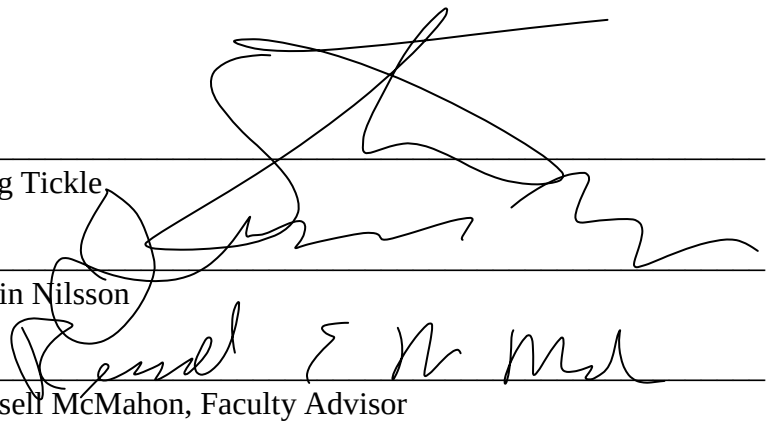
RealWeb VR

by

Greg Tickle and Justin Nilsson

Submitted to
the Faculty of the School of Information Technology
in Partial Fulfillment of the Requirements for
the Degree of Bachelor of Science
in Information Technology



Greg Tickle	4/17/2017 Date
Justin Nilsson	4/17/2017 Date
Russell McMahon, Faculty Advisor	4/17/2017 Date

University of Cincinnati
College of
Education, Criminal Justice, and Human Services

April 2017

Contents

Tables and Figures

Abstract

Virtual reality (VR) opens up many opportunities not only for entertainment, but also for evolving many popular applications beyond their current two dimensional capabilities. This growing technology impacts users with its ability to transport people to anywhere imaginable and to equip them to interact naturally with virtual environments. For these reasons and more, one of the biggest technologies to benefit from VR's capabilities will be the World Wide Web. Although the Web is already one of the most widely used technologies today, there is not yet a practical and efficient option for browsing in VR. Some current VR Web browsers present the Web as a social experience, with Web sites represented by rooms, which can make browsing more time consuming. Also, these solutions do away with much of the Web that already exists and require that developers create pages built just for VR. Other options simply present the Web as a standard two dimensional plane, which offers no benefit over traditional monitors.

This paper introduces a new kind of VR browsing application, RealWeb VR, which leverages the new capabilities of VR and allows users to interact with Web pages directly and intuitively using their own eyes and hands. With RealWeb VR, developers are not required to redesign entirely new pages just for VR; instead, they can simply add CSS properties that will render the HTML elements in stereoscopic 3D. RealWeb VR is built on several frameworks to create a unique Web portal that puts users face-to-face with a giant, dynamic screen that shows the entire Web page at one glance. This application is compatible with major VR head-mounted displays and utilizes handheld controls such as the HTC Vive controllers and Oculus Touch.

Introduction

After recent advances in technology, VR has had a resurgence of interest after the release of several Head Mounted Displays (HMDs) that finally deliver on the promises of what past visionaries have said VR could be. The HTC Vive and Oculus Rift stand at the forefront of this reborn industry, enabling consumers to "teleport" anywhere imaginable. Through the use of natural stereoscopic 3D and precise head and hand tracking, these new HMDs deliver on the idea of presence[1], or when the mind is convinced it is in an alternate location.

These compelling devices stand to offer new and exciting experiences not only to technology enthusiasts, but also to a much wider market. In fact, market analysts predict that over 200 million head mounted displays could be sold by 2020[2], with hardware and software revenues in the virtual and augmented reality space reaching up to $162 billion.[3] With this kind of potential industry growth, VR is poised to become, as Mark Zuckerberg has said, "the next computing platform"[4]. Therefore, a new opportunity arises for an already ubiquitous application: the Web browser.

Problem Statement

After decades of development and progress, the current iteration of the traditional browser has likely reached its final form. Computer screens, while serving their two dimensional purpose, limit interaction to their flat surfaces, whether with a mouse and keyboard, or with

---

[1] Woodrow Barfield and Thomas A. Furness, Virtual Environments and Advanced Interface Design (New York: Oxford University Press, 1995), 473.

[2] John Gaudiosi, "Over 200 Million VR Headsets to Be Sold by 2020", Fortune, January 21, 2016, Accessed November 02, 2016, http://fortune.com/2016/01/21/200-million-vr-headsets-2020/.

[3] "Worldwide Revenues for Augmented and Virtual Reality Forecast to Reach $162 Billion in 2020, According to IDC", idc.com, August 15, 2016, Accessed November 02, 2016, https://www.idc.com/getdoc.jsp?containerId=prUS41676216.

[4] Mathias Döpfner and Die Welt, "Mark Zuckerberg Talks about the Future of Facebook, Virtual Reality and Artificial Intelligence", Business Insider, February 28, 2016, Accessed October 31, 2016, http://www.businessinsider.com/mark-zuckerberg-interview-with-axel-springer-ceo-mathias-doepfner-2016-2?op=1.

fingertips on a mobile device. Meanwhile, human beings interact with their natural environment in three dimensions. They can move objects with their hands, bringing items closer for easier viewing. They can look around their environment, taking in sights in stereoscopic 3D. The traditional computer display lacks the means for allowing humans to interact naturally with a digital environment like a Web browser. The advent of VR finally offers the opportunity to break free from these constraints.

There have been some attempts at VR Web browsers. However, the VR browsers that currently exist either offer nothing new to the browsing experience, or they take a segmented approach that presents the Web as a series of rooms and portals where each room represents a Web site. While the latter method offers a unique take on browsing, it does not lend itself to practicality, nor does it increase efficiency or ease of use. Additionally, the need to travel from room to room by the user can induce a phenomenon called simulator sickness, which results from the body not feeling the acceleration or motion that the eye is seeing[5]. Most people looking for information on the Web are unlikely to spend the time needed to wander through rooms, especially if it makes them feel ill. There is an opportunity for an evolution of the Web browser that utilizes the capabilities that VR offers while retaining an appeal to a mainstream audience.

The museum-like approach described above encounters another issue; namely, that developers would need to recreate their entire Web sites from scratch if they want them to take advantage of VR. While some organizations and businesses would certainly benefit from having their own virtual environment that they can tailor to their needs, the likely majority of Web site owners have neither the ability nor the need to create 3D rooms from the ground up. Web

[5]"Simulator Sickness", Oculus, Accessed October 30, 2016, https://developer3.oculus.com/documentation/intro-vr/latest/concepts/bp_app_simulator_sickness/.

developers should not need to throw away their existing pages and start from scratch just to take advantage of VR's capabilities. We want to make developing for the VR Web simple.

