CS Capstone Technology Review

December 2, 2018

Creating Immersive Experiences on the Web using VR and AR

Prepared for

Intel

Alexis Menard

Prepared by

Group 47- WebPhysicsVR

Jonathan Jones

Abstract

This document is a technical review of the components needed for the project graphical API, physics API, and Audio API. Each section explains the need for each respective component and three possible technologies that could be used in its place. Reviewed are the physics APIs: Cannon.js, Ammo.js and Oimo.js; the graphical APIs: ThreeJS, BabylonJS and Blend4Web; the audio APIs: WebAudio, Howler JS and Waud.

Contents

1 INTRODUCTION

Building virtual reality applications is hard and so is building a physics simulation. Developing both at the same time is even more difficult. There are lots of moving parts to a project such as this. For the visual/audio experience to materialize, there needs to be graphical, physics engine, and audio components. Graphical APIs get images on the webpage, physics APIs create realistic image movement and audio APIs immerse the user with sound. Implementing these features on their own is difficult enough. Thankfully, web developers have been hard at work creating a multitude of libraries that provide these services. Choosing the right libraries for the job, however, can be difficult. These libraries need to work together and most importantly, with WebXR. Under the assumption that WebXR developers have been building upon their past API, WebVR, the ideal libraries will be those that have built in support for WebVR. There must also be sufficient documentation associated with each library.

2 PHYSICS API

Physics simulations are very math intensive and require a lot of calculations behind the scenes. When VR is factored in, the process becomes much more difficult. Interactive actions like grabbing, swinging and throwing objects extends beyond simple 3D physics. There are a variety of physics engines that have been developed with WebVR in mind. Each brings its own advantages and disadvantages to the table. The physics engine used for this project needs to have VR support built in, be well documented, efficient and extensive in its features.

2.1 Cannon.js

Cannon.js is a lightweight 3D web physics engine written entirely in Javascript [1]. The library is open source and has been developed under an MIT license. Cannon.js primarily serves as a rigid body simulation library that is able to make objects move and interact in a realistic way [1]. It has a multitude of features that can be used to create just about any kind of physics situation. These include rigid body dynamics, discrete collision detection, friction, restitution, object constraints, and much more. There are also various shapes and collision algorithms for those shapes included in the library [1].

Cannon.js provides just about every physics function that this project requires. It also has an enormous amount of documentation. Equally as ideal is the fact that Cannon.js is used by many of the technologies that are being considered including BabylonJS and A-Frame [2].

2.2 Ammo.js

Ammo.js is a direct port of the bullet physics engine into Javascript. Bullet was originally written in C++ and is almost identical to ammo. It is open source and has been developed under a zlib license [3]. The Bullet physics engine has been a long time favorite for C++ developers. Bullet provides most, if not more features than Cannon.js does and even includes VR support. Bullet has been around for a long time and has a vibrant community, support and documentation. The same cannot be said for ammo which seems to rely on documentation for legacy Bullet. It also is not clear whether or not Ammo.js has VR supportive functions or not.

Included with Ammo.js is a compiler that is able to take Bullet code and convert it to Ammo.js friendly code. This means that it is possible to take the latest Bullet libraries and convert them into Javascript.

2.3 Oimo.js

Oimo.js, like Cannon.js, is a lightweight 3D physics engine for Javascript [4]. Oimo.js is a port of the OimoPhysics physics engine which was written in C++. It is open source and has been developed under an MIT license and like Ammo, it has poor documentation. Its features include rigid body motion, fast collision and bounding volume hierarchy, contacts with friction and restitution, multiple collision geometries, joints with springs, limits and motors, breakable joints and constraint solvers [4]. Oimo.js is a powerful yet simple 3D graphics library. The WebGL graphical framework, BabylonJS, uses Oimo.js in conjunction with Cannon.js under the hood for its integrated physics. This makes up for the lack of documentation on the Oimo.js website as it is already provided in the Babylon.js documentation.

