Oregon State University

CS 461: Senior Software Engineering Project I

Group 47

Immersive Web VR/AR:

Tech Review

Tim Forsyth

Abstract

This document is a review of the potential technologies that could be used in the development of the project for Capstone Group 47, WebPhysicsVR. Inside is a review of potential frameworks, tools, libraries and APIs that could be used in generating the user interfaces, as well as implementing functionality to said interfaces and user controller support. A few of the technologies that will be reviewed are A-Frame, ReactVR, Three.js, Gamepad API, Reticulum, and JavaScript.

Fall 2018

1 Introduction

WebPhysicsVR, developed by Group 47, will be an interactive physics simulation which utilizes the WebXR API at its core. The goal of this project is to provide a tech demo of how WebXR can be used to develop entertaining and interactive VR experiences within a web browser and act as an example for future developers to reference in their own endeavors. Tim Forsyth will be responsible for the creation and functionality of the various interfacing within WebPhysicsVR. This can be broken up into three pieces: user interface visualization, user interface functionality, and controller interfacing. This document will go over the various technologies available that may be useful for each individual feature.

2 UI Visualization

The user interface is an essential aspect of most interactive programs, WebPhysicsVR is no different. Several different menus will likely be needed, such as a menu for picking the room or selecting objects from a tool box. Objects themselves will also need some sort of interface that displays various statistics about the object's motion, such as velocity, acceleration and height.

In most cases, the UI is on the forefront of the screen, in direct view of the user. However, this doesn't work in VR. Placing the UI too close to the user can lower the clarity and make it nearly impossible to read as well as strain the user. The solution is to use spatial UI. Spatial UI is a form of UI that exists within the virtual space. This essentially means that the UI will not be on the forefront of the screen, instead it will live within the VR environment. Doing this makes it much easier for the user to read and overall interact with the interface. The first step in achieving this is by being able to generate the UI model. There are several frameworks available that can aid in this process; this review will look at three potential tools: A-Frame, ReactVR, and Three.js.

2.1 A-Frame: Primitives

A-Frame is a high level web framework originally developed by Mozilla that has turned into an open source project dedicated to helping developers create virtual reality experiences. Being based on top of HTML, it makes it a framework that is easy to jump straight into. A-Frame provides several primitives that make modeling in a VR environment easy. Primitives are a convenience layer on top of the core A-Frame API that provides a more familiar interface with HTML attributes maping to a single value.[1] Of the several primitives available, the most useful would likely be `<a-box>`, `<a-plane>` and `<a-text>`. These respectively create box like shapes, create flat surfaces, and wrap text onto an object.[2, 3, 4]

2.2 ReactVR

Similar to A-Frame, ReactVR is a high level framework that has been created for the use of developing user interfaces in a VR environment. ReactVR is essentially React, a JavaScript library for building user interfaces, except it operates within a VR environment. This means it uses the same syntax as React, making it second nature to use for experienced React developers. The caveat is that ReactVR is still young and is not as fully fleshed out as a tool such as A-Frame. With that being said, it still does provide useful tools for creating UI, such as OVRUI which is a library that contains several geometries used in building UIs in VR.[5]

2.3 Three.js

Both A-Frame and ReactVR share at least one thing in common: the underlying library they work on top of, Three.js. Three.js is a project that aims at creating a lightweight, easy to use, 3D library. It sits on top of WebGL and simplifies the rather complex API.[6] Even though there exist other tools such as A-Frame and ReactVR that simplify it even further, it can still be useful to directly use Three.js. Using a high level framework makes it easier to work with, but it can also be more restrictive. Three.js has several geometry classes used for creating 3D models; in the case of UI, PlaneGeometry would the most useful for generating a flat surface to work with and build upon.[7]

3 UI Functionality

A menu needs more than just a user interface model, it needs to actually have some sort of interactive functionality. There are several ways to be able to detect and handle user inputs, such as button presses or clicks, and perform the requested actions. The few tools that will be discussed in this review are JavaScript, Reticulum, and A-Frame.