Solution

RealWeb VR offers a solution to these problems in that it builds on traditional browsers and adds additional features and functionality that only VR can give. In other words, RealWeb VR is the next evolution of the browser. It preserves pages already in existence, but transplants them into a VR world where users can interact naturally with them. This approach retains familiarity with traditional browsers and limits the need for user movement, which will make users more comfortable. The unique capabilities of VR offer increased efficiency and improved experience. VR also offers an exciting possibility for developers to literally add an extra dimension to their web pages. VR's capabilities mean that web designers can add untapped creativity to their web pages through the use of 3D elements that leap off the page. This brings an entirely new dynamic to web development.

Report Overview

This report will detail our efforts to create a VR Web browser with features such as hand controls, 3D elements, full page viewing, ease of use, familiarity, and CSS based VR page creation. The following sections include Project Concept, Use Case Diagram, Design Objectives, Technical Approach, Budget, Gantt Chart, User Profile, Test Plan, and Recommendations for Improvement.

Project Concept

In order to appeal to a mainstream market, RealWeb needs to offer an advantage in experience and efficiency, while still remaining familiar. To meet this challenge, a variety of

potential features were considered. The following list contains the features that most satisfy these demands:

1. RealWeb can show the entire Web page at one glance. This means that users would no longer have to scroll through a page in a monitor.
2. Virtual reality offers gesture based hand controls. This allows users to naturally "reach into" their browser, manipulating and interacting with elements, text boxes, and buttons as if they were real life objects. Additionally, hand controls enable mobile phone style pinch-and-zoom and swiping gestures for screen resizing and scrolling. VR controls improve on the traditional mouse and keyboard combo by allowing users to interact with pages in a more natural way.
3. Since VR utilizes stereoscopic 3D, page elements can be rendered in 3D. Besides being a novel visual effect, this will allow developers to add a new dimension in Web page design. One highlight in experiencing VR games has been to see interesting layered 3D menus and user interfaces. An exciting possibility arises to introduce these kinds of features to any web page. Also, when combined with gesture based controls, it could be possible for users to bring elements closer, perhaps for easier reading.
4. Instead of creating 3D environments from scratch, Web developers have the option to retain their existing pages while being able to make them VR compatible.
5. The most important feature that RealWeb VR offers is ease of use. If this new Web browser is to be successful with a mainstream audience, the controls and interface should be easy to grasp, and they should retain the familiarity of traditional browsers.

Virtual reality concepts are difficult to visualize without experiencing them, especially for those who have never tried VR before. Figure 1 attempts to portray how these features look in RealWeb VR.

Figure 1. The RealWeb VR Experience

The above figure roughly illustrates the RealWeb VR experience with its hand controls and 3D elements. The user has the ability to move the entire screen up and down and resize it. Also, the user will be able to view the entire page at one time, instead of scrolling through a window on a monitor.

Design Objectives

In order to reach the goal of delivering an intuitive and easy to access VR browsing experience, a multitude of new technologies, engines, and frameworks had to be considered. This application would need to be easy for mainstream users to download and use with as little set up as possible. Furthermore, beyond simply rendering a page in VR, the user should be provided

with an easy interface that allows for an experience even more intuitive than modern day browsing. In keeping with the above requirements, we researched ways to create a 3D VR scene and to render a Web browser within that scene. A Chrome extension that injects the RealWeb code into a web page to transform it into a 3D scene proved to be the easiest path to take. It would also ensure that users can easily install and use the application. Figure 2 provides a brief depiction of the flow of a Web page from the Internet to an HMD.

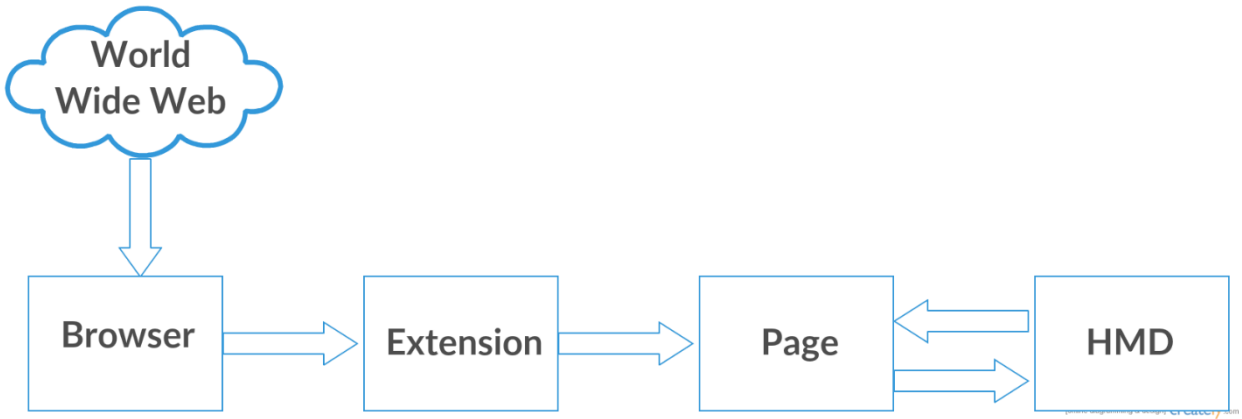


Figure 2. Flow chart overview of RealWeb VR.

When running the RealWeb VR application, Web pages that are loaded to the browser are run through a browser extension that creates a VR scene. This extension also renders the Web page to a plane that is presented to the user as a 3D element. The entire scene including the rendered Web page renders to the HMD, which constantly sends back updated positioning data for the purpose of updating the user's position. The section "Technical Approach" will describe how this rendering is accomplished in more detail.

Since developers usually create 3D scenes using a game engine such as Unity or Unreal, this was the first method researched. However, we quickly found that game engines are difficult to use for Web purposes and that they would require a lot of manual work to convert HTML to a raw texture that could be displayed "in-game". Furthermore, the use of a game engine would

require users to download a separate executable in order to use our app, which violates the requirement that this browser be easy to use. As a result, we abandoned the idea of using a game engine and instead began to search for ways to render a 3D VR scene directly from a Web browser.

