OULU UNIVERSITY OF APPLIED SCIENCES

Roman Kucherenko

WEBVR API DESCRIPTION AND A-FRAME APPLICATION IMPLEMENTATION

WEBVR API DESCRIPTION AND A-FRAME APPLICATION IMPLEMENTATION

Roman Kucherenko
Bachelor’s Thesis
Spring 2019
Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Information Technology

Author: Roman Kucherenko
Title of Bachelor´s thesis: WebVR API Description and A-Frame Application Implementation
Supervisor: Veikko Tapaninen
Term and year of completion: Spring 2019 Number of pages: 35+1

The aim of the thesis was to study and analyse the VR based technology in web browsers and define its potential in web development and Virtual Reality market. WebVR is an experimental application programming interface, which started to gain a momentum from year 2014, and it looks promising from many perspectives, such as education, design, medicine and entertainment, regarding to its simplicity from the point of applications development.

To accomplish a desired objective, WebVR API was used and tested in Chrome, Firefox and Edge browsers with VR related extensions. Other browsers such as Opera, Internet Explorer and Safari were also used in order to prove that the API is not or partly supported. The demo application was built on the 0.8.2 version of A-Frame framework based on Three.js, the Cannon.js physics library and additional A-Frame library extensions. Testing platforms were personal computers on Windows and MacOS and smartphones on Android and iOS operational systems.

As a result, the demo application demonstrates the great functionality, optimisation and relatively good performance of the WebVR API. Summarizing the gathered experience and the result, web-based VR API is dependent on a global VR market and defined by markets status.

Keywords: VR, WebVR, API, JavaScript, 3D graphics, Three.js, A-Frame

PREFACE

This thesis was based on the author’s own idea and it was done with the help of instructing teacher Veikko Tapaninen. The Web-based VR subject was picked because of its interesting concept and great experience that it can give. WebVR applications are taking the part of browser games development sphere – one of the most interesting spheres of web development. 3D visuals and graphics are playing an important role in e-commerce, they create interest and leave pleasant sensations in case of proper implementation. VR is exciting technology which increases consumer demand at a very high speed and WebVR brings it into browsers allowing easy access and experience, opening more possible values into the marketing fields. I do not believe that WebVR will become a widely useful and highly valuable technology, but it has its own potential, especially in hands of creative and skillful people.

Oulu, 8.5.2019
Roman Kucherenko

CONTENTS

VOCABULARY

API – application programming interface

AR – augmented reality

CSS – cascading style sheets

CPU – central processing unit

DOM – document object model

ECS – entity-component-system

FPS – frames per second

GPU – graphics processing unit

HMD – head mounted display

HTML – hypertext markup language

IT – information technology

JS – JavaScript

PC – personal computer

OS – operational system

UI – user interface

VR – virtual reality

WebGL – web-based graphics library

1 INTRODUCTION

The WebVR concept was invented with an intention to simplify development and experiencing of virtual reality (VR) applications by creating a JavaScript (JS) library for supporting virtual reality devices in web browsers. The ideology of the technology implies that the implementation process of web-based VR applications should become easier and require less coding experience, time and finances. The WebVR application programming interface (API) has been in existence since 2014 and till the past time it remained controversial, being almost an imperceptible deviation from the main VR industry development (1). It had not gained much popularity, mainly because WebVR applications are free in access like usual web pages and cannot bring much of a profit. However, it is relatively optimized and gets support by most of used browsers. WebVR is continuously developing, the concept is interesting and may bring many new opportunities in the near future (2). The main objective of this thesis work is to explain how the technology works, describe the implementation process of an application and in conclusion define WebVR's current opportunities and potential, provide valuable feedback and predict a possible development based on data from analytical resources.

The application for the thesis research is built on the A-Frame framework and it represents a basketball game with the concept of throwing a basketball through the hoop and scoring points. It is playable on desktop and mobile devices in both VR and non-VR modes. This game intend to show not only the capabilities of WebVR API, but also a combination of technologies, which can be used for building a browser-based VR application.

