VR in the Web Browser

Group 4: Aleksandar Kojic, Milos Kojic, Michaela Kargl-Schrammel, and Tu Ha Anh

706.041 Information Architecture and Web Usability WS 2016
Graz University of Technology

05 Dec 2016

Abstract

Browser-based virtual reality (VR) is a highly dynamic and fast evolving area. This survey explores the current state of the art regarding browser-based VR. It gives an overview of different types of VR head-mounted display (HMD) devices, explains the role of WebVR for the creation of browser-based VR experiences, and describes the functionality and usage of WebVR frameworks in detail. WebVR frameworks are very useful as they can improve the development speed and support non-technical persons with creating VR worlds. In this survey both Graphical User Interface (GUI)-based and Code-based frameworks are covered. Two GUI-based frameworks are described: A-frame, which is currently the most popular framework, and Vizor, which uses visual programming techniques and provides a very rich editor. Furthermore, the survey briefly explains also three examples of code-based WebVR frameworks: GLAM, WebVR Markup, and Primrose. GLAM and WebVR Markup both work in a similar way, the main difference between these two is that WebVR Markup is designed to handle complex objects. Primrose uses another approach, as it is completely JavaScript based. In addition to describing the creation of browser-based VR with WebVR frameworks, this survey also shows by the example of Unity how game engines can be utilised to generate browser-based VR.

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

This survey paper presents an overview of the state of the art in the field of browser-based virtual reality. Chapter 1 briefly explains what is meant by virtual reality (VR) and gives an overview of current VR devices. Chapter 2 describes WebVR and its role in providing virtual reality in the browser. Chapter 3 gives an overview of WebVR frameworks. Chapter 4 explains how the game engine Unity can be utilized to create virtual reality that can be experienced in the browser, and Chapter 5 provides a brief outlook to forthcoming developments in the field of browser-based virtual reality.

1.1 What is Virtual Reality (VR) and How Does it Work

In their book "Virtual Reality Technology and Applications", Mihelj et al. [2014] define virtual reality as follows:

> "Virtual reality is composed of an interactive computer simulation, which senses the user's state and operation and replaces ... sensory feedback information to one or more senses in a way that the user gets a sense of being immersed in the simulation (virtual environment)."

One possibility to realise VR is browser-based VR. In browser-based VR, a virtual environment is presented to the user directly in the web browser. Usually this is done via a head-mounted display (HMD). As described by Charara [2016], for this set-up three elements are required: a computer or Smartphone to run the VR application in the browser, a VR headset placing a display in front of the user's eyes, and some kind of input, such as for example head, eye or hand tracking, so that the user can interact with the virtual environment.

In order to provide the user with an effective VR feeling, the VR headset must have a large field of view (current high-end headsets have a 100-110 degree field of view), a frame rate of minimum 60 frames per second, and low latency head-tracking with a maximum lag of 50 ms. [Charara, 2016]

For the creation of an immersive VR experience the following two aspects are essential and thus featured by all VR devices:

- *Stereoscopic 3D images give the illusion of a virtual scene*: The screen of the VR headset displays a separate view of the virtual environment for the user's left and right eye, and lenses placed between the user's eyes and the screen support the creation of a stereoscopic 3D image.
- *The virtual scene is aligned with the head movements of the user*: The user's head motion is tracked and the displayed stereoscopic 3D-graphic of the virtual world on the screen of the VR HMD is adjusted accordingly to follow the head movements of the user.

In addition to these two basic functionalities, which are featured by all VR devices, some VR devices provide also advanced features such as eye tracking or motion tracking, in order to achieve further improved VR with virtual environments responding to the user's line of sight and visual focus as well as to the user's body movements.

Figure 1.1: Oculus Rift Developer Kit Headset
[By Ats Kurvet - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=35919898]

Figure 1.2: HTC Vive Headset
[By Maurizio Pesce - https://www.flickr.com/photos/pestoverde/17135543951, CC BY 2.0, https://commons.wikimedia.org/w/index.php?curid=39893639]

1.2 Devices for Virtual Reality

In the last two years several HMD VR devices were brought to the market. These HMD VR devices can broadly be categorised into two groups:

- VR headsets, which operate with a Smartphone
- VR headsets, which operate with a PC

The following two subsections describe these two types of VR headsets in more detail, and Table 1.1 gives an overview of advantages and disadvantages of these two types of VR headsets.

1.2.1 VR Headsets Operating with a PC

VR headsets of this type display the virtual environment on a built-in screen and are connected via cable to a computer running the VR video. These VR headsets come with external positional tracking sensors, which measure the movements and position of the user. For the operation of these VR headsets a powerful "gaming" PC is essential. The recommended set-up includes an Intel i5-4590 processor equivalent or better, and NVIDIA GTX 970 / AMD R9 290 graphics card equivalent or better. [OculusVR, 2016; HTC-Corporation, 2016; FOVE, 2016] Examples of this type of VR headset include Oculus Rift (as shown in Figure 1.1), and HTC Vive (as shown in Figure 1.2). Another example for this type of VR headset is the upcoming Fove VR headset , which in addition to head- and motion-tracking will offer also eye-tracking. [FOVE, 2016] According to FOVE [2016] shipping of the first FOVE VR devices is planned for January 2017.

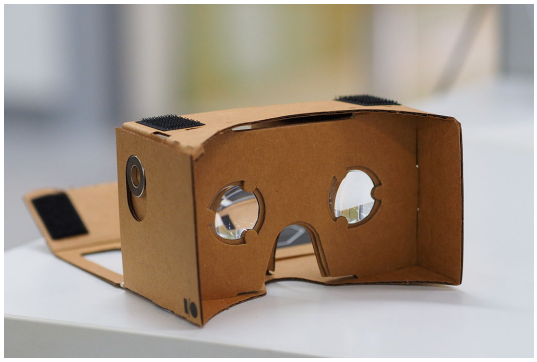

Figure 1.3: Google Cardboard VR headset works with Android and iOS Smartphones.
[By othree / Wikimedia Commons / CC BY 2.0 https://commons.wikimedia.org/wiki/File:Assembled_Google_Cardboard_VR_mount.jpg]

Figure 1.4: Google Daydream View comes with a dedicated controller and works with Google's Pixel and Pixel XL, as well as Motorola's Moto Z and Moto Z Droid Smartphones. [By Maurizio Pesce from Milan, Italia – Daydream View VR Headset Made By Google, CC BY 2.0, https://www.flickr.com/photos/pestoverde/30155280475]

1.2.2 VR Headsets Operating with a Smartphone

For using this type of VR headsets a Smartphone must be slotted into the HMD VR device. These VR headsets do not need a connection to a computer, since the VR video is running on the Smartphone and is displayed directly on the Smartphone's screen. For head tracking the internal sensors of the Smartphone (such as gyroscope, accelerometer and magnetometer) are utilised, and some of these headsets provide also additional built-in sensors for head-tracking. These VR headsets do not track movements of the user. Examples of this type of VR headsets, which work with a Smartphone, include Google Cardboard (as shown in Figure 1.3), Google Daydream (as shown in Figure 1.4), Samsung Gear VR (as shown in Figure 1.5), and Zeiss VR One, to name just a few.

Figure 1.5: Samsung Gear VR works with a Samsung Galaxy S6 / S6 Edge, S6Edge+ or Galaxy Note 5 Smartphone. [By Maurizio Pesce from Milan, Italia - Samsung Gear VR, CC BY 2.0, https://commons.wikimedia.org/w/index.php?curid=37143637]

Type of VR Headset	Advantages	Disadvantages
VR Headsets operating with Smartphone	• Comparatively cheap • Easy and fast setup • Location-independent • Free movement possible (no cable connection to PC)	• Quality of head-tracking is not very good • Some VR headsets of this type work only with specific Smartphone models • Field of Vision depends on size of Smartphone
VR Headsets operating with PC	• Large Field of Vision • Head- and motion-tracking • Strong feeling of immersion	• Expensive • High performance "gaming" PC with specific (expensive) graphics card and powerful processor needed • User is "tied" to PC with cable • Comparatively large space is needed

Table 1.1: Advantages and Disadvantages of Different Types of VR Headsets

Chapter 2

WebVR

WebVR is a JavaScript API which works as an input and output handler and enables the possibility of bringing Virtual Reality to web browsers with connected VR devices. In the year 2014 the Mozilla Foundation and the Google Chrome-Team started developing the first version, which then was released in 2016 [Team, 2016]. Since WebVR is still in the development phase the API is only available on Firefox Nightly, Chromium and the Samsung Internet Browser for Gear VR.

In the past years, the relevance of Virtual Reality and its devices increased. Many business models and different areas of usage arose. Because of the dissemination of VR the compatibility became more important and so the demand for support from browser vendors also increased [Yee, 2016]. With the release of the version 1.0 the first functions for the WebVR API were included. Features like motion control, desktop and mobile suitability and the handling of VR-specific device rendering and display were significant parts of the new interface [Yee, 2016]. Despite the efforts of finding a common ground it is still challenging to create a consistent API which fits with all requirements needed for all different VR devices. Each VR device has own functionalities which need to be included in the API. For example, the Oculus Rift has several integrated sensors. Whereas, Google Cardboard have no integrated sensors or even a screen.