Technical Approach

Technology Stack

When this project began, nothing existed that could serve as an example for how to begin to piece together a VR browser that could display parts of web pages in 3D while giving users complete control and vision of the entire page. Therefore, it was necessary to use several obscure and experimental frameworks. This means that the RealWeb browser builds onto several technologies that constantly change through updates and bug fixes, meaning that the process has been one of instability. However, this also means that the process has been a valuable learning experience, especially in dealing with open source technology, of which RealWeb utilizes a great deal. RealWeb makes use of the following open source software:

- The WebVR API sends the HMD positioning and other data to the Web browser, making it possible to run VR experiences within browsers that support the technology, such as certain versions of Chromium and Firefox Nightly.
- The A-Frame framework builds onto WebVR to create VR compatible 3D scenes within a Web browser.
- HTMLShader renders HTML elements as textures to A-Frame elements.

- RealWeb VR also makes use of an open source CSS parser to parse the CSS files for rendering custom 3D elements.[6]

The following graphic gives a quick view of the technologies used in the RealWeb VR browser:

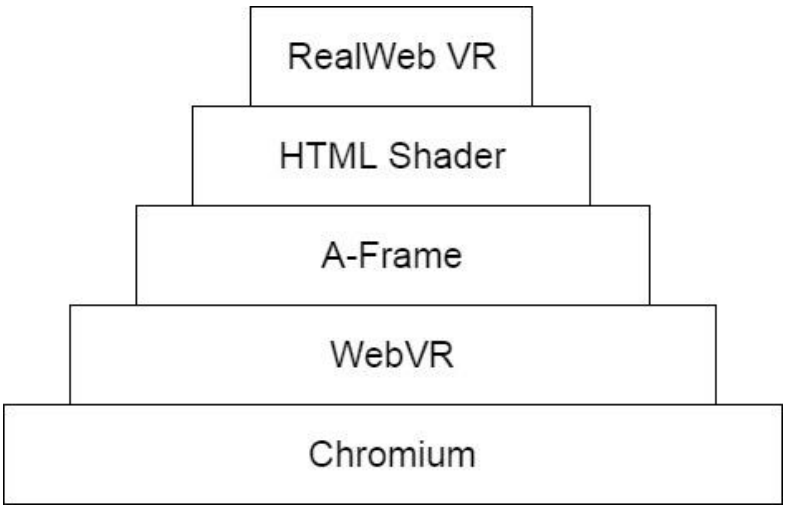


Figure 3. The RealWeb VR technology stack.

In order to build a VR browser within a browser, we needed a way to gather data from the headsets so that user position and orientation could be tracked. An experimental API called WebVR showed promise since it gives access to HMDs from Web browsers. This interface

> provides support for exposing virtual reality devices — for example head-mounted displays like the Oculus Rift — to Web apps, enabling developers to translate position and movement information from the display into movement around a 3D scene. This has numerous very interesting applications, from virtual product tours and interactive training apps to super immersive first person games.[7]

[6] This program can be found at https://github.com/jotform/css.js

[7] "WebVR API", Mozilla Developer Network, Accessed November 02, 2016, https://developer.mozilla.org/en-US/docs/Web/API/WebVR_API.

While WebVR provides means to transfer HMD data to Web apps, a WebVR compatible 3D framework was still necessary to create a 3D scene to hold the virtual browser. Several possible options appeared such as ThreeJS and WebGL, which render 3D graphics to browsers. However, both would require manual and time-consuming integration with WebVR, a likely impossibility given this project's time constraint. After additional research, we decided to leverage a new open source Web framework called A-Frame.

Rendering a 3D Scene

A-Frame allows developers to create and customize 3D rendered VR and non-VR scenes directly in the Document Object Model (DOM), effectively embedding the scene directly within a Web page. By having our scene exist within the DOM, we allow the user's existing browser to render the HTML. This way, we are able to use JavaScript and JQuery to read through the HTML document and pass data to our scene as we need. Furthermore, the open-source nature of A-Frame allows us to leverage user-created components such as HTML2Texture and HTMLShader to help convert elements on the DOM to textures we can then render in our scene.

We began research and development on RealWeb VR with the intention of creating our own Web application to which users could navigate through their browsers. However, since A-Frame scenes exist on the DOM, all navigation to sites would be required to occur via the use of an HTML iframe so that the DOM (and the VR scene) was not destroyed and re-created on every navigation event. The core problem with this approach is the lackluster security and support surrounding the iframe element. Many sites, including Google, Yahoo, Apple, and Reddit prevent the rendering of their sites within an iframe element, effectively cutting our application out of viewing many of the major popular sites.

To overcome this problem, our focus shifted from creating a Web application, to utilizing the capabilities of Chrome extensions. With a Chrome extension, we are able to create content-scripts that can inject all of our dependencies and A-Frame scenes into the DOM as the user navigates to any Web site. Through this method, we can read the body of the entire DOM, converting it into a texture that we can display in our injected A-Frame scene.

Converting a Web Page to a VR-Ready Page

A vital feature of RealWeb VR is the ability to transform any Web page on the Internet into a VR page. Using HTMLShader, any page can easily be rendered to a plane in a 3D scene; so, in that sense, any page is already VR ready. However, developers will likely want to take advantage of RealWeb's 3D element feature and render their elements as 3D elements that pop from the Web page. Fortunately, RealWeb requires no additional learning of languages or frameworks to enable a full VR experience that includes 3D elements. Simply adding custom CSS tags to any element in a CSS file should cause a 3D effect.

This feature is made possible by utilizing an open source CSS parser.[8] Since Web browsers throw away all CSS that they do not recognize, our application needed a way to preserve these custom tags that tell whether an element should have a 3D effect and the distance at which to present it. Using this CSS parser, the browser is able to parse the CSS file containing the tags into JSON and search through that JSON for our custom tags, thus preserving them for use in the application. Figure 4 shows an example of our custom VR tag for depth inside a CSS declaration:

[8] This program can be found at https://github.com/jotform/css.js

```
#content {
  float: left;
  width: 500px;
  margin: 10px 0 100px 0;
  background: #f8f8ff;
  line-height: 1.8;
  text-align: justify;
  VRDepth: 1;
}
```

Figure 4. CSS Sample with VR Tag

The VRDepth property that sets the 'content' ID to a depth of 1 would normally be discarded by the browser, but thanks to the CSS parser, it is preserved in JSON. In order to properly present elements of a web page in 3D space, more tags are needed than just VRDepth. Currently, in addition to VRDepth, RealWeb VR recognizes tags for VRRotation and also VREntityType, which indicates the desired type of 3D A-Frame object. This kind of customization allows web developers to reach new levels of creativity when developing for the VR web.