2 VIRTUAL REALITY

2.1 History of VR

Virtual Reality (VR) does not have a defined date of birth as a concept of using stereoscopic pictures in order to simulate three-dimensional images. Back in 1838, an English physicist Charles Wheatstone made a research demonstrating the ability of brains to convert two two-dimensional pictures into one three-dimensional object using a stereoscope. This research served new discoveries in binocular vision studies (3). Later in period from 1930 to our days, the scientists, inventors and enthusiasts were creating different prototypes, devices and computers using 3D visual effects and stereoscopic displays (4).

The “Sensorama” was one of the first VR prototypes, patented in 1962 by Morton Heilig and allowing watching short movies providing a stereoscopic 3D image, sound effects, smell, a feelings a wind blow and a vibration coming from the chair in response to some actions. The Telesphere Mask was the first head-mounted display, which was a small box with straps and with two displays showing the television in a stereoscopic view and producing sound. It was also invented in 1960 by Morton Heilig (5). His next prototype combined both previous inventions into one, which was supposed to become a 3D motion picture theatre, where people could see a stereoscopic picture on a distance wearing polarized glasses, smell aromas, feel wind, temperature, variations and body tilting of the seat (6). Nowadays, Morton Heilig is sometimes called the "Father of Virtual Reality" in some books and articles because of his revolutionary inventions, although they were not met well by critics and public. (5)

The 'virtual reality' term was popularized by a video game developer Jaron Lanier in 1987. He founded an VPL Research company, which was focusing on commercializing virtual reality technologies, it was the first company that sold commercial VR systems, such as VR head mountain displays named EyePhone, which had a form of today’s VR headsets, were wire-connected to computer and could track head movements (7). Additionally, VPL developed the DataGlove and the DataSuit, which represented gloves and a suit as advanced controllers for virtual reality. Gloves had 6,502 microcontrollers and was used for computer control and gaming, it also had the potential for a remote surgery. The suit was provided with sensors for measuring the movement of arms, legs, and trunk. VPL developed a working prototype but it had no time to caught on because the company was filed for bankruptcy in 1990. (8)

Besides VPL Research, on the wave of popularity of a new technology other companies started to produce their own variants of VR headsets and games from them. Great examples are VR glasses from Sega and Virtual Boy from Nintendo. The Sega VR glasses were supposed to have head tracking, stereo sound and two LCD screens for both eyes, but it never came into the market because of technical development difficulties. Nintendo’s Virtual Boy was not just glasses, it was a portable console that displayed 3D graphics, but it did not confirm to the standards of ergonomics, which caused usage problems and motion sickness for users while playing. Because of high development costs and lack of technical capabilities and computing power in 1990’s, the implementation of well-designed and qualitative VR was unattainable. (8)

Besides gaming industry, VR systems were applied in education and training to medicine and military fields. From the end of 1970’s and until 1989, an American professor and inventor Thomas Furness was developing virtual interfaces for a flight control of the U.S. Air Force. He was able to produce flight helmets with displays, which showed the most important information related to a flight, such as computer-generated 3D maps, forward-looking infrared and radar imagery, and avionics data that the pilots could view and hear in real time. The helmet had tracking systems to determine position, orientation, and gaze direction, voice control and sensors, with which a pilot could control the aircraft with gestures, utterances, and eye movements. (9)

At the same period of time, the medical field the started development of technology providing telepresence in surgeries by manipulating robotic devices controlled remotely. The first achievement was the transmission of images through micro cameras attached to endoscopic devices and to monitors for a group of surgeons during the surgery. This provided the impetus towards the development of a remote surgery. During 1990s, a DARPA initiative funded a research to produce telepresence workstations for microsurgeries and other less invasive forms of surgical procedures. The first robotic surgery was performed at the Broussais Hospital in Paris in 1998 (9). Nowadays, modern technologies allow training of future specialists in VR simulators with a similar concept of remote surgery. Using an VR head mounted display (HMD) and a set of surgical instruments, people can practice and learn how to perform real surgeries that are reproduced in a virtual environment (10).