2.1 WebVR API

For creating a VR experience three main components are necessary: The VR display, the orientation and positioning data of the headset inside of the area and the field of view which is defined by the eye parameters [Yee, 2016]. In the following the basic functionality of the API will be shown and explained.

2.1.1 Device integration

In Listing 2.1 the navigator.getVRDisplays() method shown above requests for a browser compatible VR display. It checks possible devices for their device IDs, device names and sensor IDs. Afterwards, the available vrDisplay is then listed in an array. If no VR device is reachable an error occurs. In case a google cardboard is used instead of a head-mounted display the VR display can be simulated.

2.1.2 Measuring of the eyeParameter

After the availability of an VR device has been checked the size of the render target needs to be set. Therefore, the eyeParameter of both eyes will be measured and the getEyeParameters() returns the data of each eye like it is shown in Listing 2.2. The eyeParameter can change from time to time due to external factors [Vukicevic et al., 2016].

```
navigator.getVRDisplays().then (function (display) {
if  (!display.length) {
       return;
    }
  vrDisplay = display.length[0];
}).catch (function (err)       {
   console.error ('Could not get VRDisplay' , err.stack);
});
```

Listing 2.1: Device integration with WebVR API [Yee, 2016]

```
var  eyeParameter = vrDisplay.getEyeParameters (whicheye); //left or right

var width = eyeParameter.renderWidth;
var height = eyeParameter.renderHeight;
```

Listing 2.2: Measuring the eyeParameter to set the render target [Yee, 2016]

2.1.3 WebGL canvas

To bring the content into the VR device the Web Graphics Library (WebGL) is used as a parameter for the VRDisplay.requestPresent() method. The WebGL <canvas> element represents the viewing surface and a confirmation request is send to the user so he can confirm that he wants to change into the VR mode. Furthermore, the method vrDisplay.exitPresent() enables the possibility of quitting the VR mode as shown in Listing 2.3 [Yee, 2016].

2.1.4 Rendering Loop

After all parameters are set and the content as well as the right view is adjusted a device-specific requestAnimationFrame must be defined.

The function onAnimationFrame() is rendering one single frame for VR data and vrDisplay.requestAnimationFrame (onAnimationFrame) is requesting and scheduling the next frame. This callback basically shows a rendering loop. The attribute cancelRequestAnimationFrame() returns to the status when the VRDisplay is not presenting [Vukicevic et al., 2016]. See more in Listing 2.4.

2.1.5 Position Tracking

In the next step the orientation and the position of the VR device need to be tracked. Therefore, the method VRDisplay.getPose() gathers the position of the headset as shown in Listing 2.5. The attribute position is defined by three coordinates. The x-coordinate is gathering if the user is looking right or left, the y-coordinate captures the up and down movement and the z-coordinate displays what is behind and in front of the user. The orientation attribute show the rotation activities around the y-axis. If no coordinates are trackable the method returns null [Vukicevic et al., 2016].

```
var webglCanvas = document.querySelector ('#webglcanvas');
var enterVRBtn = document.querySelector ('#entervr');
    enterVRBtn.addEventListener ('click' , function () {
        vr.Display.requestPresent ({source: webglCanvas});
    });
vrDisplay.exitPresent();
```

Listing 2.3: Bringing the content to the VR device with WebGL canvas [Yee, 2016]

```
var id = vrDisplay.requestAnimationFrame (onAnimationFrame);
function onAnimationFrame () {
    id = vrDisplay.requestAnimationFrame (onAnimationFrame);
}
vrDisplay.cancelRequestAnimationFrame (id);
```

Listing 2.4: Rendering AnimationFrame with rendering loop [Yee, 2016]

```
var Pose = vrDisplay.getPose ();
var orientation = pose.orientation;
var position = pose.position;
```

Listing 2.5: Tracking the orientation and the position of the VR device [Yee, 2016]

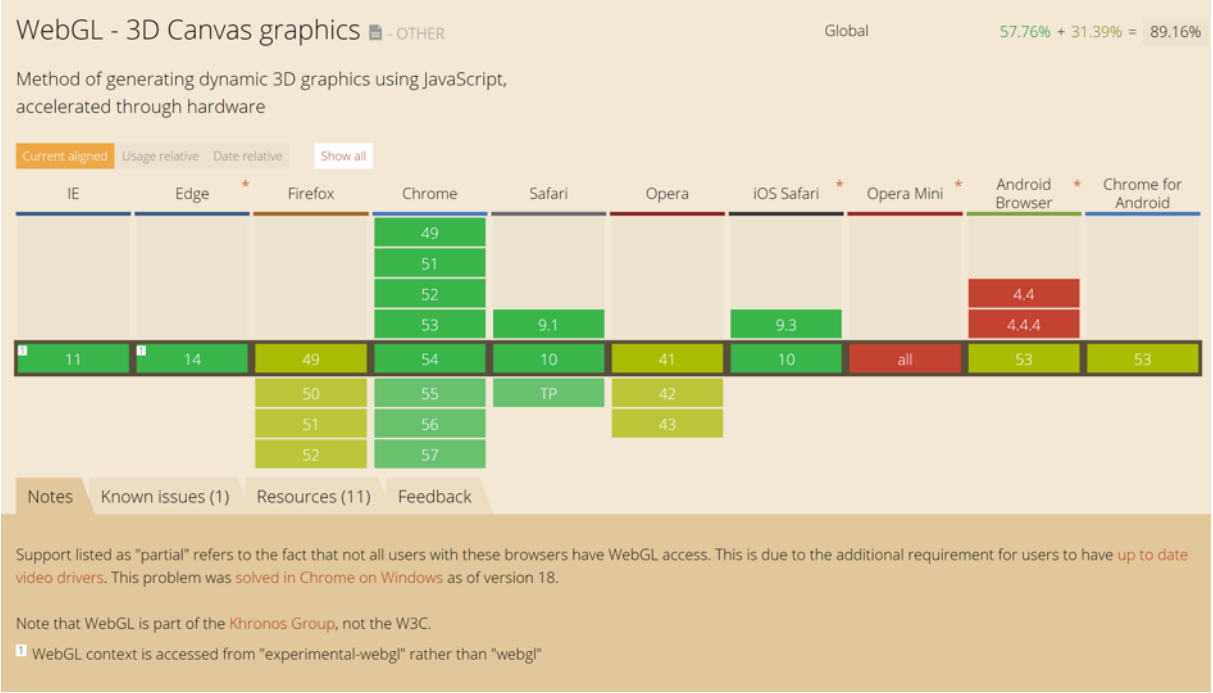

Figure 2.1: Compatibility of WebGL [Group, 2016]

2.1.6 The Field of View and scene projection

To finally project the scene to the VR device the offset data of the eye parameters and the field of view (FOV) are necessary [Yee, 2016]. The offset attribute consists of a three-component vector which describes the offset of the eyes. The FOV should cover the complete area "of the viewable frustum of the eye" [Vukicevic et al., 2016].

The code in Listing 2.6 shows that first the eyeParameters are called. Afterwards, the eyeOffset is set and the projection matrix is created. The function function makeProjectionMatrix() finally applies the eyeMatrix to the view of the user and sets the x- and y-scale [Yee, 2016].

2.2 WebVR with WebGL

The Web Graphics Library (WebGL) is a rendering API which allows web browsers to render interactive 3D and 2D objects without any additional plugins. WebGL is JavaScript based and derived from OpenGL ES 2.0 [Jackson, 2014].

WebGL is completely integrated into the existing web standards and uses the HTML canvas element. Furthermore, WebGL can be used with other HTML elements. Due to the high flexibility of WebGL it is available in nearly every browser.

Due to the similarity of WebGL to 3D and VR development it supports a quick integration of content and engines. For the integration of the API several WebVR elements are necessary such as the Field of View, the eye parameters and the sensor data of the position tracking [Vukicevic, 2014]. Besides, WebGL enables an engine integration. Therefore, especially gaming projects are interested in integrating Unity or the Unreal Engine.

2.3 WebVR with HTML/CSS

HTML and CSS are the most widespread standards in the web. Therefore, the compatibility between VR and HTML/CSS is an important aspect for the future development of VR in web browsers. The basis for

```
var eyeParameters = vrDisplay.getEyeParameters('left');

var eyeOffset = eyeParameters.offset;

var eyeMatrix = makeProjectionMatrix(vrDisplay, eyeParameters);

function makeProjectionMatrix (display, eye) {
  var d2r = Math.PI / 180.0;
  var upTan = Math.tan(eye.fieldOfView.upDegrees * d2r);
  var downTan = Math.tan(eye.fieldOfView.leftDegrees * d2r);
  var rightTan = Math.tan(eye.fieldOfView.rightDegrees * d2r);
  var leftTan = Math.tan(eye.fieldOfView.leftDegrees * d2r);
  var xScale = 2.0 / (leftTan + rightTan);
  var yScale = 2.0 / (upTan + downTan);

  var out = new Float32Array(16);
  out[0] = xScale;
  out[1] = 0.0;
  out[2] = 0.0;
  out[3] = 0.0;

  out[4] = 0.0;
  out[5] = yScale;
  out[6] = 0.0;
  out[7] = 0.0;

  out[8] = -((leftTan - rightTan) * xScale * 0.5);
  out[9] = (upTan - downTan) * yScale * 0.5;
  out[10] = -(display.depthNear + display.depthFar) / (display.depthFar- display.
      depthNear);

  out[12] = 0.0;
  out[13] = 0.0;
  out[14] = -(2.0 * display.depthFar * display.depthNear) /(display.depthFar - display.
      depthNear);
  out[15] = 0.0;

  return out;
}
```

Listing 2.6: Projecting the rendered scene to the VR device using the eyeParameters [Yee, 2016]

```
1 var state = hmdSensor.getState();
2 camera.style.transform = stateToCSSTransform(state);
```

Listing 2.7: Creating a transformation matrix with the position and orientation parameters [Vukicevic, 2014]

```
#camera {
    transform: vr-orientation() vr-position();
}
```

Listing 2.8: Setting CSS properties [Vukicevic, 2014]

integrating VR in the web with HTML and CSS are the various features modern CSS offers. Features such as 3D transforms, gradients and animations can help to bring content into the VR environment. Furthermore, the use of HTML/CSS are the foundation for applying responsive design. The WebVR API creates its own virtual environment without using perspective properties of CSS. The Field of View defined by the API will set the parameters for the virtual space [Vukicevic, 2014].

