CS Capstone Design Document

December 2, 2018

Creating Immersive Experiences on the Web using VR and AR.

Prepared for

Intel

Alexis Menard _____ _____
Signature *Date*

Prepared by

Group 47

Brooks Mikkelsen _____ _____
Signature *Date*

Evan Brass _____ _____
Signature *Date*

Jonathan Jones _____ _____
Signature *Date*

Brandon Mei _____ _____
Signature *Date*

Tim Forsyth _____ _____
Signature *Date*

Abstract

This document is a software design specification that is intended to outline the various components used in the creation of the WebPhysicsVR simulation. There is a list of requirements that must be met for this project, their technical solutions, descriptions of those technologies and rational for using them.

CONTENTS

1 Introduction

This software design document is intended for the WebXR Senior Capstone Project for Oregon State University. The stakeholder of this project is Alexis Menard of Intel's open source software division. The goal of the project is to provide an example project for other developers interested in using the up and coming WebXR API built for creating VR and AR experiences on the web. Another goal of the project is to showcase an application that can be impossible to perform without a VR and AR device while progressively enhancing the experience to users. The website will serve an additional purpose as an immersive educational physics application.

We will be building a virtual reality physics simulation for use within classrooms to demonstrate how to use the API while creating something valuable for physics students at the same time. The project can be divided into individual components for the purpose of project planning, including: project setup, site hosting, physics, graphics, audio, and user interface. This document will list project requirements, then address these using design viewpoints, views, and elements, and rationales.

2 Design Stakeholders and their Concerns

2.1 Physics

2.1.1

The physical constants for a scene or object can be changed.

2.1.2

Objects in the scene must be interactive through translation, scaling, rotation and collision.

2.2 Graphics

2.2.1

3D content is rendered onto the connected devices.

2.2.2

Content is tailored to mobile devices when accessed through one.

2.3 Performance

2.3.1

Simulation is rendered in real time.

2.3.2

Rendering the content should fit within frame budget in VR, typically within 11 milliseconds.

2.3.3

Frame rate is 60-90fps on connected HMDs.

2.4 Client Reach

2.4.1 Mobile Networks and Offline Use

Content must be accessible from mobile speed networks. Stretch: Content may be ”installable” to make it available offline.

2.4.2 Inexpensive and Available Hardware

Content must render comfortably on cell phones manufactured later than two or three years ago.

Content must be interact-able without an external controller.

2.5 User Interface Visualization

2.5.1

Users must be able to choose to display information about an object, such as velocity, acceleration, and mass.

2.5.2

A menu must be available for users to switch stations or select objects.

2.6 User Interface Functionality

2.6.1

A scene can be paused/frozen.

2.6.2

An object can be paused/frozen.

2.6.3

Users can bring objects into the scene through interfaces in the simulation.

2.7 Hardware Interfacing

2.7.1

Users will be able to interface using mobile devices.

2.7.2

Users will be able to interface using HMDs whether they are opaque, transparent or utilize video pass-through.

2.7.3

Users must be able to interact with objects in the scene using position tracked controllers.

2.7.4

Users must be able to interact with objects in the scene using a mobile touchscreen interface.

2.7.5

Simulation pauses when a device has been disconnected and waits until reconnection to resume.

2.7.6

Website will notify user if the web browser or hardware is not compatible with WebXR.

2.8 Web Hosting

2.8.1

Users must be able to host directly from the GitHub repository.

2.8.2

2.9 Version Control

2.9.1

Users must be able to edit, push, pull request, and the changes are live.

3 Design Viewpoints

3.1 Physics

Concerns: 2.1.1, 2.1.2, 2.2.2, 2.3.2, 2.6.1, 2.6.2
Elements: Cannon.js

Analytical Methods: The simulation must be animated in a realistic way. Objects must interact with each other and be interactive with the user.
Viewpoint Source: Jonathan Jones

3.2 Graphics

Concerns: 2.2.1, 2.2.2, 2.4.2
Elements: Three.js
Analytical Methods: 3D content needs to be rendered onto the website and tunneled to the respective devices accessing the website. VR must supported.
Viewpoint Source: Jonathan Jones

3.3 Performance

Concerns: 2.2.2, 2.3.2, 2.3.3, 2.4.2
Elements: WebAssembly (WASM), Rust, Web Workers
Analytical Methods: Doing things faster and more efficiently doesn't just mean a better frame rate for the end user, but also better battery life. When our computation is spread across the available cores the device has the option to run those cores at lower power levels or idle them.
Viewpoint Source: Evan Brass