2.2 VR technology description and its appliance

Virtual reality represents a computer simulation of a certain environment or world transmitted to a person through the main senses - sight and hearing. The user conducts entering in virtual reality by VR-devices, such as stereoscopic HMD, various motion sensors and controllers, which are tracking users head and body movements and controller inputs, and other specialized equipment. VR is one of the main technological trends of recent times and became one of demanded directions of Information Technology (IT) development industry. The reason behind that is a big potential that opens new edges for digital marketing and retail strategy (11). The concept brings fundamentally new ways of experiencing usual things, creates opportunities for developers and implies revolutionary ideas. The technology is already applied in many industrial and marketing areas, such as computer and mobile game development, architecture and town planning, vehicle manufacturing, extraction of minerals, training and education, promotion of tourism, marketing and advertising, military and others. The automotive industry provides an ability to explore, customize and test-drive a car, which does not even have a prototype (12). Online shops, like Alibaba and Ikea, let customers select products like clothes or furniture by VR applications. People are able to view the product from all sides for example by rotating it, changing colors, shape or size (13).

The Virtual Reality gaming industry is one of the fastest growing and exciting sphere of VR industry in the development of mobile and computer game applications. By the year 2025, the global VR gaming market revenue is going to be up to 45 billion dollars according to the forecast from Grand View Research Inc. The current revenue with a yearly forecast can be seen in Figure 1. The main reason for that is improvements in the VR HMD picture quality, ergonomics and technical capabilities. Other reasons affecting the VR market growth are promotion of technology and advertising from large companies like Google, Samsung, Sony and Oculus VR. In addition, a price reduction based on the growing competition on the market will increase consumers interest. Mobile gaming impacts the VR market growth as well since it is a related market and it is becoming more common. The choice of mobile applications is expanding and new applications are released more often. (14)

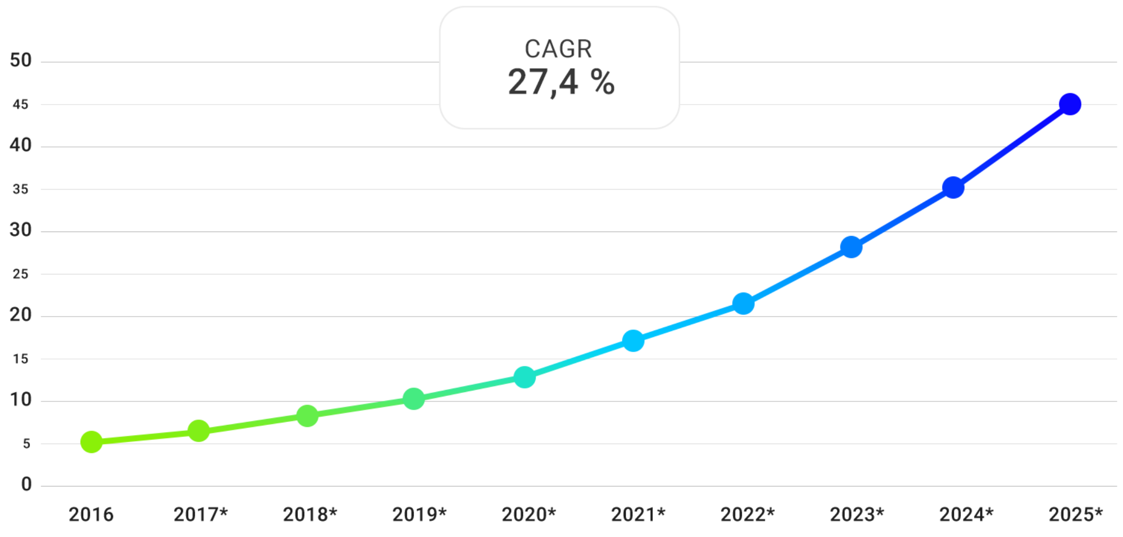


FIGURE 1. VR gaming market revenue forecast in $ billions. (14)

Nowadays, a wide range of the VR-devices is available on the market. They can be divided into two groups. Those that are intended for personal computers (PC) and gaming consoles, and others that are simpler and designed for mobile devices. Devices for PC and game consoles are more complex and cost significantly more, but they provide a better immersion effect. Advanced VR headsets represent a box with two built-in displays for each eye to output an image and a rotational and positional tracking feature. In order to provide a satisfying experience, each display should have a high resolution to make a picture smoother and pixels almost invisible for an eye while using a headset. That requires a high computing power of a processor and involves large expenses that make this option less accessible. In fact, to provide a smooth VR PC gaming experience, the computing power should provide stable 90 frames per second (FPS) and a minimum resolution of 960x1080 to both separate displays, which is in total 1920x1080 pixels. (15)