3 WebGL Frameworks - Graphics

For most graphics programmers, OpenGL is the tool of choice. It works universally across most systems and is capable of just about anything that can be thought of. OpenGL though, is not embedded in web browsers, without the use of plugins. For that, there is WebGL; a cross-platform, royalty-free API used to create 3D graphics in a Web browser [5]. WebGL by itself can do amazing things. These things are hard to program though and have already been done by others on the web. VR, especially, is difficult to implement but these open source libraries have taken care of the boilerplate already. The graphical APIs explored in this document all use WebGL as their backend.

3.1 ThreeJS

Three.js is a lightweight cross-browser JavaScript library/API used to create and display animated 3D computer graphics on a Web browser. Three.js scripts may be used in conjunction with the HTML5 canvas element, SVG or WebGL [6]. The library provides Canvas 2D, SVG, CSS3D and WebGL renderers [7]. Three.js has been used to make hundreds of graphical simulations on the web and a large number of them include the WebVR API. Three.js is feature heavy and includes animation, shaders, effects, objects, debugging, virtual reality and more. It is compatible on all modern browsers and can even run on older browsers as it is able to fall back to one of its multiple renderers if one is not available on the current browser [8].

Three.js has an incredible amount of documentation and a vibrant community that is constantly making improvements to the API. It has the most documentation out of the three graphical APIs that are discussed here. There are hundreds of examples and demos on the Three.js website that can be used for reference when developing this project.

3.2 BabylonJS

BabylonJS is a graphical web framework for WebGL that was developed by Microsoft. It has a strong focus on the development of 3D games. It is about as feature heavy as Three.js and includes some features not found in the other. BabylonJS is relatively new and is much older than Three.js. It has an integrated physics engine that uses cannon.js and oimo.js as well as an integrated spatial audio system that uses WebAudio. Essentially, this framework is able to take care of most of the features that need to be implemented by itself. BabylonJS is highly optimized and looks to perform well on most systems [9].

Like ThreeJS, Babylon has an amazing community of developers working on it who are always actively improving the framework. It is extensively documented and also boasts many demos and tutorials.

3.3 Blend4Web

Blend4Web is an open source, free GPLv3 licensed WebGL graphical framework that has a built in material editor with heavy support for Blender [10]. This is the most complete framework out of this list; it includes everything that Three.js and BabylonJS have and more. What is most ideal about this framework is how it was created with Blender in mind. Asset development is hard, Blender makes it easy, and Blend4Web makes implementation of Blender products even easier. Like Babylon, Blend4Web has an integrated physics engine, spatial audio systems and support for VR animation [10]. It has been described as a tool for interactive 3D visualization on the Internet [10]. There are lots of examples of Blend4Web projects and many pages of documentation on the web. Still, it remains lesser known than the previously listed frameworks which unfortunately can play into its usefulness as there is still a much better community tied to the other two.

4 Audio API

The goal of VR is immersion and audio is an incredibly important part of this process. For this project to be immersive, it needs event driven spatial audio and an audio API that can provide it. Spatial audio is a full-sphere surround sound technique that utilizes the three dimensions to mimic real life audio [11]. This is the only way to correctly convey sound to the user in a virtual reality simulation. Much like the other components in this review, this part of the project does not need to be created from scratch as other developers have done that job already.

4.1 WebAudio

The WebAudio API provides a powerful and versatile system for controlling audio on the Web that gives developers the ability to choose audio sources, add effects to audio, create audio visualizations, apply spatial effects and much more [12]. WebAudio works by using a series of audio nodes in an audio routing graph to bring organized audio framework to a webpage [12]. It is just about the only audio API that is used world wide on the internet. Part of the reason for this is the fact that it was developed by the W3C (World Wide Web Consortium). As far as VR development goes, Mozilla has said that It's really useful for WebXR and gaming. In 3D spaces, it's the only way to achieve realistic audio. [13] There is an enormous amount of documentation associated with WebAudio, mostly from the Mozilla developer site.

By itself, web audio is already easy to use and implement into web applications. Still, there are many libraries that simplify its use even further and add features themselves. The following APIs wrap around basic WebAudio and abstract it.