In the current development state of WebVR and HTML/CSS a transformed element serves as a "camera" and the position and orientation parameters create a transformation matrix as shown in Listing 2.7. Therefore, it is still a mixture of JavaScript and CSS [Vukicevic, 2014]. For the future integration of WebVR with CSS it will be possible to set different CSS properties like in Listing 2.8.

The vr-orientation and vr-position will be set by the attribute transform. This implementation style will keep the code easier to read and the performance will be higher. Listing 2.9 show that responsive design can be applied through the integration of CSS Media Queries. The WebVR rendering with CSS is still in the development phase and only available for Firefox. The next step will be the integration of CSS properties to facilitate the CSS rendering process [Vukicevic, 2014].

```
@media vr
{
    #camera {
        transform: vr-orientation() vr-position();
    }
    #contentArea {
        width: 100cm;
        height: 80cm;
        transform: translateZ(50cm);
    }
}
```

Listing 2.9: Media Queries enable responsive design [Vukicevic, 2014]

Chapter 3

WebVR Frameworks

Even though WebVR is a pretty new technology, there are many WebVR frameworks which one can use to create 3D worlds and VR environments.

There are two classifications of the frameworks:

- Graphic User Interface (GUI)-based WebVR frameworks
- Code-based WebVR frameworks

3.1 Graphic User Interface (GUI)-based frameworks

There are multiple benefits of using Graphic User Interface based frameworks.

Some of the benefits are:

- Less time spent in development process.
- Easier development.
- Programmers do not have to be familiar with how lower layers of technology work.

3.1.1 Examples

WebVR GUI-based frameworks are used not only by developers, who usually use GUI to test the capabilities of the framework before they dive into it, but also by non-programmers as they do not need any technical knowledge to create their own VR environments. In the following two examples of GUI-based frameworks are described in more detail:

- A-frame
- Vizor

3.1.2 A-frame

A-frame is the most widely used framework for creating virtual reality environments. It was developed by Mozilla VR team. A-frame is based on Three.js and converts JavaScript entities to DOM. Even though there is a GUI editor available on the A-frame website it is not necessary to use it. One is able to create the entire VR environment through code as well. The Editor provides users with an option to export their scenes as code. Creating content in A-frame is very easy as users are provided with a rich and very intuitive editor.

Some examples of the possibilities the editor provides for handling and modifying a single element:

- **transform:** Change the position of an element in the scene

- **texture:** Add either color or texture image to an element
- **animate:** Define starting and ending points, easing effect and duration
- **interaction:** Add clicking or gazing functionality
- **lighting:** Add both ambient or directional light

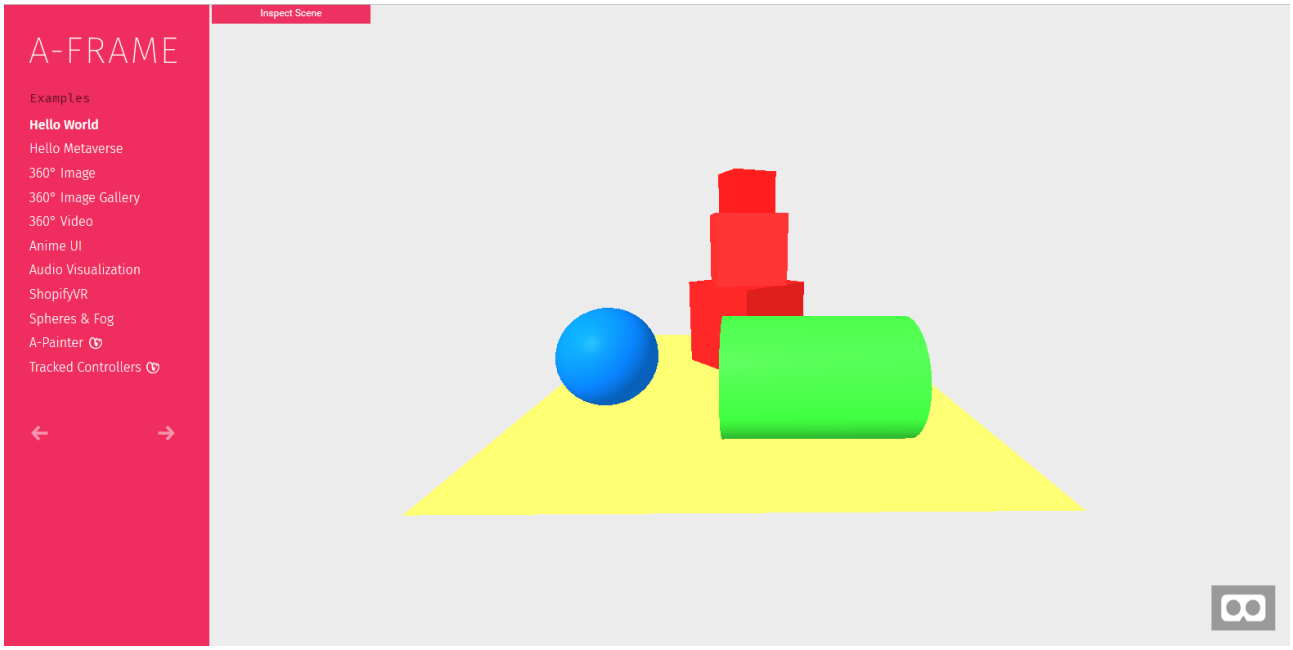

Figure 3.1: An example of A-frame [team, 2016]. [Screenshot taken by the authors of this survey.]

By using A-frame through code one can go beyond the boundaries of the editor and create more complex environments. The structure of an element is very intuitively designed and developers, who are familiar with HTML, will find it easy to use. Basically, everything is in regular HTML and there are only a few A-frame specific tags that are used to ease and speed up the development process. First of all, there is a tag which wraps up all the element tags on the scene. It is called *<a-scene>* tag and all the elements are nested inside it. Then, there are several tags for setting up the scene like:

- Sky tag: Adds sky to the scene with specific color (Code: *<a-sky color="#ECECEC"></a-sky>*)
- Camera tag: Sets up the camera view (Code: *<a-entity camera=""></a-entity>*)

And finally, the element tags:

- Box: This box element is rotated by 45 degrees and is coloured in blue (Code: *<a-box position="-1 0.5 1" rotation="0 45 0" color="#4CC3D9"></a-box>*)
- Sphere: An example of sphere with 25% of increment in radius (Code: *<a-sphere position="0 1.25 -1" radius="1.25" color="#EF2D5E" ></a-sphere>*)
- Cylinder: In this example radius and height are changed and it is coloured in yellow (Code: *<a-cylinder position="1 0.75 1" radius="0.5" height="1.5" color="#FFC65D"></a-cylinder>*)

```
<html xmlns="http://www.w3.org/1999/xhtml"><head>
    <meta charset="utf-8" />
    <title>Hello, World! - A-Frame</title>
    <meta name="description" content="Hello, World! - A-Frame" />
    <script src="https://aframe.io/releases/0.3.2/aframe.min.js"></script>
</head>
  <body>
    <a-scene class="fullscreen" canvas="" inspector=""
      keyboard-shortcuts="" screenshot="" vr-mode-ui="">

      <a-sphere position="0 1.25 -1" radius="1.25" color="\#EF2D5E"
          material="" geometry="" rotation="" scale="" visible="">
        </a-sphere>

        <a-box position="-1 0.5 1" rotation="0 45 0" width="1"
          height="1" depth="1" color="\#4CC3D9"
          material="color:\#4CC3D9"
          geometry="primitive:box;width:1;height:1;depth:1" scale="1 1 1"
          visible="true">
        </a-box>

        <a-box position="-1 1.4 1" rotation="0 10 0" width="1"
          height="1" depth="1" color="\#4CC3D9"
          material="color:\#4CC3D9"
          geometry="primitive:box;width:1;height:1;depth:1"
          scale="0.8 0.8 1" visible="true">
        </a-box>

        <a-box position="-1 1.4 1" rotation="0 10 0" width="1"
          height="1" depth="1" color="\#4CC3D9"
          material="color:\#4CC3D9"
          geometry="primitive:box;width:1;height:1;depth:1"
          scale="0.8 0.8 1" visible="true">
        </a-box>

        <a-cylinder position="1 0.75 1" radius="0.5" height="1.5"
          color="\#FFC65D" material="" geometry="" rotation=""
          scale="" visible="">
        </a-cylinder>

        <a-plane rotation="-90 0 0" width="4" height="4"
          color="\#7BC8A4" material="" geometry="" position=""
          scale="" visible="">
        </a-plane>

        <a-sky color="\#ECECEC" material="" geometry="" scale=""
          position="" rotation="" visible="">
        </a-sky>

        <a-entity position="0 0 3.8" rotation="" scale="" visible="">
          <a-camera camera="" position="" rotation="" look-controls=""
          wasd-controls="" scale="" visible="">
          </a-camera>
        </a-entity>

        <a-entity camera="" position="" rotation="" scale="" visible="">
        </a-entity>

    </a-scene>
  </body>
</html>
```