Mobile VR gaming occupies a low-cost market segment and is accessible to a wider consumer audience. A mobile VR headset is technically simpler due to the fact that it is just a plastic or even cardboard box with two lenses. The lenses, as in more advanced headsets, are used to focus a user's eyes on a display. Any average performance smartphone placed inside of such box is playing the roles of a display, head movements tracker and small computer. This makes the VR user experience lower comparing to an experience with advanced headsets, and additionally the set of applications is very different depending on the platform. WebVR solves that issue as well by defining a single platform, which is a browser, and it does no longer matter which headset type is used, WebVR applications are accessible from every device with a browser supporting the API. (16)

Designing and development of native applications for mobile and PC VR groups differ depending on targeting operational systems (OS), device type and technical specifications. Developers have to consider the different trackers, the screens and types of stereoscopy, computing the correct perspective and other differences between hardware and software. This state of affairs creates difficulties in optimization and realization of the idea of the product, which is a usual problem in the software development sphere. A number of technical nuances often arises during the development process, bringing complications and higher costs. This problem is highly relevant in the field of PC VR software and applications development, and the number of various displays and controlling devices with a set of different technical specifications aggravates it (17). Removing such difficulties reduce costs and simplifies the development process, optimization and usability will improve and as a result, the product becomes more attractive and user-friendly. The possibility to achieve that simplicity for the VR market was found in the idea of providing the VR content in the web via browsers by developing a new API called WebVR.

3 WEBVR API DESCRIPTION

WebVR is an experimental technology, which was conceived by The Mozilla Corporation in 2014 and released in the beginning of 2016 with a stable version 1.0. The API provides support for using virtual reality devices in web applications, allowing developers to transform location and motion information from the device into movements and actions inside the virtual 3D environment. Its main goal is to facilitate a qualitative experience of virtual reality for each user, regardless of a device type and its operational system. (18)

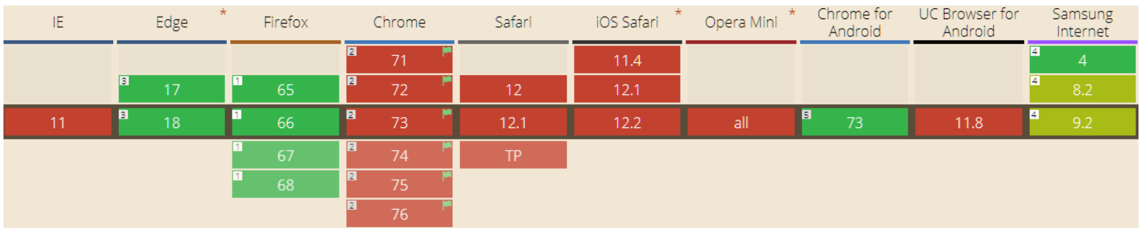
WebVR represents the JavaScript API, which accesses the Web-based Graphics Library (WebGL) API built into modern browsers for rendering interactive 2D and 3D graphics inside a hypertext markup language (HTML) <*canvas*> element (19). WebGL allows browsers low-level access to machine's graphics processing unit (GPU) for more advanced graphics rendering. Essentially, WebGL uses the graphics processor to perform two tasks. The first task is to handle the positions and the vertices of the clipping space - part of the scene that is in the user's field of vision. The second task is to draw pixels based on the results of the first task. The WebVR API is a software interface for working with devices and it is responsible for other tasks: VR device presence recognition, its capabilities and technical specifications, retrieving parameters of device position, orientation, velocity and acceleration, and rendering stereoscopic images to both displays or their simulation using a split screen view. Thus, WebVR gets needed data for the image while the actual image is created using mostly HTML, WebGL, Canvas and CSS. (20)

WebVR applications are accessible in browsers in a similar way as usual web sites and other web applications. They are built to be viewed and played by any type of VR headset and they work in browsers where the WebVR API is supported. Such applications are possible to experience without a stereoscopic HMD as well, WebVR provides functionality of screen splitting, thereby a mobile or a computer screen can be divided into two independently rendering pictures with a little different view angle. (21)