Interaction with Page Elements

Since this browser only presents web pages as textures rendered to 3D objects, page interactivity required additional work in order to allow users to navigate links. Our current approach is to iterate through the DOM and render each interactive element to a separate "pop-out" plane at their respective positions on the page. When a user points their controller at the pop-out and interacts with it, we can manually trigger an event for the respective element on the page.

Scaling and Positioning 3D Elements

Although the generated 3D elements can be created with depth rather easily by indication in the CSS file, these newly created elements still need to be sized and positioned correctly based on their original size and position on the page. Because the 3D elements are initialized as child elements of the main A-Frame plane, these elements are scaled accordingly. Thus, any element set to a scale of 1 is as large as the main plane. As a result, elements cannot be set to a scale of 1; they must each be sized based on a calculation of the percentage of the original elements size compared to the size of the original document, for both width and height.

```
var getElementSize = function (element) {
    //Width
    var elementWidth = element.width();
    var bodyWidth = $('body').width();
    var widthPercent = elementWidth / bodyWidth;

    //Height
    var elementHeight = element.height();
    var documentHeight = $(document).height();
    var heightPercent = elementHeight / documentHeight;

    return {width: widthPercent, height: heightPercent};
};
```

Figure 5. Code sample for sizing 3D elements

The above function in Figure 5, getElementSize(), demonstrates how RealWeb deals with calculating each element's size. For width, the element width is divided by the width of the HTML body to give a percentage. The function uses the same manner for height and returns a JavaScript object to be used in the creation of a new element plane.

Positioning proved to be slightly more complicated since the A-Frame plane uses Cartesian coordinates instead of referencing the top left corner, as do HTML pages. In this case, the default coordinates given by the elements need to be converted to Cartesian using the formula of

$$cartY = -1(elYPos - screen\_height/2)$$

where *cartY* is the resultant Cartesian coordinate for the element's Y-axis and *elYPos* is the original coordinate for the element's Y position.

VR Controller Support

One of the key focuses during the development of RealWeb VR was to provide a familiar and easy to pick up control scheme for our users. When first putting on the VR headset and taking the controllers in hand, users should be able pull from normal web browsing experiences to help them learn how to manipulate their environment and navigate between pages. To accomplish this task, we considered how people typically move through a page, zoom in and out, and interact with links. The control scheme we ultimately developed has taken concepts from both desktop and mobile browsing, improving the overall web browsing experience.

Figure 6. Oculus Touch and HTC Vive controllers.[9]

[9] Jon Martindale, "Fantastic Contraption Dev Says Oculus Touch and HTC Vive Controllers 'Almost Identical'", Digital Trends, April 14, 2016, Accessed March 18, 2017, http://www.digitaltrends.com/computing/fantastic-contraption-dev-says-oculus-touch-and-htc-vive-controllers-almost-identical/

The first focus around developing our control scheme was figuring out a method in which users could interact with links and buttons on a page. To accomplish this, we provided each dynamically generated floating element with a target attribute bound to the selector for that element on the original page. Using a combination of d3.js and the vive-cursor component, we were able to wire up a listener to each floating element that triggers a click by proxy. The result allows users to point their cursor at the desired floating element they wish to interact with and then pull the trigger to interact.

With navigation and element interaction working, our next task was to allow the user to manipulate the main plane or "page". Specifically, just as users are able to scroll with a mouse and swipe with a finger, we needed a method of scrolling that fit a VR experience. Our solution came in the form of giving the user the ability to grab the page by pointing their controller cursor at the plane and moving it around at will. The implementation is fairly simple in that we are able to fire off an event when the cursor collides with the plane in combination with a trigger press. From the event details, we are given the point of intersection and can calculate the difference in position of each intersection between each event.

One commonly used feature of web browsing on a mobile device is the ability to zoom in and out by pinching. This is especially useful on mobile devices when a site may not be responsive by design, requiring the user to zoom in to properly view content. For VR, we chose to eliminate this issue by allowing the user to change the size of the plane via the use of both controllers and their grip buttons. When the user presses both grip buttons on the controller, they can expand the main plane by moving their hands farther apart. Likewise, they are able to shrink the plane by moving their hands closer together.

With our implementation of moving the page, changing the size of the page, and interacting with page links and buttons, RealWeb VR provides a clear proof of concept of what is possible for VR Web browsing on traditional 2D sites. With the virtues of VR, users can perform actions like dragging the entire page to the specific point they want to see. Users can also effortlessly shrink and grow the page to a level that provides them the most comfortable viewing level. Lastly, they can navigate to and from sites with ease by the floating elements that signify what they can interact with.

The Finished Result

Figure 7. The above image shows the result of the various components of the RealWeb VR tech stack working together to create a VR Web browser.

Budget

The costs shown for the VR equipment listed below in Figure 4 reflect what was paid by this team; however, the cost for a PC capable of satisfactorily running VR HMDs has dropped