4.2 HowlerJS

HowlerJS makes working with audio and WebAudio in Javascript easy. It is a single API for all audio needs that defaults to WebAudio and falls back to HTML5 if needed [14]. It is well optimized, easy to use, modular and lightweight. Additionally, it is compatible with most modern browsers because of its fallback to HTML5 [14]. HowlerJS has stereo panning and 3D spatial audio support which is very important to this project. Finally, it has full codec support, meaning that all audio file types are compatible [14]. As a WebAudio API, it is one of the most popular in its field and has developed a large community. As a result, this API has extensive documentation.

4.3 Waud

Waud is essentially the same as HowlerJS. It has all the same features and support for spatial audio. Like Howler, it abstracts WebAudio, defaults to it, and falls back on HTML5 on unsupported browsers [15]. Both have zero dependencies and both are fully modular. One key difference however is the absence of volume fading in Waud. It is important to keep as many options open as possible when dealing with audio so this is a clear disadvantage to HowlerJS.

5 Conclusion

For this project to be successful, the right technologies must be chosen for the right components. For the graphical, physical engine and auditory components there are many options. Each with its own advantages and disadvantages. The sheer amount of Javascript libraries that area available for 3D visualizations is staggering and can make this choice unclear. For this project, however, there are clear choices for each component.

The physics engine should be Cannon.js. While Oimo.js and Ammo.js are both great physics engines, only one Cannon.js built directly into the A-Frame API. As it seems that this project will be making use of A-Frame, it only makes sense to use what works with it. This along with the extensive documentation that Cannon.js has makes this an easy choice.

The graphics API must be Three.js. This API stands out as a definitive choice among the three APIs listed. Like Cannon.js, Three.js is built into A-Frame which makes it highly desirable. Its incredibly feature richness and vibrant community is another factor in this recommendation. It is important that there is lots of support behind each component. Success is made all too easy when the entire world of web developers is at the back of the API being used by a project.

For the audio API it was another easy choice. HowlerJS is the favorite among web developers for 3D audio and spatialization. Its wide compatibility and abstraction of WebAudio make it simple and easy to use for beginners in the audio arena. Like the others, HowlerJS has the most extensive documentation among its competitors.

Each technology has its advantages and disadvantages but these technologies will only serve to improve the development experience of this project.

REFERENCES

[1] "Cannon.js readme," https://github.com/schteppe/cannon.js/blob/master/README.markdown, accessed: 2018-11-4.

[2] "Cannon a-frame article," https://hacks.mozilla.org/2017/05/having-fun-with-physics-and-a-frame/, accessed: 2018-11-4.

[3] "Ammo.js readme," https://github.com/kripken/ammo.js/blob/master/README.markdown, accessed: 2018-11-4.

[4] "Oimo.js github," https://github.com/lo-th/Oimo.js, accessed: 2018-11-4.

[5] "Webgl wiki," https://www.khronos.org/webgl/wiki/Getting_Started, accessed: 2018-11-4.

[6] "Three.js stack overflow," https://stackoverflow.com/questions/tagged/three.js, accessed: 2018-11-4.

[7] "Three.js readme," https://github.com/mrdoob/three.js/blob/dev/README.md#threejs, accessed: 2018-11-4.

[8] "Three.js browser support," https://threejs.org/docs/#manual/en/introduction/Browser-support, accessed: 2018-11-4.

[9] "Babylonjs site," https://www.babylonjs.com/, accessed: 2018-11-4.

[10] "Blend4web readme," https://github.com/TriumphLLC/Blend4Web/blob/master/README.rst, accessed: 2018-11-4.

[11] "Spatial audio how to," https://blog.storyhunter.com/how-to-use-spatial-audio-to-create-immersive-360-videos-e0805b51c143, accessed: 2018-11-4.

[12] "Mozilla web audio introduction," https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API, accessed: 2018-11-4.

[13] "Mozilla web audio spatialization," https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API/Web_audio_spatialization_basics, accessed: 2018-11-4.

[14] "Howlerjs readme," https://github.com/goldfire/howler.js/blob/master/README.md, accessed: 2018-11-4.

[15] "Waud github," https://github.com/waud/waud, accessed: 2018-11-4.