3.1 Browsers support

Since the VR application works in a browser, it does not require any separate software and does not depend on any operating system. It only depends on WebVR API support of the browser, where the application is used. Support lies in the presence of a built-in WebVR API and its optimization for a browser environment with specific engines and libraries. Currently, WebVR has better support in Mozilla Firefox, Google Chrome for Android and Microsoft Edge for Windows and Android, and it is partly supported by Chrome for Windows and macOS, Safari mobile version and Samsung Internet browsers, which is visible below in Figure 2. Additionally, the API is supported in less popular browsers, such as Brave Browser for mobile platforms, UC Browser and Servo for desktops. As for other browsers, which are not made specially for VR, the API is not supported there at all (22). Besides the API support issues, a specific browser may not support various headsets, for instance Firefox supports Oculus Rift, while Chrome does not. (23)

The API is enabled by default in Firefox, Chrome for Android and Edge for Windows/Android. In the Chrome desktop version browser it can be enabled by setting a proper flag named *WebVR* in browser configurations. Regarding other browsers, in case if WebVR is not available natively, the problem can be resolved using an API polyfill. A polyfill is an imitation of a new, not yet implemented in some browsers API, giving them the reverse functionality (24). The WebVR polyfill usage opens an VR experience in more browsers, the most popular of which are Safari and Chrome for iOS. (21)



IE	Edge *	Firefox	Chrome	Safari	iOS Safari *	Opera Mini *	Chrome for Android	UC Browser for Android	Samsung Internet
			2 71		11.4				4 4
	3 17	1 65	2 72	12	12.1				4 8.2
11	3 18	1 66	2 73	12.1	12.2	all	5 73	11.8	4 9.2
		1 67	2 74	TP					
		1 68	2 75						
			2 76						

FIGURE 2. List of browser versions supporting WebVR API. (22)

Developers from Google and Mozilla are not the only ones who place bets on the WebVR concept and are sure that browsing in the future will be VR-based. Few browsers been released specially for VR browsing, with a completely different logic of page rendering tailored for surfing in the Internet with the VR headset. The list of such browsers includes e.g. Supermedium, Oculus Carmel, JanusVR, LensVR. The design of such browsers is more or less common and can be seen in Figure 3. Most browsers, currently, are in beta or developing stages, making them unstable to use. (25)

A Bulgaria-based Coherent Labs company was developing their own browser for VR browsing that is called LensVR. The browser architecture designed for high performance and support of virtual and augmented reality as well. “A virtual reality experience is more similar to a video game than to a 2D web – there is a 3D world, 3D object, light, sound, multiple users, etc. It requires a different underlying technology that works in the same way game engines do.” – says the product manager Bilyana Vacheva. Therefore, WebVR cannot give users a full potential of virtual experience in a web, since it is only an extension to usual browsers, made for plane web pages. It is hard not to agree with this, because the current look of existing WebVR applications are very primitive and unlikely to become more advanced on current bases. (25)