3.4 Installability and Caching

Concerns: 2.2.2, 2.4.1, 2.4.2
Elements: Service Worker, HTML-Meta Elements, responsive images
Analytical Methods: Service Workers expose the browser's caching system to developers. Our service worker can prime the browser cache with VR assets and respond to asset requests when disconnected. To describe to the browser what modules and assets will be needed later, we will use HTML meta elements. On parallel serving protocols like (G)QUIC and HTTP 2 the browser can load those assets over a low priority stream or maintain a connection to the server while waiting for the app to request those assets. We will also use responsive images to give the browser multiple options for an image source and it will pick whichever one would load best in terms of resolution, supported format, etc.
Viewpoint Source: Evan Brass

3.5 Audio

Concerns: Delivery of event driven spatial audio.
Elements: HowlerJS
Analytical Methods: Spatial audio is required to achieve full immersion into the physics simulation. Audio clips must be played when events happen in the scene.
Viewpoint Source: Jonathan Jones

3.6 Web Hosting

Concerns: 2.8.1

Elements: GitHub Pages

Analytical Methods: The web hosting services must able to host static web pages for GitHub users to publish content. GitHub Pages can also be served securely which is a requirement for using service workers. **Viewpoint Source:** Brandon Mei

3.7 Version Control System

Concerns: 2.9.1

Elements: GitHub

Analytical Methods: The service must host models assets, including audio, texture, models, and video files.

Viewpoint Source: Brandon Mei

3.8 Blog Publishing System

Concerns: 2.8.1

Elements: Jekyll

Analytical Methods: The product must take contents right from GitHub repositories to static websites.

Viewpoint Source: Brandon Mei

3.9 Language

Concerns: Project Setup

Elements: TypeScript

Analytical Methods: The product must compile to browser-executable, bug free JavaScript code.

Viewpoint Source: Brooks Mikkelsen

3.10 Asset Bundler

Concerns: Project Setup

Elements: Parcel

Analytical Methods: The product must download all required assets on page load or before they are required to be displayed.

Viewpoint Source: Brooks Mikkelsen

3.11 Continuous Integration

Concerns: Project Setup

Elements: Circle CI

Analytical Methods: The product be written with correct and maintainable code.

Viewpoint Source: Brooks Mikkelsen

3.12 User Interface Visualization

Concerns: 2.5.1, 2.5.2

Elements: Three.js, A-Frame

Analytical Methods: Various user interfaces, such as a menu, are required for users to interact with the simulation. These user interfaces need some sort of graphical visualization.

Viewpoint Source: Tim Forsyth

3.13 User Interface Functionality

Concerns: 2.6.1, 2.6.2, 2.6.3

Elements: JavaScript, A-Frame

Analytical Methods: User interfaces need some sort of functionality for users to interact with.

Viewpoint Source: Tim Forsyth

3.14 Hardware Interfacing

3.14.1 Head Mounted Display

Concerns: 2.7.2, 2.7.3

Elements: Gamepad API

Analytical Methods: The product should be compatible with VR HMDs and controllers for users to interact with the environment with 6 DoF.

Viewpoint Source: Tim Forsyth

3.14.2 Mobile Device

Concerns: 2.7.1, 2.7.4

Elements: Gamepad API, Reticulum

Analytical Methods: The product should be compatible with most mobile devices, including support for those with Daydream compatibility to offer a wide variety of experiences.

Viewpoint Source: Tim Forsyth

3.14.3 Personal Computer

Concerns: 2.7.5, 2.7.6

Elements: WebXR

Analytical Methods: The product should be playable on a computer browser with the ability to switch to a full VR experience if the appropriate hardware is detected.

Viewpoint Source: Tim Forsyth

4 Design Views

4.1 Physics

A physics simulation needs a physics engine to function properly. Cannon.js is a lightweight 3D web physics engine written entirely in Javascript. It provides rigid body dynamics, discrete collision detection, friction, restitution, object constraints, etc.

4.2 Graphics

WebGL will be used to render graphics to the connected devices and the website. The website will use a JavaScript library to implement WebGL features. 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. The library provides Canvas 2D, SVG, CSS3D and WebGL renderers.