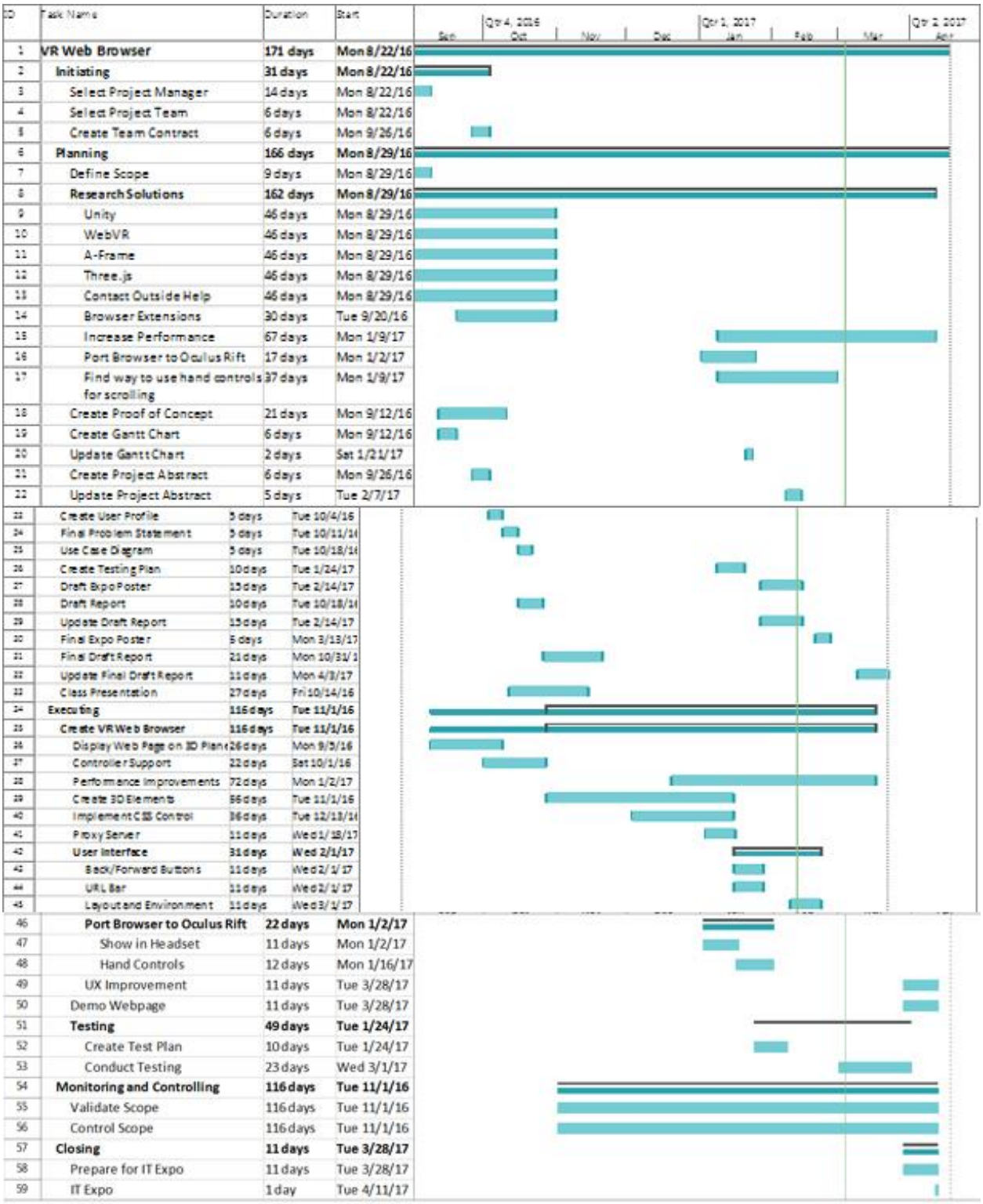
significantly during the development of this project. VR requires graphically powerful computer equipment; thus, the cost for an entry level VR machine was about $900 when the Oculus Rift and HTC Vive were first released. Today, due to advancements in VR rendering technology, the minimum system requirements for a VR-ready machine capable of running an Oculus Rift has dropped from $900 to $499 during the development of this project. The Rift and the Touch controllers have also seen a reduction in price, with both costing $599. For this reason, costs for someone looking to purchase a minimally viable VR setup including HMD, controllers, and compatible PC would be about $1100. However, this team's equipment was purchased before these price drops, so Figure 8 reflects a summary of the costs for the equipment used in order to complete this project.

Item	**Quantity**	**Cost**
VR-Ready PC	2	$2000.00
Oculus Rift w/ Controllers	1	800.00
HTC Vive w/ Controllers	1	800.00
Total Cost		**$3600.00**

Figure 8. Budget Breakdown

Gantt Chart

Due to this application being built from several new technologies, and since there are no other Web browsers like it as of the time of this writing, this project required much up front research time. As shown in Figure 9 below, the planning phase spanned most of both semesters, with researching solutions to how to implement important features being a major part of this project. The execution phase was interspersed throughout these periods of research, until the end in which time was mostly spent on execution. Monitoring and controlling the scope also came into play since difficulties in implementing some of the features of this browser required that we tone back the scope several times.



ID	Task Name	Duration	Start
1	**VR Web Browser**	**171 days**	**Mon 8/22/16**
2	**Initiating**	**31 days**	**Mon 8/22/16**
3	Select Project Manager	14 days	Mon 8/22/16
4	Select Project Team	6 days	Mon 8/22/16
5	Create Team Contract	6 days	Mon 9/26/16
6	**Planning**	**166 days**	**Mon 8/29/16**
7	Define Scope	9 days	Mon 8/29/16
8	**Research Solutions**	**162 days**	**Mon 8/29/16**
9	Unity	46 days	Mon 8/29/16
10	WebVR	46 days	Mon 8/29/16
11	A-Frame	46 days	Mon 8/29/16
12	Three.js	46 days	Mon 8/29/16
13	Contact Outside Help	46 days	Mon 8/29/16
14	Browser Extensions	30 days	Tue 9/20/16
15	Increase Performance	67 days	Mon 1/9/17
16	Port Browser to Oculus Rift	17 days	Mon 1/2/17
17	Find way to use hand controls for scrolling	37 days	Mon 1/9/17
18	Create Proof of Concept	21 days	Mon 9/12/16
19	Create Gantt Chart	6 days	Mon 9/12/16
20	Update Gantt Chart	2 days	Sat 1/21/17
21	Create Project Abstract	6 days	Mon 9/26/16
22	Update Project Abstract	5 days	Tue 2/7/17
23	Create User Profile	5 days	Tue 10/4/16
24	Final Problem Statement	5 days	Tue 10/11/16
25	Use Case Diagram	5 days	Tue 10/18/16
26	Create Testing Plan	10 days	Tue 1/24/17
27	Draft Expo Poster	15 days	Tue 2/14/17
28	Draft Report	10 days	Tue 10/18/16
29	Update Draft Report	15 days	Tue 2/14/17
30	Final Expo Poster	5 days	Mon 3/13/17
31	Final Draft Report	21 days	Mon 10/31/16
32	Update Final Draft Report	11 days	Mon 4/3/17
33	Class Presentation	27 days	Fri 10/14/16
34	**Executing**	**116 days**	**Tue 11/1/16**
35	**Create VR Web Browser**	**116 days**	**Tue 11/1/16**
36	Display Web Page on 3D Plane	26 days	Mon 9/5/16
37	Controller Support	22 days	Sat 10/1/16
38	Performance Improvements	72 days	Mon 1/2/17
39	Create 3D Elements	86 days	Tue 11/1/16
40	Implement CSS Control	36 days	Tue 12/13/16
41	Proxy Server	11 days	Wed 1/18/17
42	**User Interface**	**31 days**	**Wed 2/1/17**
43	Back/Forward Buttons	11 days	Wed 2/1/17
44	URL Bar	11 days	Wed 2/1/17
45	Layout and Environment	11 days	Wed 3/1/17
46	**Port Browser to Oculus Rift**	**22 days**	**Mon 1/2/17**
47	Show in Headset	11 days	Mon 1/2/17
48	Hand Controls	12 days	Mon 1/16/17
49	UX Improvement	11 days	Tue 3/28/17
50	Demo Webpage	11 days	Tue 3/28/17
51	**Testing**	**49 days**	**Tue 1/24/17**
52	Create Test Plan	10 days	Tue 1/24/17
53	Conduct Testing	23 days	Wed 3/1/17
54	**Monitoring and Controlling**	**116 days**	**Tue 11/1/16**
55	Validate Scope	116 days	Tue 11/1/16
56	Control Scope	116 days	Tue 11/1/16
57	**Closing**	**11 days**	**Tue 3/28/17**
58	Prepare for IT Expo	11 days	Tue 3/28/17
59	IT Expo	1 day	Tue 4/11/17