Listing 3.1: A-frame full code example

After putting all that together, one can see the code (Listing 3.1) used for creating the environment as seen in Figure 3.1

A-frame supports two types of view experience:

- Flat experience
- VR experience

An example of flat view experience can be seen in Figure 3.1. If there are no VR devices attached, this is the type of view that one gets. This type of view can be very useful in the development phase, as one would not have to put the device on in order to test every feature through the VR device while developing.

Once a VR device is connected, the WebVR API gets device specific information so that it knows how to handle that device and get the input data. Now, users can have VR experience which does not only change the camera view and gets input from VR device, it also can get the input of additional input devices connected to the VR device such as for example Vive controllers. Figure 3.2 shows an example of VR experience on one A-frame demo.

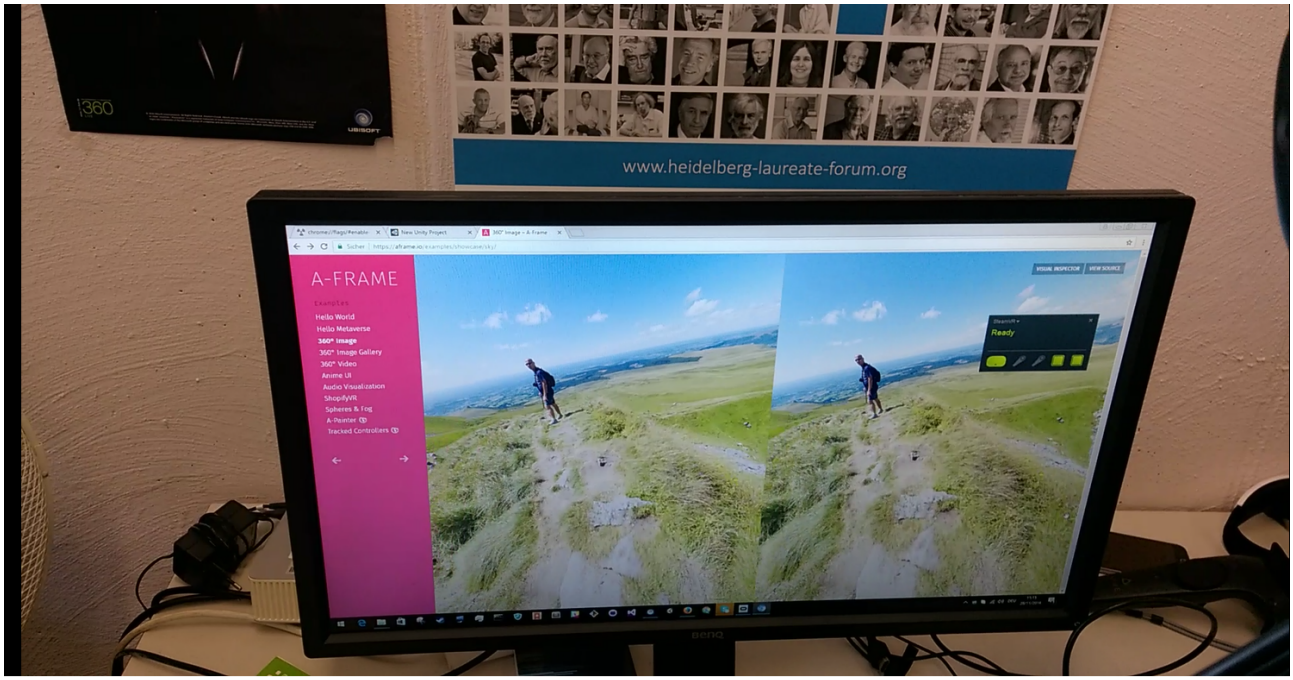

Figure 3.2: An example of A-frame VR experience demo [team, 2016]. [Screenshot taken by the authors of this survey.]

There are two camera renderings, one for each eye, and thus by putting the VR device on, one can have a VR experience.

3.1.3 Vizor

Vizor is another GUI-based WebVR framework. The interesting thing about Vizor is that it uses visual programming techniques for creating a VR world. One component can be composed out of many input parameters and it usually provides an object as the output. Patches are another nice feature in Vizor, as they are predefined components that one can use to start off. There are two types of cameras available in the editor:

- Editor camera

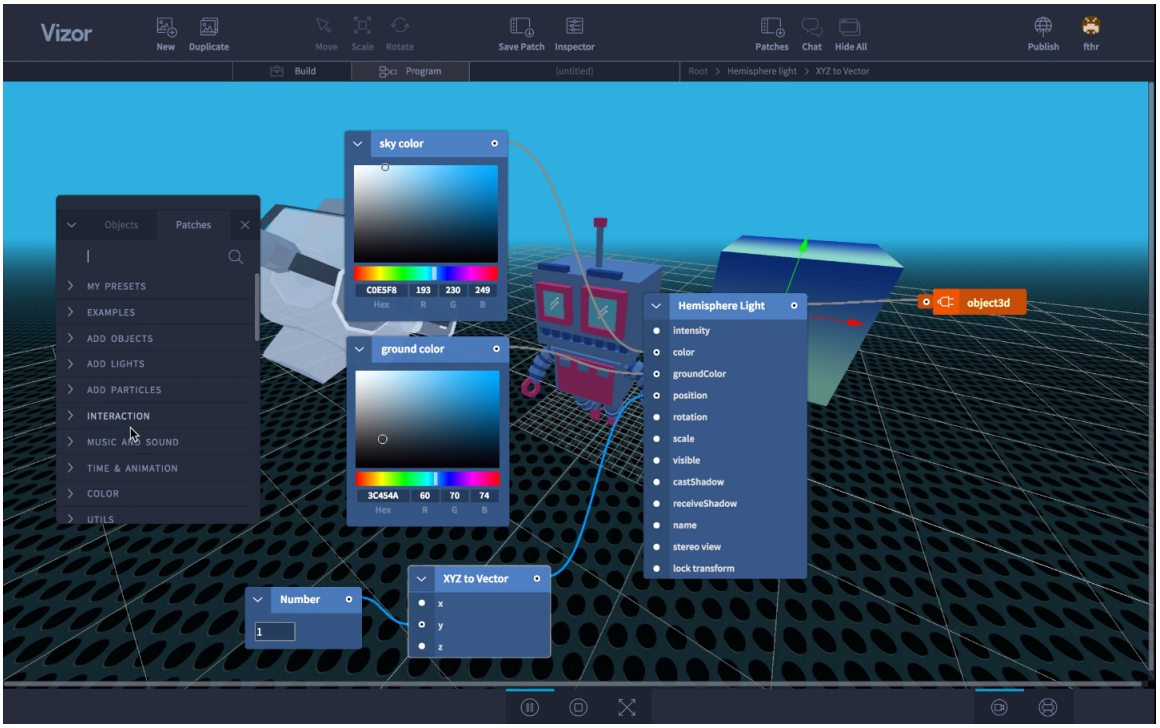

Figure 3.3: Vizor editor [Pixelface, 2016]. [Screenshot taken by the authors of this survey.]

- VR camera

Developers spend most of the time in editor camera view, as in this view they can play around with their scenes, add new objects and modify them. During the development phase, the VR camera is used for testing purposes so that the developer can see what end users are going to get. Here is an example in Vizor (See Figure 3.3).

In this example, there are several inputs attached to the Horizontal light component. In Vizor, there are visual equivalents for most of the regular programming elements, so one can use:

- **Data types** (Integer, Float, Boolean)
- **Regular expressions** (AND, OR, XOR, NOT)
- **Math functions** (Sin, Cos, Log, Min, Max, SQRT)
- **IF statements and LOOPs**

The output of a component is an object, which can be used as an input to some other component.

3.2 Code-based VR frameworks

Code-based frameworks have their advantages as well. There are some specific features which cannot be created with GUI frameworks as they do not provide all functionalities, so there are always some things which have to be done through the code. Another benefit of the code-based approach is that developers can have full control over their work and they can customize every single element of the VR environment.

3.2.1 Examples

Most popular code-based frameworks are:

- GLAM
- Primrose
- WebVR Markup

There are many other code-based frameworks, however in this survey the focus will be on the three most popular ones.