3.1 JavaScript

Many tools that have already been discussed use JavaScript underneath their overarching framework. In fact, most modern web browsers and APIs use JavaScript, including the WebXR API that will be used. Therefore, JavaScript will be used in some manner, whether it is through some sort of framework or not. With that being said, there is reason to use JavaScript directly for event handling. Being such a popular language, most people know JavaScript and have experience with it to some extent. Luckily, event handling with JavaScript event listeners in WebXR is exactly the same as with any other web API. This means previous knowledge and experience of JavaScript will be able to be used in helping process user clicks and perform actions.

3.2 Reticulum

Reticulum is a simple gaze interaction manager that is powered by Three.js. It boasts gaze and click events for targeted objects, allowing us to have the option of implementing a gaze based UI in which users could simply look at what they want to click instead of aiming with a controller. It also has built in fuse support, which adds a timer or a countdown that lets the user know a click event is going to occur from staring at the object. This helps WebPhysicsVR become easier to use for those who are using VR head mounted displays that do not support controllers, such as Google Cardboard. Reticulum is also compatible with all of the other technologies that have been looked at so far.[8]

3.3 A-Frame: Components

A-Frame has the concept of components, which are reusable chunks of data that can be simply plugged into an entity or object to add functionality by way of JavaScript. While A-Frame itself my not be implementing the event listening, per say, it can make it a lot easier to apply JavaScript event listeners functionality to entities within the VR environment. This means A-Frame components could be used to add functionality to a UI menu.[9]

Custom components are able to be created by registering the component you wish to create using `AFRAME.registerComponent`. The first parameter of `registerComponent` is the name of the component and the second is a schema object that defines the functionality of the component. Components also have several unique properties that help make the JavaScript programming easy. These properties include `schema`, `data`, and `el`. The property `schema` contains the component names, types and overall functionality. Next, the `data` property is the parsed data that is derived from the defined schema of the component. Finally, the `el` property is an extremely useful tool as it is what references an entity element. This makes it extremely easy to apply components to essentially any entity.[10]

4 Controller Interfacing

User interaction is a large portion of WebPhysicsVR. This means the users will need to have some sort of control scheme, whether it be a mouse and keyboard, HMD and controllers, or even a touchscreen in the case of mobile users. User inputs, such as button presses, clicks, or taps, need to be parsed and the appropriate actions need to be performed in return. The three technologies that will be reviewed in this section are the Gamepad API, the legacy Gamepad Extensions API, and once again, A-Frame.

4.1 Gamepad API

The Gamepad API is an API dedicated to helping developers access and respond to signals and inputs from gamepads in a simple way. It is able to respond to gamepads being connected and disconnected as well as access other information about the gamepads, such as what buttons are actively being held down. Gamepad API was developed by Mozilla in an attempt to simplify controller interfacing. This API makes implementing controller support trivial.

There are several interfaces provided by the API: `Gamepad`, `GamepadButton`, `GamepadEvent`, `GamepadPose`, and `GamepadHapticActuator`. Most of these are rather self explanatory: `Gamepad` represent the actual controller that is connected, `GamepadButton` represents a single button on a controller, `GamepadEvent` is an object that represents events related to gamepads, `GamepadPose` represents the orientation of a controller in a 3D space, and `GamepadHapticActuator` represents the hardware which provides haptic feedback, such as vibrations. All of these interfaces are important and have their uses, but for a VR project like WebPhysicsVR, `GamepadPose` would most definitely be useful.[11]

4.2 Legacy Gamepad Extensions API

WebXR also supports the legacy Gamepad Extensions API, which expands upon the Gamepad API to supply support for more complex devices. Controllers paired with virtual reality headsets are generally more complex and rely on advanced features, such as motion tracking. Thus, using the Gamepad Extensions would make sense as it would ease the ability to interface with VR controllers correctly. Similarly to Gamepad API, it also has a `GamepadPose` interface, however it is expanded on by being able to define the gamepad's linear and angular velocity and acceleration in addition to its position. It is clear that this could be extremely useful in a physics simulator like WebPhysicsVR.[12]