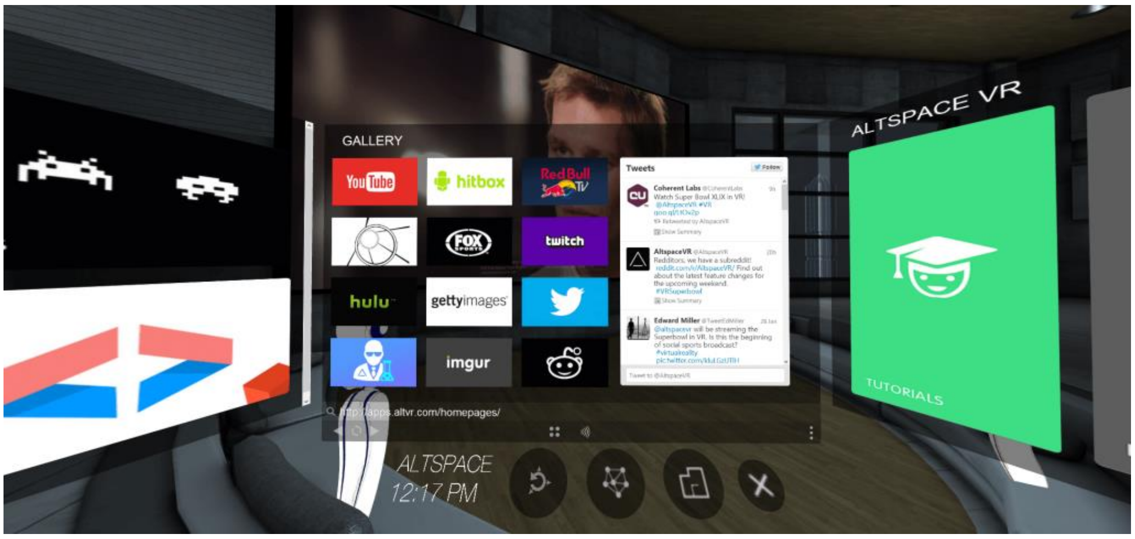


FIGURE 3. Example of VR-based browser user interface. (26)

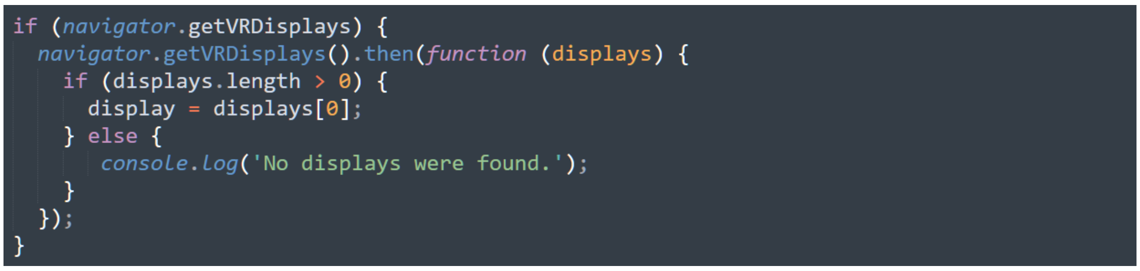
The complexity and advancement of WebVR applications strongly impacts on the performance. Processing and rendering visible objects in a tick of frame and maintaining their physical properties in relation to each other requires graphics and central processing units resources. For now, the infrastructure of the browsers shifts the 3D graphics processing load basically onto a central processing unit (CPU) and does not allow to use the full potential and possibilities of the GPU (27). In addition, performance slightly depends on the Internet speed and hosting specifications of a used resource. Thus, the main issue here is that applications should be enough simple and light to be played and behave correctly and provide best user experience. (28)

3.2 API interfaces and methods

The WebVR API provides several basic interfaces for working, such as *Navigator*, *VRDisplay*, *VRLayerInit*, *VRPose*, *VREyeParameters*, *VRFrameData*. The *Navigator* interface represents the state and features of the user agent. This allows scripts to recognize them and register themselves to perform certain actions. Other interfaces implemented by Navigator hold information about headset display and frames data, position and direction of movements, information on how to render a picture to each eye and information about the scene frame for a projection on a single eye. (29)

To detect the available devices, the API provides an *navigator.getVRDisplays()* method, which returns a promise fulfilled with an array of all connected VR displays and resolves to a number of *VRDisplay* objects. Depending on the display presence, VR applications can or cannot be viewed in VR mode. WebVR applications usually have a button titled “Enter VR” or “VR not found”, where buttons text and functionality directly depend on availability status of the displays. It is possible to test the *getVRDisplays()* method manually by running it in the console of browser inspect panel, but if no HMD emulation is used, no devices will be found because computers are not treated as VR displays. Not every device with a display and accelerometers can be considered as an HMD and in case with the emulation they can only provide a pseudo-VR experience. A specialized Chrome extension named “WebVR API Emulation” enables an emulator of VR

HMD object, more specifically HTC Vive, to allow VR mode use for developers. Mobile browsers provide such emulation by default. Other VR devices, such as Oculus Rift or HTC Vive, are the actual HMDs and does not require emulations. An example of *getVRDisplays()* is presented in Figure 4 below. (30)



```
if (navigator.getVRDisplays) {
  navigator.getVRDisplays().then(function (displays) {
    if (displays.length > 0) {
      display = displays[0];
    } else {
      console.log('No displays were found.');
    }
  });
}
```

FIGURE 4. Example usage of navigator.getVRDisplays() method.

