive in, a few helpful tips:

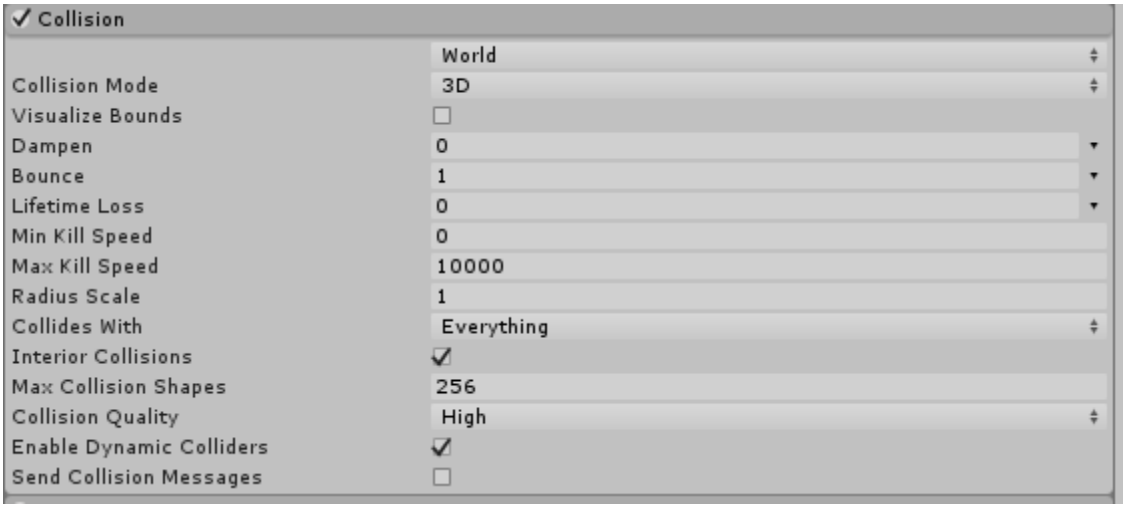
- Even if you don't have a device, the HoloLens emulator comes with some predefined 'room' objects that you can use to simulate different spatial environments in your application. These meshes can help you test how your application might respond to physical objects, right from the emulated device.

- If you have access to a HoloLens, you can capture the data from the room you're in with your HoloLens and download it as an object that can be used in Unity. You can use this instead of a room model while developing, if you want to more closely mirror your current environment within your editor.

To implement the spatial mapping feature, I hopped into the HoloLens portal and downloaded my room mesh, pictured above. I imported it into my Unity project, then added the SpatialMapping prefab object into my hierarchy. The spatial mapping prefab has three scripts attached to it, and I set my room model to the imported room (SR Mesh):



I needed to do a few more things for my particles to bounce, now that I had the spatial mapping component added into my application. I gave each of the music notes a particle system with physics enabled, making sure to specify that I wanted my particles to collide with objects and bounce off of them:



I also had to make sure that I was setting the proper permissions in my build. Currently, with Unity, instead of building a direct binary for the HoloLens, you develop a C# project and finish the build process in Visual Studio. Windows 10 projects, such as the ones built for HoloLens, have special permissions similar to mobile apps where your application needs to specify the features they're using, so we're going to need to make sure that we set ours to know that we're accessing the spatial mapping data from the device. To set this up, it's just a matter of going into Player Settings, then at the bottom of the Inspector window in Publisher Settings, checking 'Spatial Perception'.

With that, it was time to build and deploy! You can check out the final project on YouTube: https://www.youtube.com/embed/GSLYbGovVUU

Of course, things are constantly evolving with both the Cardboard / Daydream (GVR) SDK as well as the HoloToolkit, and each project is different, so this isn't going to be a one-size-fits-all solution, but I learned a lot about the capabilities and development tools for HoloLens by working on this project! If you're interested in trying them out yourself, the full source code for each project is up on GitHub:

Allegro VR (Cardboard - older SDK, may or may not be compatible with more recent builds of Unity)

HoloLens Allegro - Built with Unity 5.4 HTP, may have some compilation fixes needed for more recent updates