3.2.2 GLAM

The name GLAM is composed of GL (taken from WebGL) and Markup. GLAM was created by Tony Parisi, the author of the book "Learning Virtual Reality" (See [Parisi, 2015]). Tony Parisi is one of the pioneers in Virtual Reality and co-creator of the Virtual Reality Markup Language (VRML), which was published more than 20 years ago. GLAM combines the power of WebGL with a set of easy-to-use markup tags and style properties. This is a basic example (See Listing 3.2 and 3.3).

```
<html>
<head>
  <title>A simple GLAM scene</title>
  <script src="pathtoglam/glam.js"> </script>
</head>
<body>
  <glam>
      <scene>
          <cube></cube>
        </scene>
    </glam>
</body>
</html>
```

Listing 3.2: GLAM Markup code example [Parisi, 2016]

GLAM has a similar structure as A-frame. There is a <*scene*> tag which surrounds all elements, however it is necessary to wrap that tag with the <*glam*> tag. The main difference between GLAM and A-frame is that in GLAM presentation is separated from structure, in other words, GLAM uses a CSS-like approach to define the parameters of an element. In this way, the DOM stays clean of style rules and it is more readable.

GLAM uses CSS animations and CSS transitions but also defines some new CSS rules which can be used to style elements.

3.2.3 Primrose

Primrose is a very popular code based VR framework. Contrary to GLAM, Primrose is completely based on JavaScript as it wraps around a standard Three.js scene. The initial scene provides a ground to walk on, stereo

```
#cube1 {
    animation-duration: 10s;
    animation-name: kfRotateY;
    animation-iteration-count: infinite;
    animation-timing-function:linear;
}
```

Listing 3.3: GLAM CSS example [Parisi, 2016]

```
var geom = box(1, 2, 3),
  mesh = colored(geom, 0xff0000);
put(mesh)
  .on(scene)
  .at(-2, 1, -5);
```

Listing 3.4: Primrose box example code [theory, 2016]

view with a VR head-mounted display, and the ability to walk around with mouse/keyboard, game-pad, and point-and-click teleportation. This is an example of creating a box (See Listing 3.4).

And this is how it looks like on the scene (See Figure 3.4).

Primrose provides many events that can help developers with creating advanced environments (See Listing 3.5).

3.2.4 WebVR Markup

WebVR Markup works similarly as GLAM, however, the main difference is that with WebVR Markup one can define components which consist of sub items. That is a nice and useful feature when creating very complex objects. This is an example of a room (See Listing 3.6).

There are 5 wall items which build up the room object.

WebVR Markup provides a nice code playground so that developers can see the effect of their code while they work on it (See 3.5).

3.3 WebVR Structure and Hierarchy in Respect to the Frameworks

WebGL is a ground technology for creating 3D worlds. However, since it is not that easy to use it, there are many 3D libraries which help developers to speed up the process of development and ease the use of some WebGL functionalities. Examples of such 3d libraries are Three.js and Babylon.js.

It seems that the Three.js library is more popular among WebVR developers. For example, A-frame, which is considered as a most popular VR framework, is based on Three.js. Many framework creators use A-frame as a grounding point for their own solutions. Thus, there are some frameworks like CreateVR which are built on A-frame.

Figure 3.6 presents a WebVR hierarchy.

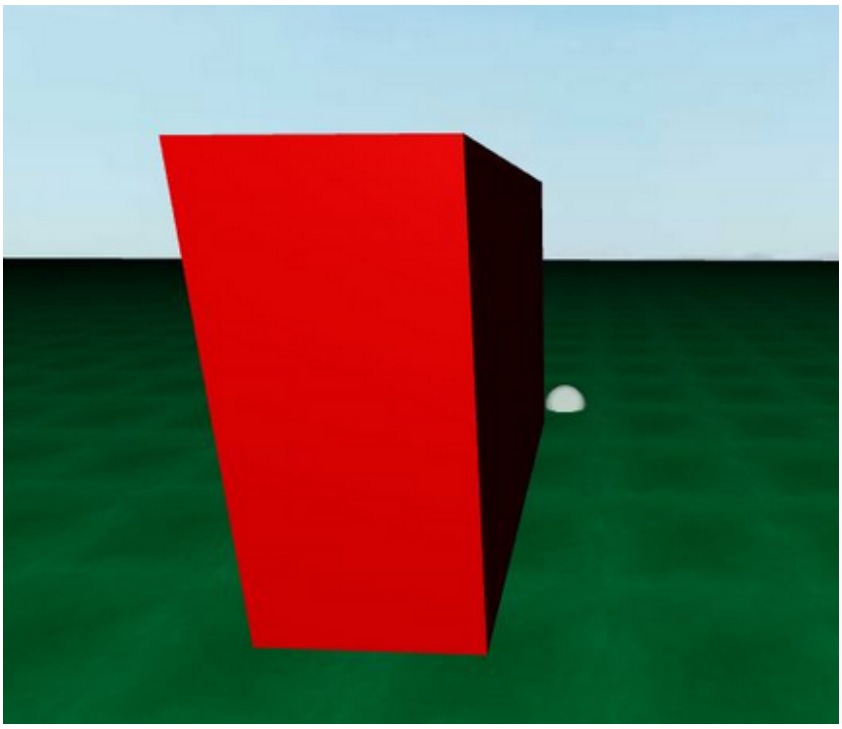

Figure 3.4: Primrose box [theory, 2016]. [Screenshot taken by the authors of this survey.]

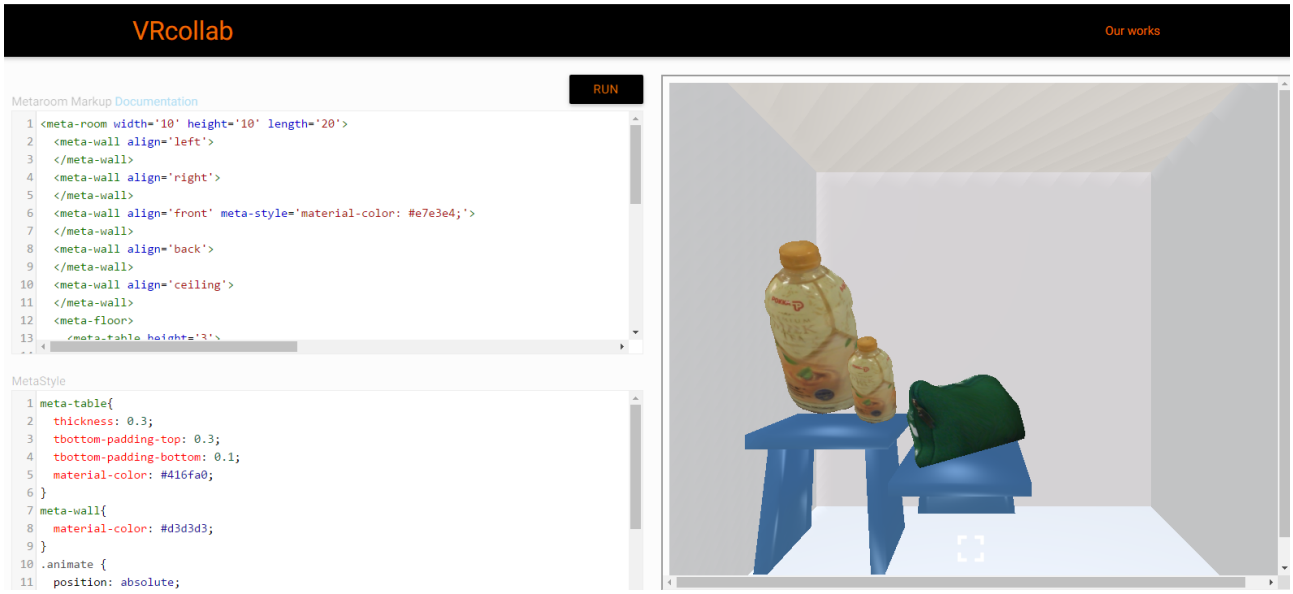

Figure 3.5: WebVR Markup example [Markup, 2016]. [Screenshot taken by the authors of this survey.]

```
<script type="text/javascript">
      var env = new Primrose.BrowserEnvironment({
        skyTexture: "bg.jpg",
        groundTexture: "deck.png"
      });

      env.addEventListener("ready", function () {
        // Perform any post-initialization setup. Once this event fires, the Primrose
        // framework is ready and will start animation as soon as this function returns.

        env.insertFullScreenButtons("#fsb");
      });

      env.addEventListener("gazecomplete", function (evt) {
        // You can respond to "intended stare" events here, i.e. when the user gazes
        // at a particular object for an extended period of time. Usually, about three
        // seconds.
      });

      env.addEventListener("pointerend", function (evt) {
        // You can respond to the user "clicking" an object here. This could be by using
        // a mouse on their desktop PC or by touching the screen while looking at an
        // object on a mobile device.
      });

      env.addEventListener("update", function (dt) {
        // Perform per-frame updates here, like moving objects around according to your
        // own rules.
      });
    </script>
```

Listing 3.5: Primrose events [theory, 2016]

```
<meta-room>
  <meta-wall align='left'></meta-wall>
  <meta-wall align='right'></meta-wall>
  <meta-wall align='front'></meta-wall>
  <meta-wall align='back'></meta-wall>
  <meta-wall align='ceiling'></meta-wall>
</meta-room>
```