Figure 9. Gantt Chart

Use Case Diagram

As Figure 10 illustrates, users and developers will be able to interact with the RealWeb VR application through six use cases. The “Browse Web” use case will get its content from external Web sites.

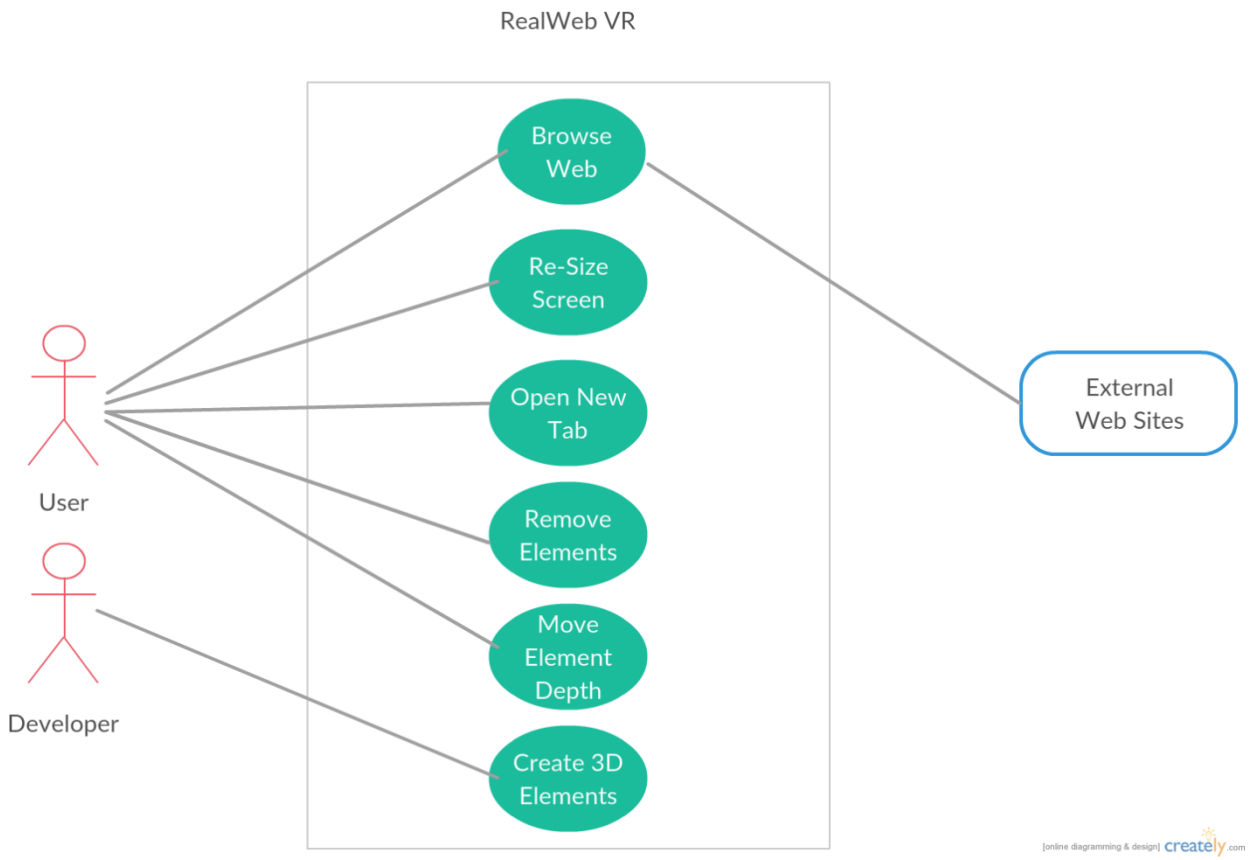


Figure 10. RealWeb VR Use Case Diagram

User Profile

Figure 11 shows that the initial user base is aimed towards tech oriented individuals familiar with VR and who already own the necessary peripherals to use RealWeb. We expect that crafting an experience towards a technical user base is an ideal first iteration solution towards gradual improvements through community feedback. Through this feedback we can

slowly improve the experience with the expectation that our number of users will grow as the application becomes more friendly and easy to use.

APPLICATION: VR Web Browser that lets users have full view of Web page with 3D elements. Provides support for hand-tracked input or gaze input (Mobile). Enables developers to easily create VR web pages.
POTENTIAL USERS: • VR Users • Gamers • Tech Enthusiasts • Web Surfers • Web Developers
SOFTWARE, INTERFACE, AND RELATED EXPERIENCE: End-users should have experience with common Web browsers, in addition to familiarity with their respective VR devices, controls, and software. Those wanting to develop for VR should have working knowledge of HTML and CSS.
EXPERIENCE WITH SIMILAR APPLICATIONS: Any current Web browser.
TASK EXPERIENCE: • Downloading and installing browser extension • Installation of Chrome Web Browser • Setup and use of VR systems and controllers
FREQUENCY OF USE: Variable; whenever the user desires to browse the Web in VR or when developers wish to test their VR Web pages.
KEY PROJECT DESIGN REQUIREMENTS THAT THE PROFILE SUGGESTS: • Easy setup • Intuitive controls • Minimize possibility of simulation sickness • Clearly visible content • Simple for developers

Figure 11. User Profile

As shown above in Figure 11, Web developers make up an important segment of the RealWeb intended audience. Since RealWeb VR is developed as a way to bridge the gap into

virtual sites, Web developers should find it easy to create custom content for this application, while also maintaining compatibility for non-VR experiences. Furthermore, developing custom content for RealWeb allows less experienced Web developers to become more familiar with VR. Most importantly, developers using and creating with RealWeb VR in mind will not be forced to learn new languages and frameworks.

