david ayman shamma

VR Research Collection in Mozilla Hubs

david ayman shamma Jul 21, 2020 · 7 min read

Group photo from the ACM CHI2020 SocialVR Workshop

Before there was a global pandemic, we were planning on running a workshop on SocialVR at ACM CHI2020 and were underway to have a half in real life and half virtual meeting. Of course, the pandemic took everything 100% virtual. As organizers, we quickly pivoted to run the whole SocialVR workshop in VR (which makes good sense).

As researchers, we wondered how people would flock, cluster, break out, and use the VR tool (which was also a question of the workshop already). No turnkey system would let us run the workshop and collect data. However, there is the open-source Mozilla Hubs, which I wanted to turn into a research platform. I've never done any VR programming work in the past, so this is a quick guide to how Hubs kind of works, how you can instrument it, and what we can do next. Plus, I'll share my code along the way in case you want to set up your own Hubs Cloud instance and collect similar data with informed participant consent of course.

Hubs uses a few frameworks with node.js and react.js and web platform with A-Frame and Three.js for VR and 3D stuff. I knew node and react already. If you don't, go run some quick tutorials. What I wished I did first but didn't is go through the A-Frame tutorial and Three.js tutorial. Instead, I just started reading the codebase while hunting and pecking the DOM in the Javascript console. Don't do what I did, take a moment, and go through the two short tutorials, and you'll save yourself a lot of time.

Setting up Hubs Cloud

You can set up your own Mozilla Hubs Cloud instance in AWS if you want to modify and push your client. This is what I did but AWS is not fun if you don't know it. Thanks to the support from Mozilla community on Discord and my colleague Yulius Tjahjadi (who helped architect this research collection with me), I managed to get it installed. You can also use the Digital Ocean instance which is still experimental (I think). You'll need to setup your custom client too and add asset packs. I'd recommend making sure this all works before going on to modding the code. Once your Hubs Cloud is standing up and working, then continue here. There's a lot of documentation on this AWS step, read it twice then try it. Digital Ocean might be easier and cheaper but I set this up before there was a droplet.

Collecting Data

And while Hubs is a very nice system but it does tend to max out around 25 participants, which is our anticipated load. We want to track all opt-in users of them across all of our rooms on the server. While we could write some special code to constantly query the Janus server to see where everyone is or even modify the Janus server. In reality, I had a few weeks to do this and didn't want to overload the server.

Instead, I decided to have every client POST log data to a different server. This adds a small overhead on the browser and nothing more. What I'll share here is V2 which was a much clearer method than my first approach; I wrote V2…ahem…after I went through the A-Frame tutorial.

Here's what we are looking to collect for every user in every room at every tick:

1. Have they entered in the room or are in the lobby?
2. Where are they in the room (x, y, z)?
3. Where are they looking/facing in the room?
4. Are they muted?
5. Are they talking (and if so, how loud on their input mic)?
6. Do they have the spacial audio dampened?

We could, of course, collect other things like are they using the laser pointer, are they drawing something, are they typing, even what they put into chat? But really, we just wanted positional data and audio flags to see how they cluster. It's important here to note we are not capturing "what" they are saying. We only capture the input level of their microphone or if they are in a muted state. I get that could be their dog barking, but it's gonna be the signal we pull. This is similar to a study I did about people watching videos together live over chat where, instead of wiretapping, we only collected number of words and number of characters people typed. It's one thing to say one will do no evil with the data they collect, it's another to just not collect it in the first place.

Making a Custom Logger in a Hubs Cloud Client

There are two major components to our research logger, the Hubs client system and the server which logs the data. For the Hubs client, we will make a new system component in A-Frame to log data and POST it. For the Hubs Client, we will make a new A-Frame component in the `src/systems` directory and since I'm making a few of these little things, I went ahead and made a `research` directory under there. We'll call the file `research-logger.js`. Simply, we register a system in A-Frame.

```
AFRAME.registerSystem('research-logger', {
    init: function() {
        ...
    },

    tick() {
        ...
    }
});
```

What this does is let us set up an `init()` function as a constructor and an `tick()` method to call on every tick. It's actually pretty simple from here on out. On every tick we want to get the avatar's position and audio state from the DOM and log it. We can easily look at `AFRAME.scenes[0].states` to get muted and room state and get position from `document.getElementById('avatar-rig').avatarRig.object3D`. Easy right? Pretty much everything you need is *somewhere* in the DOM. My README.md file documents every column I added to the log.

Not so fast, calling HTTP POST request every tick isn't such a good idea. That's a lot of traffic. Instead, we'll store our tick data line by line as a CSV queued to upload. Next, we'll batch a lot of ticks and do a POST of the batch to keep from flooding sockets. After testing various sizes, we settled on 4000 ticks per post as a good size which didn't kill the client or the server.

I added a few other things such as a persistent UUID (which you won't need if you make people register on your Hubs Cloud server because then we will just log their numeric account ID), the `axios` library for making HTTP Requests, and the `moment` library for getting everyone on a NTP UTC stamp. Plus, I added a query parameter to the URL to toggle logging as needed. I also added one last little thing. Many times, instead of getting a 0.0 value, Javascript will compute something as `1.3600232051658077e-15`. I just rounded these to `0.0` if the float was smaller than `10^-12` to make the upload payload smaller. The full logger is on my hubs-cloud fork on GitHub. One last little thing to mention, when you make a new A-Frame system, you have to include it so it loads in. To do this, drop it in somewhere with the other imports in the `src/hubs.js` file.

Collecting Data on the Server

You'll notice my logger just makes a simple POST to a server:

```
researchCollect(data, url = "https://*.*/log") { ... }
```

You will need a server machine somewhere to take that request and write it to disk. It will do this for every connected, logging client. This is pretty straightforward and can be written in whatever you like; you can use PHP or Python script with Flask and uWSGI. You'll want to do a sanity check to make sure the data is clean (sanitize your inputs!) and make sure the CORS request is properly set up on your webserver to only allow connections from your Hubs Cloud instance. Likewise, back on your Hubs Cloud Admin panel, you'll need to add your data collection server to the `Content-Security-Policy`. You should also have tens of gigabytes free to log a half day workshop of 20 people. If you're super fancy, write the log compressed to disk which saves a lot of space at some CPU cost.

Deploy and Collect

Now you're set! Run `npm deploy` to get it to your Hubs Cloud and log away. We ran our ACM CHI 2020 workshop on SocialVR with the logger and are looking now to watch how people congregated, flocked, and collaborated across 4 rooms (a main hall and 3 breakout rooms). Here's a look at the main hall at 10,000x speedup.

A detailed post is coming later on how we made this viz.

My collaborator Julie Williamson at the University of Glasgow used known anchor points on the map to define the visualization coordinate system. These anchor points were taken from Spoke, which uses the same coordinate system as the live Hubs environment. The points per person per tick were resampled to 5FPS using `pandas` then used to generate the animation here using `matplotlib`. This was stitched together using `ffmpeg` then sped up using Adobe Premiere.

Beyond HMDs, VR is moving into our browsers and will become a new platform for interaction. By studying how people collaborate in these systems we can begin to understand the social dynamics of complicated spaces and even start to build new tools for enhancing collaboration and building generative worlds for future of work scenarios (and fun too!).

tl;dr: Feel free to reuse the logger in my Hubs Research directory.

ps. Many thanks to the Mozilla Hubs team and the community on Discord who offered much support along the way!

Thanks to Julie R Williamson (hide).

Mozilla Mozilla Hubs VR Research Visualization