4.3 Audio

A physics simulation should have sound for it to be immersive in the VR world. The applicaiton will use the WebAudio API to deliver event driven spatial sounds to the simulation. HowlerJS is an API for WebAudio that makes its implementation easier. Spatial audio is supported as well as event playback.

4.4 Web Hosting

The hosting will need to host static web pages for GitHub users, blogs, and project documentation. We will be using GitHub pages as there are no required databases to setup and no server to configure. It also needs to serve all of the project site from a personal URLs that is tied to an organization or GitHub account. The GitHub pages will automatically build and deploy our site.

4.5 Version Control Systems

The hosting service must have a repository to sort all the files that can be accessed with a unique URL. We will be using GitHub to manage repositories and it also will be used to host open-source projects. GitHub provides access control and features such as bug tracking, feature request, and task management. If we need a static site generator to manage and push changes to our GitHub repository, we can use Jekyll, as it re-generates all the HTML pages for the websites each time we commit a file but it depends on templates.

4.6 Blog Publishing System

The hosting sites must converts the files into a static website that can be deliver with any standard web servers. That means it needs to automatically generate the HTML code in the background if there are changes that are made to the files. Jekyll is a engine of GitHub Pages that can be use with each other, while it provides updates and HTML rendering.

4.7 Language

The application must run in modern browsers that have implemented the WebXR API. That means that the code must compile down to JavaScript (or be written in JavaScript). We will be using TypeScript to write code for the client side application. TypeScript is a language written by Microsoft that provides static typing on top of traditional JavaScript syntax. It can be compiled down to JavaScript for use in both Node.js and in browsers.

Rust is a systems programming language founded by Mozilla for designing programs with safe memory sharing and simpler parallelism without the usual tradeoffs of a garbage collector or heavy runtime.

4.8 Asset Bundler

The product must download all required assets on page load or before they are required to be displayed. We will be using Parcel to bundle all of our assets, including code, images, 3D object files, and textures. Parcel takes a single HTML file as an entrypoint, and then resolves all dependencies in a tree format without needing any configuration. It also supports code splitting using dynamic imports with no configuration necessary. If we need more configuration in our asset bundler, we can use Webpack, as provides all this functionality, but requires more configuration.

4.9 Continuous Integration

The product must be written in a clean, easily maintainable fashion. To ensure this, we will use Circle CI to build and test our application. Circle CI is a continuous integration service that can hook into GitHub and can be configured to build and test our application after each push to GitHub and before each pull request is merged to ensure the quality of our code.

4.10 User Interface Visualization

User interfacing needs some sort of graphical representation that users will be able to view and interact with. This will be accomplished by using Three.js along with A-Frame to create the necessary models to be used as a graphical interface. A-Frame provides primitives that simplifies creating objects, such as a flat plane, that can be used in the creation of user menus or graphical displays. Three.js can also achieve this at a lower level, which will be used in adding finer details.

4.11 User Interface Functionality

Many of the graphical user interfaces will need some sort of functionality, otherwise they will just be objects that do nothing. The main user interface functionality is menu functionality. The user must be able to interact with a menu to select an experiment station, pause and play objects or the environment, spawn or delete objects, and others. Nearly everything the user can interact with requires some sort of interface. While the direct functionality may not be defined with A-Frame, A-Frame components will be used to apply the functionality to objects, in this case the interface models.

4.12 Hardware Interfacing

4.12.1 Head Mounted Display

HMDs must be supported, therefore some sort of interfacing is needed to set up the peripherals such as the controllers and head tracking. This will be done using Gamepad API and A-Frame. A-Frame provides a very high level built in component, laser-controls, that provides tracked controls with a laser or ray cursor. Gamepad API is a much lower level API that these A-Frame components are built from. Gamepad API will be used for finer details in creating specific controls and gamepad setup.

4.12.2 Mobile Device

Mobile devices should be supported, in particular those with Google Daydream support. The high level A-Frame component, laser-controls, includes support for Daydream controls. This means it will be possible to set up support for various control schemes at once by using the laser-controls component. Touchscreen should also be supported for devices without Daydream support. This will be achieved by using various A-Frame components with touchscreen support, such as the look-controls component that allows the camera to be moved by using a touchscreen with touch-drag support.