Test Plan

In creating a test plan, we wanted to verify that the RealWeb application would be usable piece of software that met acceptable standards of user experience (UX). Conducting UX, functionality, and unit testing allowed us to ensure that this application was free of fatal flaws and design mistakes, while also certifying that the code was performing as expected. The following outlines the test plan, with the testing results being displayed in the appendix.

UX Testing

Since VR web browsers are a new type of product, there have not yet been any specific standards set for user experience. In this case, we will draw upon the current best practices put forth by Oculus[10] in order to test for user comfort and experience in VR. Additionally, testing will evaluate whether users can intuitively operate the browser as intended, while gathering feedback for furthered improvement. User feedback and opinions will be compiled. This testing will be conducted alongside the Functionality Testing below.

Functionality Testing

Functionality testing includes verifying that the VR browser features perform as required. The following list of features will be tested, with the results being recorded in the table below.

[10] https://developer3.oculus.com/documentation/intro-vr/latest/concepts/bp_intro/

#	Description	Input	Expected output	Actual output	Pass/ Fail	Reason for failure/success
1	Browser shows entire web page	Extension turned on; browse to web page	A-Frame scene begins; web page is displayed on plane; browser does not crash			
2	Working hand controls	Controllers turned on	Controller models display in scene; 1 to 1 movement with user hands; Controller models placed in correct position			
3	Scene displays in VR headset	Extension turned on; browse to web page; A-Frame begins	User can put on HMD and view scene with reasonable performance			
4	VR browser works with any website	Browse to any website	Website displays correctly on plane; no artifacts such as rogue elements			
5	Browser UI buttons	User clicks back, forward, or URL bar	Browser navigates back or forward			
6	3D elements	Browser reads CSS input that indicates VR element positioning	3D elements are positioned and scaled correctly			
7	Virtual Keyboard input	User clicks input box	Keyboard appears; user can type input			

Figure 12. Functionality Test

Unit Testing

Unit testing for RealWeb VR will be carried out in a similar manner to other JavaScript based applications. We will be modeling our testing framework from the A-Frame project by using Mocha for our unit testing and Chai as our assertion library, as well as Sinon for any stubbing and Karma for configuration. As RealWeb VR is a chrome extension that does not contain any background html or popup, our UI testing will consist of asserting proper entities are created within the scene. Therefore, once we have a feature complete solution, we will be converting all our logic to A-Frame components to leverage the A-Frame testing paradigm.

With how our code is structured today, there are methods and logic written in ways that can be improved upon and abstracted. As our project becomes more stable and features are written, we will be making passes to improve the modularity of our code and write tests wherever they can be written. There are, however, many dependencies which we had to modify to meet our goals. One such example is the public HTMLShader component for A-Frame. Since the HTMLShader component does not have any existing tests, it is difficult for us to make changes to the component and properly test them without writing tests that did not originally exist.

The table below illustrates an abstraction of what logic can and will be tested moving forward. This logic includes such things as dependency injection, scene creation, event listener creation (For controllers and elements), interactive element creation, etc.

File	Method	Description
realWebMain.js	render3dLinks	**Given**: A set of interactive elements that exist in the DOM **When**: An applicable element is found. **Then**: Assert a new entity is created for that element.

realWebMain.js	createNewEntities	**Given**: A set of custom CSS rules applied to elements on the DOM. **When**: An element meets that CSS rules. **Then**: Assert a new entity is created for that element.
realWebMain.js	renderElementsFromCss	**Given**: A custom CSS file provided by a developer. **When**: The method parses the custom selectors. **Then**: createNewEntities method is called with parsed selectors
dependencyInjector.js	addScript	**Given:** a relative path to a script. **When:** addScript is called. **Then:** Assert the proper script element has been created in the DOM.
realWebMain.js	createEventProxies	**Given:** All interactive entities have been created in the DOM. **When:** Method is called. **Then:** Assert event listeners have been created for each interactive element.
SceneInjector.js	injectSceneTemplate	**Given:** A reference to the scene html template **When:** The page has finished loading. **Then:** The scene has been injected into the DOM.
realWebViveControls.js	createCursorEventListener	**Given:** A vive controller element exists in the DOM. **When:** createViveListeners is called **Then:** Assert that the vive controller element has proper listeners created
realWebViveControls.js	createPlaneMovementEvent	**Given**: The Vive controls component has initialized. **When**: createPlaneMovementEvent is called.

		Then: Assert that the proper event observables are created.
RealWebViveControls.js	createScaleEvent	**Given**: The Vive controls component has initialized. **When**: createScaleEvent is called. **Then**: Assert that the proper event observables are created.

Figure 13. Unit Testing.

Future Recommendations/Recommendations for Improvement

There are many features that stand to be added in the future. At the time of this writing, the application can create a 3D scene that is VR compatible, enable rudimentary VR controls, and display entire Web pages. 3D elements allow users to view layered 3D web pages. In the future, the browser user interface could be improved with a fleshed-out menu system. Tabbed browsing, while not currently feasible due to performance issues, could potentially one day bring the ability to view pages in multiple windows at once. For developers, a web design view could allow the easy creation of VR web pages by simply placing elements in a page by hand.

Long term, RealWeb VR could benefit from hand tracking technologies such as Leap Motion, a VR accessory that tracks hand and finger movements within a virtual space. Leap Motion would enable an even easier user experience since controllers would not be needed. All user input could come from the user's own hands and fingers.

Hand tracking would also help RealWeb cross the line into the augmented reality (AR) space. AR, which inserts virtual objects into a real world space, would expand the RealWeb market beyond VR and could arguably be even more effective as a medium for a virtual Web browser.

Conclusion

Over the course of this project, this team has overcome a number of challenges in creating an application for which there was no precedent. In order to achieve the goal of creating a VR Web browser that retains the familiarity and ease of use of traditional browsers, while innovating and improving on them, we navigated (and continue to navigate) through a slew of obstacles. Creating something for which we had no example meant that we had to think deeply about what a VR Web browser would look like and could do. It also meant that we were forced to discover and rediscover ways to implement the ideas and features that we thought were essential to an evolved browsing experience.

We believe that this project has potential to be a usable browser for VR enthusiasts everywhere. With additional work and help, perhaps from the open source community, we even anticipate that one day this could be a high performance browser with the ability to transform browsing and web pages, both in design and efficiency.