Listing 3.6: WebVR Markup structure code [Markup, 2016]

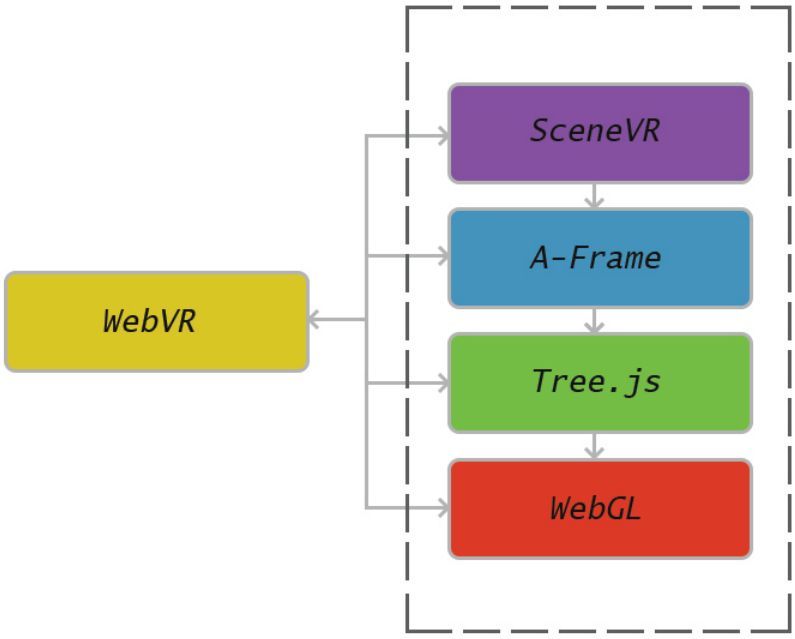

Figure 3.6: Architecture [Screenshot taken by the authors of this survey.]

Chapter 4

Unity Game Engine and WebVR

Unity is currently the most popular game engine with the largest community, an excellent set of technologies and outstanding developed games. This game engine provides stable API support for a wide variety of VR devices as well as build support for WebGL. Since WebVR development in Unity is not yet officially supported, there are solutions which make it possible to develop WebVR projects in Unity.

4.1 About Unity

This section is based on the Unity Documentation [Unity, 2016].

Unity is one of the most popular and most powerful game engines for building 2D and 3D games. To this fact definitely contributed the excellent graphical environment and wide variety of supported platforms such as Android, iOS, Windows, Linux, OS X, PlayStation, Wii, WebGL, Oculus Rift and many other. Unity also provides a large number of integrated tools and technologies which supports fast and comfortable development of a video game.

4.1.1 Editor

Editor provides simple and fast way of working with elements on scene. The role of the scene is to be a holder of all the objects, which the user is working with, to help the user to create the desired environment with all the obstacles or decorations. The field for object manipulation has a core significance because every action over the object (simple game object, camera, enemy player...) is connected with and within the inspector section, where the user can adjust all the properties and test them even during the run mode.

4.1.2 Graphics

Graphics is one of the most important elements in every video game, therefore Unity offers many possibilities. While mesh and textures define how the scene is going to look like, lights define color and complete the atmosphere of a 3D environment. It is possible to combine several lights such are directional lights, point lights, spot lights, area lights and the result is pretty impressive. All these lights are being made in such a way that they are calculated and executed in real time for every single frame and if it is already known that some lights are not going to be changed, Lightmapping can be used to improve game performances. In order to enrich a game experience, Unity offers a camera system which can be controlled according to user needs. One camera will be always present on the scene but more cameras can be added as well. This provides the option to split the screen for more players. These cameras can be animated and controlled by physics and practically everything can be achieved with cameras.

Shaders (small scripts which enables color leveling inside the image) are used in order to render the graphics in Unity. Shaders are being used through materials which combine shader code with elements such as textures. Every user can make shaders but there are a number of them provided by the engine. They can even be

optimized for mobile devices in order to provide better performances. All of these things help users make very professional and engaging environments for the final players. One of the components, which is worth mentioning, is the terrain generator. This system has great capabilities for landscape creation. Highly optimized terrain rendering is being executed during the runtime and in the editor mode users can manipulate with different tools and quickly and easily create the landscape. Along with everything mentioned, there is also a particle system, which provides the function to simulate moving entities such as fluids, clouds or flames by generating and animating a large number of small 2D images. Meshes are not dedicated to be used in these situations, they are used to represent solids and they are the main graphical base in Unity. Unity also supports Skinned Mesh Renderer, which is very important for animated meshes for character models, and Text Mesh, which is used for rendering 3D text on the scene. Regarding 2D games, users can use Sprite Rendered for their sprites.

4.1.3 Physics

Physics in game enables objects to behave in a way like they would behave in the nature. The virtual reality system needs to perform simulation of movement speed, gravity, collisions and other forces. To make this easier, unity provides a physics engine which can be managed with providing the parameters. With these tools, users can make almost everything, from car movement to cloth movement simulation. There are two main physics engines for 2D and 3D elements. In order to enable physics simulation, the Rigidboty component needs to be added on a game object. With this component, attached objects will automatically get physical properties and start to react with gravity. When the body is not moving itself anymore or that movement is under some threshold, the body is being set to the idle mode to improve game performances. If some other object hits or collide with an idle object, then this object goes back into the motion state.

Collider components define the shape of the object over which physics should be applied. Components of this type which have simple shapes are called Collider basic components. In the 3D physics those are the Box Collider, Sphere and Capsule Collider, and in the 2D physics they are: Box Collider 2D and Circle Collider 2D. When an object form is very complex, there are Collider tools which will automatically adjust the shape of the desired appearance. 3D physics can use a Mesh Collider and 2D physics uses Polygon Collider. When there is a need for cases such as passage through a door or throwing the ball into a basket, where the system should only detect contact with an object but not stop it, then the triggers should be used. There are many different combinations of collisions in the Unity environment, and it all depends on the type of game and the elements which are on the scene.

4.1.4 Scripting

Scripting is a necessary component in the development of video games. It is there to respond to a given command, control a car or a character, make an effect, control physical movement of objects and so on. Scripting is based on the Mono open source project, which is an implementation of the .NET Framework, and developers can use languages such as UnityScript, C# or Boo.

4.1.5 Sounds

Sounds are giving a spirit to every game and it would not be complete without the sound effects or background music. The Unity audio system offers great opportunities by supporting a variety of audio formats with a sophisticated way of processing and reproduction sounds in 3D space, with a number of effects. In order to simulate the sound effect from a certain position, the sound source should come from the Audio Source component which exists on a game object. These sound sources are being listened and gathered by Audio Listener which is located on some other object, usually the main camera. In this way, Unity can calculate the distance to the sound source and also reproduce a sound with a specific strength. In order to simulate the Doppler effect, the system can use the speed of a moving sound source.

4.1.6 Animation system

Animation system such as Mecanim (system for humanoid characters) is designed to facilitate the users' work with animations. Since humanoid characters appear in many video games, the Unity system provides special tools which facilitate development. When an Avatar (an interface which supports the usage of animation intended for one model to other models) is properly configured, Mecanim will recognize the structure of the skeleton. After that, Mecanim supports motion range control using set of the setting for bones and muscles. The full power of Mecanim system can be seen when working with humanoid animations. However, other types of animations are possible as well, but without the Avatar system, and other features. These other animations are called Generic Animations.

4.2 VR and Unity

Since VR is becoming more and more popular, the need for VR devices being supported by Unity became very obvious. Unity provides an API which can target various VR devices, which means that there is no need for any additional plugin and everything can be done directly within Unity. API expansion and coverage is following VR growth and new devices are being covered in updates on a regular basis. The VR support can be turned on very easily, just by enabling the option "Virtual Reality Supported" in the project settings section. After that, users will be able to use a stable version of the single API for various VR devices and to have a clean project without additional plugin folders, completely ready for the development.

4.3 WebGL and Unity

Having unity games in the Web sounds perfect, because users would not be bothered with installing additional plugins or installing a new game every single time. In order to achieve this, it is required to have WebGL 3D graphics library and JavaScript, because JavaScript is the standardized language which runs in all web browsers. Fortunately, there are converters (for example Emscripten [GitHub, 2010]) which can convert C/C++ into highly-optimised JavaScript, or precisely saying asm.js format, which is a research project developed by Mozilla. Its aim is to formally define a subset of JavaScript and code, which can avoid some potential slowdowns in code (such as using variables of the same type), and which is still runnable by all web browsers.

The remaining part is a conversion of .NET game code, into C++. Unity has developed a solution, called IL2CPP, which uses .NET byte code and converts it into corresponding C++ code files. These code files can be taken by Emscripten compiler, which compiles it into JavaScript. The process of code conversion can be seen on Figure 4.1. After the WebGL build was performed, Unity generates the following files:

- index.html - starting point for web a web browser.
- Release / Development folder - contains generated game content.
- TemplateData folder - contains loading bar and additional template assets.

If a game is ready for release, these files are compressed and ready for usage in the web. However, a development build can be done as well as additional debugging and profiling.