4.12.3 Personal Computer

Computers should be supported if they are running in a compatible web browser. It should display the environment in their web browser with the ability to move with a keyboard and interact with the environment by using a mouse. This will be achieved by using several A-Frame components: wasd-controls, look-controls, and cursor. The component wasd-controls will provide the ability to move an object by using the wasd keys, most likely the camera, look-controls allows the user to rotate an object when the mouse is moved and cursor will hover and click functionality for the user to interact with other entities.

5 Design Elements

5.1 Physics

5.1.1 Cannon.js

Type: Physics Engine API
Purpose: Physics engine written in JavaScript. Animates objects in the scene with realistic physical movements.

5.2 Performance

5.2.1 Web Workers

Type: Web API
Purpose: Efficient utilization of modern multi-core CPUs for better battery life and response times.

5.2.2 WebAssembly

Type: Binary Instruction Set and Web API
Purpose: Faster and more efficient execution with a smaller runtime and no garbage collection.

5.2.3 Service Worker

Type: Web API
Purpose: Browser cache priming, pruning, and offline service.

5.3 Graphics

5.3.1 Three.js

Type: WebGL Framework
Purpose: Renders 3D content onto the site and connected devices for viewing.

5.4 Audio

5.4.1 HowlerJS

Type: Web API
Purpose: Handles audio events and playback on call.

5.5 Web Hosting

5.5.1 GitHub Pages

Type: Static Hosting
Purpose: GitHub Pages will be used for hosting static websites and able to edit, push, and make live changes directly from a GitHub repository.

5.6 Blog Publishing System

5.6.1 Jekyll

Type: Static Site Generator
Purpose: Jekyll will be use for taking content and spits out static website to be ready to serve a web hosting server. Jekyll is part of GitHub Pages, which can be used to host sites from the GitHub repositories.

5.7 Version Control Systems

5.7.1 GitHub

Type: Hosting Platform (Version Control)
Purpose: GitHub will manage code and track version control changes of the code.

5.8 Language

5.8.1 TypeScript

Type: Language
Purpose: TypeScript will be used to write all client side code for the app, except for that interacting with the WebXR API.

5.8.2 Rust

Type: Language
Purpose: Safe systems programming with a small runtime and a WebAssembly cross-compilation target.

5.9 Asset Bundler

5.9.1 Parcel

Type: Build tool (Asset Bundler)
Purpose: Parcel will be used to bundle all of our assets, including code, images, 3D object files, and textures.

5.10 Continuous Integration

5.10.1 Circle CI

Type: Continuous Integration
Purpose: Circle CI will be used to build and test our application after every push to GitHub and before each pull request is merged to maintain code quality.

5.11 User Interface Visualization

5.11.1 Three.js

Type: API
Purpose: Three.js will be used to create graphical models that will be used for user interfaces, such as menus or graphical displays.

5.11.2 A-Frame

Type: Framework
Purpose: A-Frame will assist Three.js in creating these menus by using A-Frame specific tools, such as primitives.

5.12 User Interface Functionality

5.12.1 JavaScript

Type: Language
Purpose: JavaScript will be used to write functional code that performs requested actions from a user interacting with a menu.

5.12.2 A-Frame

Type: Framework
Purpose: A-Frame will be coupled with JavaScript to apply functional code to specific objects, in this case the user interface models, by using A-Frame components.

5.13 Hardware Interfacing

5.13.1 Gamepad API

Type: API
Purpose: Gamepad API provides tools to connect and map controllers to perform specific actions.

5.13.2 A-Frame

Type: Framework
Purpose: A-Frame provides built in components that aid in setting up controls for a wide range of devices. This includes HMDs, mouse and keyboard, and mobile devices with gaze interaction.

5.13.3 WebXR

Type: API
Purpose: WebXR provides capabilities for detecting connected hardware which is useful in determining compatibility.

6 Design Rationale

6.1 Physics

Cannon.js was chosen as the physics engine because of its enormous feature set and VR support. It is also the built in physics engine for the A-Frame library. Additionally, Cannon.js has an active community and extensive documentation. There are lots of examples with source code that could be helpful during implementation of this project.

6.2 Graphics

The decision to use Three.js is mostly due to its popularity as a 3D graphics API in the web development world. It has a proven record of quality and efficiency along with a large community that supports it. It has VR support and is built into the A-Frame library.

6.3 Audio

There were not a lot of options for Audio APIs, still, HowlerJS stood out among its peers. Its abstraction of the WebAudio API and support for event driven spatial audio was much desired for this simulation. This API makes the use of the WebAudio API much easier and in the event that it is unsupported on certain browsers, it falls back to HTML.