The VR display has a number of capabilities, which are used in a *VRDisplayCapabilities* interface to determine if a display is applicable. Capabilities are defined by five attributes returning the Boolean value, being *hasPosition*, *hasOrientation*, *hasExternalDisplay*, *canPresent* and *maxLayer*. The *hasPosition* and *hasOrientation* attributes obviously define if a display is capable of tracking its position and orientation. The *hasExternalDisplay* determines a presence of external display and in case if it exists, the application should mirror the VR content and update a non-VR UI because that content is visible on primary display of a device. The *canPresent* obtains a value depending on ability of a head mounted display or other device to present VR content. The *maxLayer* return a maximum number of layers that the VR display can present at once, but is null in case *canPresent* is also null. (30)

After VR devices get detected the *VRDisplay.requestPresent()* usually used to start displaying a scene on a VR device. The *VRLayerInit* interface, in its turn, represents a content layer, being an *HTMLCanvasElement* or *OffscreenCanvas* element that is intended for a VR output load. The *VRLayerInit* objects can be retrieved using a *VRDisplay.getLayers() method*, which present these objects using the *VRDisplay.requestPresent()* method. (30)

A *VRPose* interface contains position, orientation, velocity, and acceleration information of a *VRDisplay* and it is acceptable with *VRDisplay.getPose()*. It represents a state of VR sensor at a given timestamp and populates *VRFrameData* object with provided position information. That object is the one that calls the *VRPose by getFrameData()* method, which also gets view and a projection matrices for the current frame. Both matrices are calculated for each eye. They are responsible for proper rendering of a clipping space from two angles. Each eye data is provided with a *VREyeParameters* object. It returns a field of a view and width and height of an eye viewport. If eyes are rendered in a single render target, for example on a smartphone display, then the render target is calculated to be wide enough to fit both viewports because rendering pictures cannot overlap each other. (30)

The WebVR API additionally provides a *VRStageParameters* interface, holding information of a stage area for devices that support room-scale experiences. The *VRDisplayEvent* interface is responsible for detecting a display connection, pausing manually or when an application is not used for some time after a user took HMD off. The *Gamepad* interface allows using VR controllers or gamepads, which can be connected by a wire or Bluetooth. Starting from a middle of year 2018, the WebVR API became a part of a new WebXR API, where virtual reality is complemented with augmented reality (AR). Considering a novelty of that library, it is still under development and not stable, for now it is only used in Firefox, Chrome for Android and Edge. (30)

4 APPLICATION IMPLEMENTATION

There are several options to implement a web VR application, but JavaScript is the main programming language for implementation nevertheless. JS libraries such Three.js or Babylon.js, are made for building animated 3D computer graphics and also optimized for a VR environment creation. Auxiliary frameworks, for instance A-Frame and React 360, specialize on the browser-based VR and simplify implementation. Both of them are based on the Three.js library, but they are different in design and coding approaches. React 360 is developed by the Facebook team and has the same concepts, features, and benefits of React JS, so a developer have to be familiar with React. A-Frame is made by the Mozilla team and it became available in one year earlier. It is designed for people with less experience in JS since a 3D scene with objects can be defined in HTML as Document Object Model (DOM) elements. In the Figure 5 below it is shown that Reach VR gained more GitHub project starts in one month and then their number grew slower, while A-Frame projects start growth is more stable and steady. (31)