4.4 WebVR integration

Currently, virtual reality in the web is not officially supported by Unity. Builds for VR and WebGL are possible but independently. Since official support for WebVR does not exist in Unity, there is a bridge, developed and maintained by the community, which makes this possible. The process of plugin integration is shown in Figure 4.2, Figure 4.3 and Figure 4.4.

A practical example with integrated WebVR plugin (see Figure 4.5) displays game object and camera structure. The project is completely ready to be built for WebGL. The selected WebGL template contains

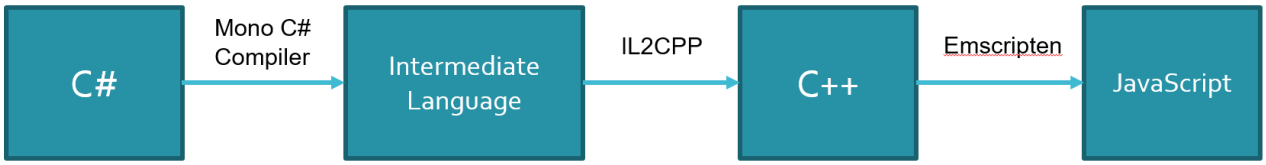

Figure 4.1: Build process of Unity game content to WebGL content

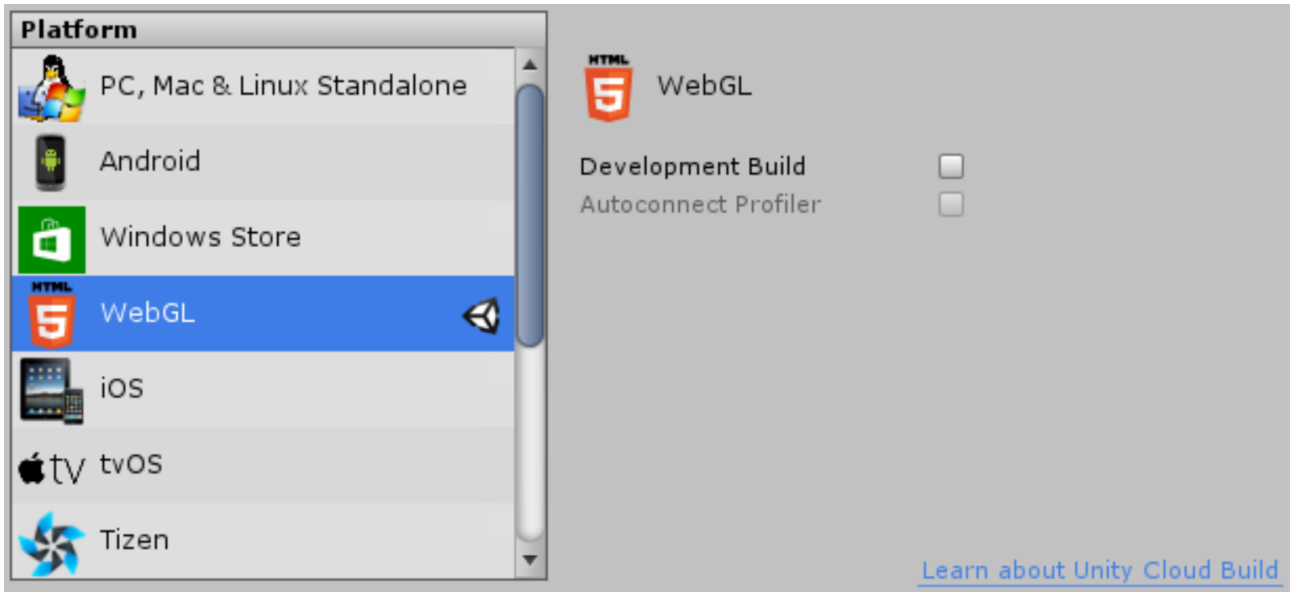

Figure 4.2: Selecting WebGL as a build platform [GitHub, 2015]. [Screenshot taken by the authors of this survey.]

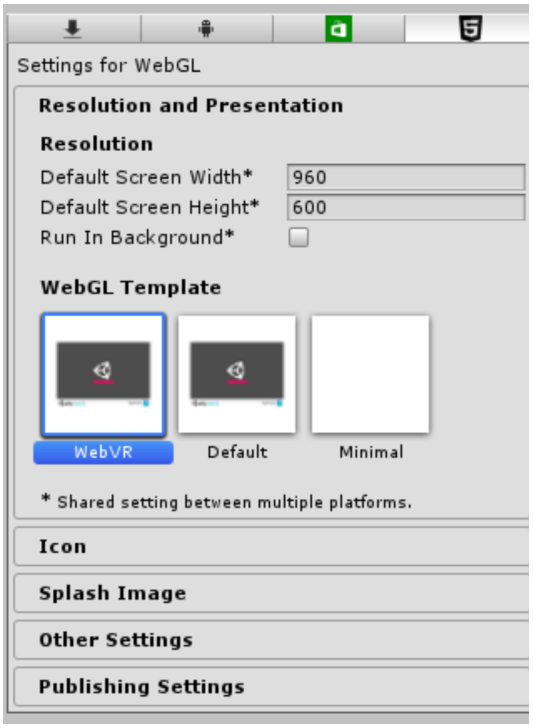

Figure 4.3: Selecting a template in project settings. The template WebGL requires is WebGL Template [GitHub, 2015]. [Screenshot taken by the authors of this survey.]

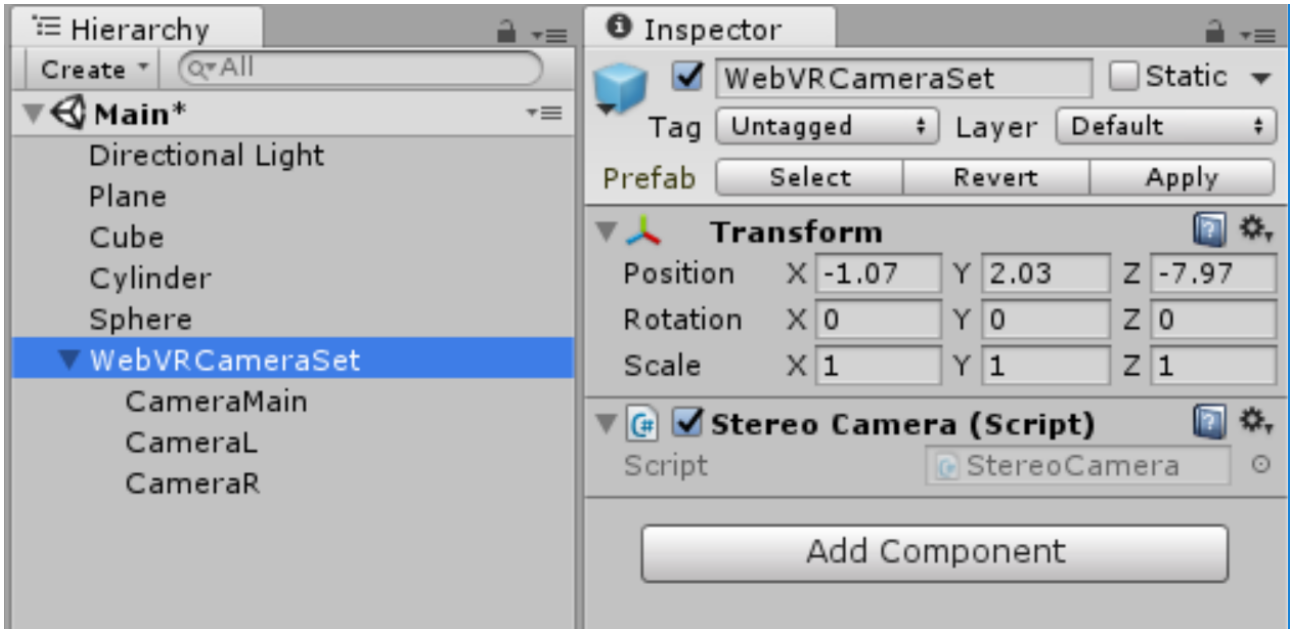

Figure 4.4: Setting up the camera can be achieved by replacing Main Camera with WebVRCameraSet prefab [GitHub, 2015]. [Screenshot taken by the authors of this survey.]

WebVR manager script which takes position and eye parameters from WebVR API and then controls the game using the interface SendMessage('WebVRCameraSet', 'attribute', value).

Once a WebGL project is run (see Figure 4.6), everything is ready for a user, who can attach a VR device and test the project in immersive environment.