6.4 Web Hosting

The web hosting services we choose to use GitHub Pages because it will offer us static web pages for GitHub users. Static web page is a web pages that can deliver the users exactly what is stored. GitHub Pages has intergrated with Jekyll to automatically update GitHub Pages servers and regenerate the site. To host a website from GitHub Pages we will need to create a repository with a repository name plus github.io. Then enter the project folder and add/push an index.html file in order to host the website easily.

6.5 Version Control Systems

We will be using GitHub to our project repository because it will allow us to have Git repository hosting service and publishes web sites. GitHub host code repositories for all types of language, not just HTML and CSS. It also has a feature to make it easy to create a multi-file website hosted at GitHub Pages. GitHub also provides access control and collaboration features like fork, pull, and merge.

6.6 Blog Publishing System

Since Jekyll is intergrated with GitHub Pages and repository of GitHub. We chose to use Jekyll as a very simple, blog-aware, and static site generator for our project. While not have to worry with needless complexity and configuration, it allows us to concentrate on the content instead.

6.7 Language

We will be using TypeScript for the client side application because it will give us static typing on top of traditional JavaScript. Static typing has the advantage of easier refactoring and less error-prone code. TypeScript compiles to JavaScript so that it can be run within the browser. While it does add some overhead to the developers' workload, it should reduce the amount of time we spend debugging our application. All of the JavaScript libraries and APIs we will be working with except for WebXR have TypeScript type definitions that we can utilize.

6.8 Asset Bundler

We chose to use Parcel to bundle our client-side assets because of how simple it is to set up. There is essentially no configuration - we will only need to include the root TypeScript file in our HTML page, and Parcel will figure out the dependencies based on our import statements. If we determine that we need more granularity of control in our configuration, we will switch to using Webpack, since it provides the same features, but requires (and allows) more configuration.

6.9 Circle CI

We chose to use Circle CI because it is simple to configure and free for open source projects. It also has built in webhooks with GitHub, so it will be simple to configure it to build and test our application after pushing changes to GitHub and before merging pull requests. Travis CI is very similar in that it is free for open source projects and easily configurable, but build tasks tend to run somewhat slower on Travis CI than Circle CI.

6.10 User Interface Visualization

The design decision to use A-Frame comes from the extensive amount of primitives that are available built in to the framework. It's built on top of Three.js, which is the graphics API we will be using. Three.js is used since it is a well regarded API for 3D modeling. The user interface will be using 3D modeling rather than some 2D user interface toolkit, such as Open Source Qt, because it will be in a VR environment. Menus and other graphical user interfaces cannot exist on the forefront of the screen, like it would on a normal program, in a VR simulation without causing strain on the user. This is because the user's eyes will be very close to the screens due to the nature of HMDs. Instead, spatial UI is needed, which is essentially UI that exists within the environment of the simulation. Therefore, some sort of 3D graphics modeling will be needed for creating the graphical user interfaces, in the case of this project being A-Frame and Three.js.

6.11 User Interface Functionality

User interface functionality is vital to the project working correctly. Without some sort of user interface functionality, it would be impossible for the user to interact with a lot of the entities in the simulation. Many things, such as changing the properties of objects, pausing an object or the environment, changing laws of physics, and spawning or deleting items depend on some sort of user interface. A-Frame is chosen as a tool that will connect some sort of functionality to a GUI using components. The choice of choosing JavaScript for writing the actual functionality stems from that a custom component can be created using JavaScript. This means some sort of functionality can be created, such as deleting an object, as a new component which can then be applied to an object, such as a button on a modeled interface.

6.12 Hardware Interfacing

Hardware interfacing is an absolute necessity; without it, the user would be unable to do anything. The choice of supporting low end devices, such as mobile devices using Google Cardboard, stems from the goal of the project: providing an affordable method of physics education. Those who aren't able to afford the expensive materials required

to perform some physics experiments likely won't be able to afford a 500 dollar VR headset, not to mention a computer that can effectively support it. The reason behind choosing A-Frame as the tool for interfacing between the different pieces of hardware is that it provides nearly everything needed for supporting a wide variety of devices in the form of built in A-Frame components. It also makes use of the Gamepad API and extensions, which is built into the WebXR API that is the basis of this project.