Implementing Multiplayer Virtual Reality For any Application in Three.js COMP 3490

Wynand Badenhorst

December 16, 2017

Due on:	December 18, 2017
Developed by:	Wynand Badenhorst
	(And everyone who made the libraries
	that this project depends on)
Professor:	Neil Bruce

1 Objective

Before starting this project, I was curious about the limits of mobile VR. I sought out to implement a very basic multiplayer VR environment as a proof of concept. This was achieved, and now you can watch an arcade machine with your friends, as you all move the claw up down and around in Virtual Reality.

2 Challenges in Implementation

a Implementing the stereoscopic view was very difficult at first, but after thoroughly studying the documentation of Three.js, I found a that you could add effect to the rendering of your scene, and one of those is 'Three.StereoEffect'. This works much more efficiently than making your own two cameras, and is much simpler to implement. The command is simply:

```
effect = new THREE.StereoEffect( renderer );
effect.setSize( window.innerWidth, window.innerHeight );
```

b Porting the game to mobile proved to be challenging as well, but socket.io and express.js proved very helpful in getting this done. I learned about this from an incomplete video tutorial (https://www.youtube.com/watch?v=LBj4XlySZLU).
Upon further research and trial and error, I succesfully managed to get the server and clients communicating.

c Sharing the world between clients, and keeping them synchronized, proved to be much easier than anticipated. My first idea was to stream video to the clients and make the application server heavy, but then I realized if only geometry were maintained on the server side, then the rendering sould be done by the clients.

How I achieved the shared updatable world was by creating the Geometries of the game in the server, but never adding them to a scene or rendering them. Whenever a client affects the world around it, it informs the server, the server updates it's data structures, if it chooses to, and then the server broadcasts the results to all the clients.

The clients then do their own rendering and their own (strictly local) geometry updates.

d In my first iteration, I was using more bandwidth than available, so messages to the server were being cached until I ran out of memory. What I realized was that the updates to the server were being sent once every time interval instead of once every time interval when there was an actual update. This was corrected by refactoring my code to only send updates when the user has interacted with the scene and manipulated it.

e I did not implement any physics or world-world interaction, because this would be continuous, or would need to be done almost entirely client-side with 'corrections' from the server on some interval.

3 Techniques used to implement Multiplayer VR

a Implementing VR:

Implementing the virtual reality component of the game proved simpler than expected. I started work on building my own camera object, but Three.js already has stereo vision implemented.

With a regular camera with orbit controls, we saw that it is implemented as follows:

```
cameraControls = new THREE.OrbitControls(camera, renderer.domElement);
```

While the render function does this:

```
renderer.render(scene, camera);
```

Stereo vision is implemented like this:

```
effect = new THREE.StereoEffect( renderer );
effect.setSize( window.innerWidth, window.innerHeight );
cameraControls = new THREE.DeviceOrientationControls( camera );
```

While the render function does this:

```
effect.render(scene, camera);
```

b Implementing Multiplayer:

Let's assume the client already has code, somehow.

The client's freedom should be restricted, for safety. For instance, a command might look like:

```
$.ajax({
        type: 'POST',
        url: '/update',
        data: JSON.stringify({
                name: 'hangingArm',
                direction: 'decr'
        }),
        contentType: 'application/json',
        cache: false
});
```

4 Results and Conclusions

5 Concerns with VR user experience

6 What's next