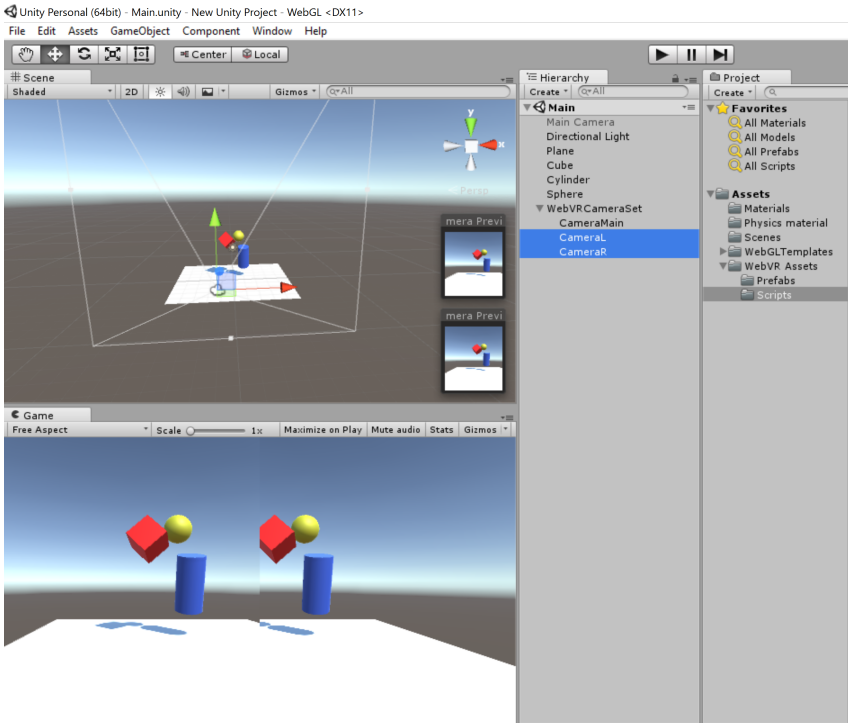

Figure 4.5: Simple Unity example with integrated WebVR plugin. [Screenshot taken by the authors of this survey.]

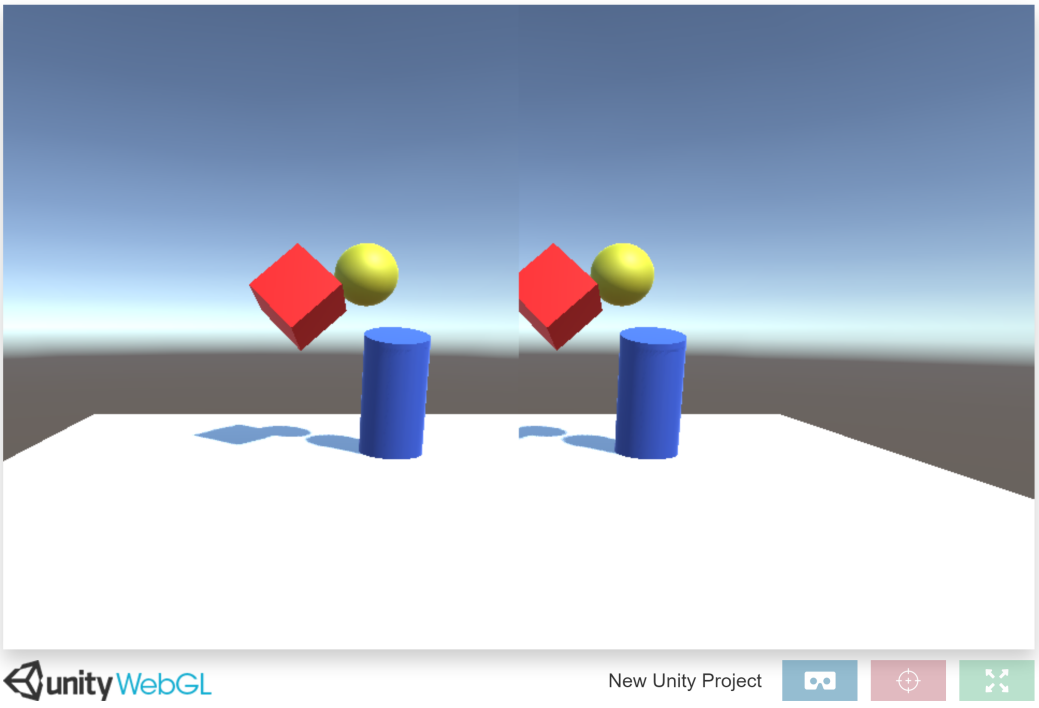

Figure 4.6: Unity WebVR demo run in Chromium [2016]. [Screenshot taken by the authors of this survey.]

Chapter 5

Concluding Remarks

Currently, in December 2016, experimental implementations of WebVR are only available in the Firefox Nightly release and the experimental builds of Chromium Browser as well as in the Samsung Gear VR internet browser. However, browser-based Virtual Reality is a highly dynamic area, evolving and developing quite rapidly. Many "big players" in the VR field are supporting the development and uptake of WebVR as a standard for browser-based Virtual Reality. The following examples are strong indications for "big players" engagement with respect to the adoption of WebVR :

- There is a W3C WebVR Community Group (including Microsoft, Mozilla, Google, Oculus...) working on the further elaboration of the WebVR specification. [W3C WebVR Community Group, 2016; Olivier, 2016]
- On September 20, 2016 Frank Olivier, Principal Program Manager Lead at Microsoft, announced that Microsoft is working on WebVR support in the Microsoft Edge browser. [Olivier, 2016]
- In October 2016 Oculus announced that the forthcoming Oculus VR web browser (called "Carmel") will also support WebVR. [Constine, 2016]

There are mainly two reasons for the strong support of the adoption of WebVR among the producers of HMD VR devices: On the one hand browser-based VR makes virtual reality easily accessible for everybody and thus increases the (potential) customer group for HMD VR devices, and on the other hand WebVR makes it also easier for developers to create cross-platform applications and thus increases the available content that can be experienced by users of HMD VR devices.

Bibliography

Charara, Sophie [2016]. *Explained: How does VR Actually Work?* 5th Oct 2016. https://www.wareable.com/vr/how-does-vr-work-explained (cited on page 1).

Chromium [2016]. *Chromium*. 29th Nov 2016. https://www.chromium.org/ (cited on page 28).

Constine, Josh [2016]. *Oculus Will Bring VR to the Web with React VR and Carmel VR Browser*. TechCrunch. 6th Oct 2016. http://social.techcrunch.com/2016/10/06/oculus-webvr/ (cited on page 29).

FOVE, Inc. [2016]. *FOVE Eye Tracking VR Headset*. 2016. https://www.getfove.com/ (cited on page 2).

GitHub [2010]. *Emscripten*. 22nd Aug 2010. https://github.com/kripken/emscripten (cited on page 25).

GitHub [2015]. *Unity-WebVR-Assets*. 26th Apr 2015. https://github.com/gtk2k/Unity-WebVR-Assets (cited on pages 26–27).

Group, Khronos [2016]. *Can I use*. 1st Oct 2016. http://caniuse.com/#feat=webgl (cited on page 8).

HTC-Corporation [2016]. *Recommended Computer Specs*. 2016. https://www.vive.com/us/ready/ (cited on page 2).

Jackson, Dean [2014]. *WebGL Specification*. 27th Oct 2014. https://www.khronos.org/registry/webgl/specs/1.0/ (cited on page 8).

Markup, WebVR [2016]. *WebVR Markup*. 24th Feb 2016. http://vrcollab.com/2015/08/10/what-is-metaroom-markup.html (cited on pages 20–21).

Mihelj, Matjaž, Domen Novak and Samo Beguš [2014]. *Virtual Reality Technology and Applications*. Volume 68. Intelligent Systems, Control and Automation: Science and Engineering. Netherlands: Springer, 2014. ISBN 9789400769090 (cited on page 1).

OculusVR, LLC [2016]. *Rift - Overview - Recommended PC Specification*. 2016. https://www3.oculus.com/en-us/rift/ (cited on page 2).

Olivier, Frank [2016]. *Bringing WebVR to Microsoft Edge*. Microsoft Edge Dev Blog. 9th Sep 2016. https://blogs.windows.com/msedgedev/2016/09/09/webvr-in-development-edge/ (cited on page 29).

Parisi, Tony [2015]. *Learning Virtual Reality*. 1st Oct 2015. http://shop.oreilly.com/product/0636920038467.do (cited on page 18).

Parisi, Tony [2016]. *GLAM*. 24th Feb 2016. https://tparisi.github.io/glam/ (cited on pages 18–19).

Pixelface [2016]. *Vizor*. 5th Dec 2016. https://vizor.io/ (cited on page 17).

Team, Mozilla VR [2016]. *WebVR Bringing Virtual Reality to the Web*. 1st Jan 2016. https://webvr.info/ (cited on page 5).

team, Mozilla VR [2016]. *A-frame*. 23rd Nov 2016. https://aframe.io/ (cited on pages 14, 16).

theory, Notion [2016]. *Primrose VR*. 12th May 2016. https://www.primrosevr.com/ (cited on pages 19–21).

Unity [2016]. *Unity Documentation*. 29th Nov 2016. https://docs.unity3d.com/ (cited on page 23).

Vukicevic, Vladimir [2014]. *Slideshare*. 20th Sep 2014. http://www.slideshare.net/fitc_slideshare/web-vr-fitc (cited on pages 8, 10–11).

Vukicevic, Vladimir, Brandon Jones, Kearwood Gilbert and Chris van Wiemeersch [2016]. *W3C on GitHub*. 9th Nov 2016. https://w3c.github.io/webvr/ (cited on pages 5–6, 8).

W3C WebVR Community Group [2016]. *Draft WebVR Community Group Charter*. 2016. https://w3c.github.io/webvr/charter/ (cited on page 29).

Yee, Chasey [2016]. *Hacks*. 1st Mar 2016. https://hacks.mozilla.org (cited on pages 5–9).