Bibliography

Barfield, Woodrow, and Thomas A. Furness. *Virtual Environments and Advanced Interface Design*. New York: Oxford University Press, 1995.

Döpfner, Mathias, and Die Welt. "Mark Zuckerberg Talks about the Future of Facebook, Virtual Reality and Artificial Intelligence." Business Insider. February 28, 2016. Accessed October 31, 2016. http://www.businessinsider.com/mark-zuckerberg-interview-with-axel-springer-ceo-mathias-doepfner-2016-2?op=1.

Gaudiosi, John, "Over 200 Million VR Headsets to Be Sold by 2020", Fortune, January 21, 2016, Accessed November 02, 2016, http://fortune.com/2016/01/21/200-million-vr-headsets-2020/.

Martindale, Jon. "Fantastic Contraption Dev Says Oculus Touch and HTC Vive Controllers 'Almost Identical'", Digital Trends, April 14, 2016, Accessed March 18, 2017, http://www.digitaltrends.com/computing/fantastic-contraption-dev-says-oculus-touch-and-htc-vive-controllers-almost-identical/

"Simulator Sickness", Oculus, Accessed October 30, 2016, https://developer3.oculus.com/documentation/intro-vr/latest/concepts/bp_app_simulator_sickness/.

"WebVR API", Mozilla Developer Network, Accessed November 02, 2016, https://developer.mozilla.org/en-US/docs/Web/API/WebVR_API.

"Worldwide Revenues for Augmented and Virtual Reality Forecast to Reach $162 Billion in 2020, According to IDC", idc.com, August 15, 2016, Accessed November 02, 2016, https://www.idc.com/getdoc.jsp?containerId=prUS41676216.

Appendix

- **UX Testing**

As mentioned in the test plan section, we drew our UX testing standards from the Oculus best practices guide. Per this guide, the most important UX factor when designing for VR is reducing or, ideally, eliminating simulator sickness. Our UX testing, which included over a dozen users, yielded no reports of simulator sickness at all. This is likely due to at least two factors:

1. Our application runs at a minimum framerate required to reduce the chance of sim sickness.
2. Our application is a stationary experience.

In addition to testing for simulator sickness, we gathered feedback for our user interface and control scheme. Feedback for the overall look and feel of the application was very positive, with the overall experience rating high. Users were, for the most part, able to navigate and user the controls with relative ease once they were explained. However, there were some occasions where user difficulty showed that some polishing still needs to be done to improve the user experience. Users occasionally had trouble determining what was a clickable link and what was not. Also, one user did not realize he had to first clear the keyboard before beginning to type.

- **Functionality Testing**

#	Description	Input	Expected output	Actual output	Pass/ Fail	Reason for failure
1	Browser shows entire web page	Extension turned on; browse to web page	A-Frame scene begins; web page is displayed on	A-Frame scene begins; web page is displayed on	Pass	

			plane; browser does not crash	plane; browser does not crash		
2	Working hand controls	Controllers turned on	Controller models display in scene; 1 to 1 movement with user hands; Controller models placed in correct position	Controller models display in scene; 1 to 1 movement with user hands; Controller models placed in correct position	Pass	
3	Scene displays in VR headset	Extension turned on; browse to web page; A-Frame begins	User can put on HMD and view scene with reasonable performance	User can put on HMD and view scene with reasonable performance	Pass	
4	VR browser works with any website	Browse to any website	Website displays correctly on plane; no artifacts such as rogue elements	Some websites display correctly on plane; Many websites contain missing or disfigured elements	Fail	Some websites do not follow conventions or have complex construction that is not compatible with the current iteration of RealWeb VR
5	Browser UI buttons	User clicks back, forward, or URL bar	Browser navigates back or forward	Browser navigates back or forward; however, users cannot enter VR mode automatically	Pass	Due to security features added to Chromium later in development, the browser requires a manual input in order to enter full screen VR mode
6	3D elements	Browser reads CSS input that indicates VR element positioning	3D elements are positioned and scaled correctly	3D elements are positioned and scaled correctly	Pass	

7	Virtual Keyboard input	User clicks input box	Keyboard appears; user can type input	Keyboard exists for URL bar, but does not work for input boxes	Fail	Making input boxes functional would require additional time

- **Unit Testing**

Before we could begin any unit testing for RealWeb VR, we needed to consider how the technologies used within our project interfaced with one another. Specifically, the combination of using A-Frame within a chrome extension placed our project in uncharted territory in terms of proper unit testing. In addition, our lack of experience with both A-Frame and Chrome extensions required us to revise our code and logic structure multiple times throughout the project lifespan. However, with the experience gathered throughout the development of RealWeb VR, we have gained enough knowledge to determine the best way to move forward with conducting our unit testing.

Throughout both semesters, our code was divided up into several individual scripts that were injected into each page. These scripts primarily interacted with the injected A-Frame scene to properly wire up controls, create floating elements, and manage updating the page and elements within the scene. However, by dividing our logic into several scripts we prevented ourselves from being able to write tests that would make all necessary assertions on state changes within A-Frame and our scene. The answer to this problem was a complete re-factor of all our work into individual A-Frame components that we could simply place directly within the Scene.

Although we were not aware of it at the time, all the logic we were writing for RealWeb VR should have been done on a per component basis. Not only does this keep our code-base clean, but it allows all our logic to exist within the A-Frame domain. By having our logic exist as

components within A-Frame, we can begin to follow the proper component testing paradigms set by the A-Frame development team. As of the writing of this report, we have recently converted most of our logic into individual components and are near completion with setting up our testing framework. However, due to time constraints, unit testing will need to be carried out later.

Browser Security

A brief note should be made about browser security since the transfer of sensitive information often occurs within browsers. While the RealWeb VR application is fully capable of browsing the web, in its current form it is not capable of receiving input from input boxes; therefore, sensitive data like passwords, credit cards, and other personal information could not be received. In its current form, RealWeb is a browser of static content. If RealWeb eventually allows input of data, security practices would need to be enabled.

Additionally, since RealWeb VR is a browser extension, there could be concern that harvesting personal data could happen and be sold or used for nefarious purposes. Again, RealWeb does not have the capability of receiving input data; furthermore, since RealWeb will likely be released as an open source application, anyone who uses the application could verify from the source code that no such activity is taking place. Since this application is primarily a prototype application that serves to show the potential of VR in Web browsing, browser security will be an emphasis in future iterations.