4.3 A-Frame

Once again, A-Frame as a prime option. A-Frame components can be taken advantage of to implement controller support as well as support for mouse and keyboard and even mobile touchscreens. In particular, the look-controls, wasd-controls, laser-controls, and cursor components are some of the most critical components that would aid in implementing user controls. For starters, the look-controls component implements the ability to control the rotation of the camera with all three control options: VR HMD, mouse, and touchscreen.[13] The cursor component provides interaction for hovering and clicking the mouse and the wasd-controls component allows control of an entity, generally the camera, with the WASD or arrow keys, making mouse and keyboard support possible.[14, 15] Finally, the laser-controls component implements tracked controls that shoot a laser from a VR HMD paired controller that can be used for interactions. Laser-based interactions scale the best across different pieces of hardware as it can range across several different degrees of freedom (DoF). Common Degrees of Freedom include 0 DoF (Google Cardboard), 3 DoF (GearVR with controller wands), and 6 DoF (HTC Vive).[16]

5 Conclusion

WebPhysicsVR will be a physics simulation that relies heavily on user interaction. Thus, it is important to include an interactive user interface that makes it an easy and enjoyable experience for the user. The first step in doing so is generating the interface. Several technologies are available for doing so, such as A-Frame primitives, ReactVR and Three.js. Next is to supply functionality to the UI, which can be done using JavaScript, Reticulum or A-Frame components. Finally, controller support is able to be implemented using the Gamepad API, the Gamepad Extensions API, or A-Frame. An obvious and common theme here is A-Frame. A-Frame has the capabilities of providing essentially everything needed in terms of creating and implementing an interactive user interface due to the fact that it operates on a high level. It operates on top of some of the other technologies referenced, such as JavaScript and Three.js. In conclusion, A-Frame is the best choice of technology going forward in the development of the user interfacing.

References

[1] "Introduction: Getting started," https://aframe.io/docs/0.8.0/introduction/, 2018, accessed: 11/2/2018.

[2] "Primitives: `<a-box>`," https://aframe.io/docs/0.8.0/primitives/a-box.html, 2018, accessed: 11/2/2018.

[3] "Primitives: `<a-plane>`," https://aframe.io/docs/0.8.0/primitives/a-plane.html, 2018, accessed: 11/2/2018.

[4] "Primitives: `<a-text>`," https://aframe.io/docs/0.8.0/primitives/a-text.html, 2018, accessed: 11/2/2018.

[5] M. Argmstrong and A. Imm, "Introducing the react vr pre-release," https://developer.oculus.com/blog/introducing-the-react-vr-pre-release/, 2016, accessed: 11/2/2018.

[6] mrdoob, "three.js," https://github.com/mrdoob/three.js/, 2018, accessed: 11/2/2018.

[7] "three.js: Planegeometry," https://threejs.org/docs/index.html\#api/en/geometries/PlaneGeometry, 2018, accessed: 11/2/2018.

[8] skezo, "Reticulum," https://github.com/skezo/Reticulum, 2017, accessed: 11/2/2018.

[9] "Component," https://aframe.io/docs/0.2.0/core/component.html, 2018, accessed: 11/2/2018.

[10] "Component: Under the hood," https://aframe.io/docs/0.2.0/core/component.html\#under-the-hood, 2018, accessed: 11/2/2018.

[11] "Gamepad api," https://developer.mozilla.org/en-US/docs/Web/API/Gamepad_API, 2018, accessed: 11/2/2018.

[12] "Gamepad extensions," https://w3c.github.io/gamepad/extensions.html, 2018, accessed: 11/2/2018.

[13] "Components: look-controls," https://aframe.io/docs/0.8.0/components/look-controls.html, 2018, accessed: 11/2/2018.

[14] "Components: cursor," https://aframe.io/docs/0.2.0/components/cursor.html, 2018, accessed: 11/2/2018.

[15] "Components: wasd-controls," https://aframe.io/docs/0.2.0/components/wasd-controls.html, 2018, accessed: 11/2/2018.

[16] "Components: laser-controls," https://aframe.io/docs/0.8.0/components/laser-controls.html, 2018, accessed: 11/2/2018.