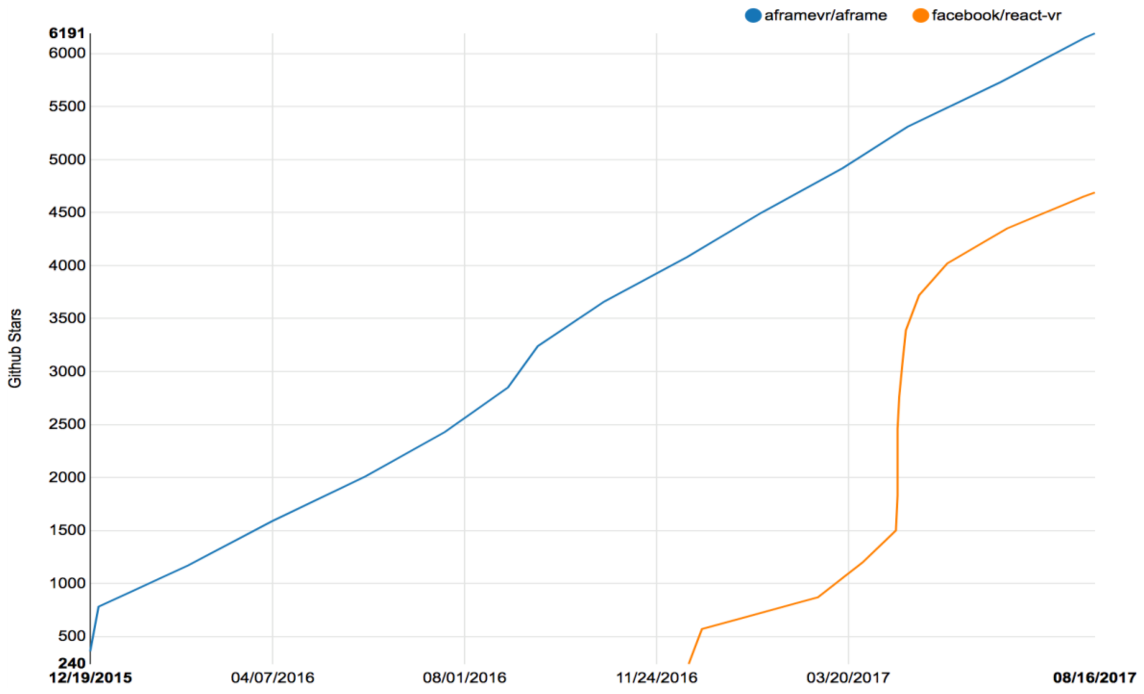


FIGURE 5. A-Frame and React VR GitHub starts comparison.

The A-Frame framework has a number of advantages over React 360 and regular Three.js. The open source framework uses an Entity-Component-System (ECS) architecture, which provides an accessible plug-and-play ecosystem. The ECS design pattern brings flexibility in designing a common software architecture by separating logic from data and providing easiness in modifying systems and components. In addition, it has a bigger community and number of components. In this way, A-Frame is an optimal option to develop the VR application, which is going to imitate a basketball field for the demo application of current research. (32)

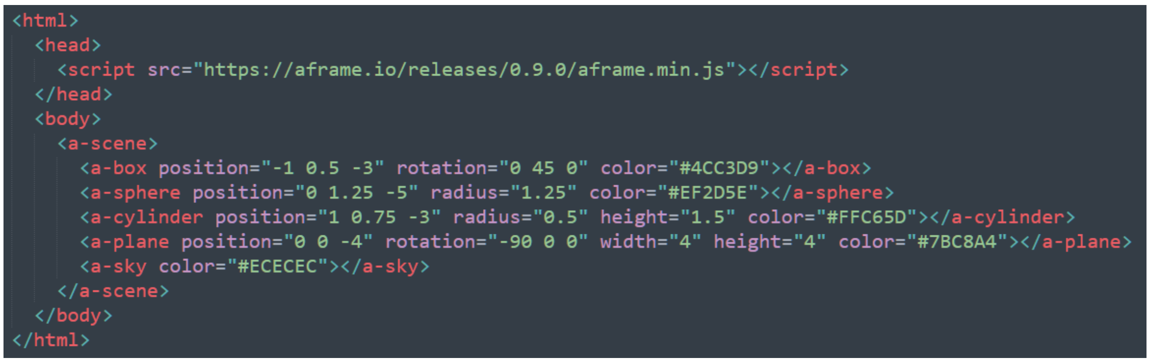
For this thesis work, the demo application is built on the A-Frame 0.8.2 version. The application implementation process itself is possible to divide into few stages, a number of which depends on its complexity. In case of this thesis work, the application implementation consists of four main stages: creating a scene and entity objects; adding physics; adding an ability to move a camera (player) using a keyboard and a mouse and a Bluetooth or wire connectable controller device; implementing interaction with objects and their collision detection.

4.1 A-Frame basics and creating of a scene

In order to use A-Frame, an external script of a framework library must be added to an HTML head section. The framework defines a scene by an *<a-scene>* tag that must be placed under the body element to represent a global object, inside of which the 3D objects and all related actions are handled. The A-Frame scene set up canvas, renderer and render loop by default. Canvas is a scene output element, and the renderer is what draws canvas output. The render loop causes the renderer to draw the scene every time when screen is refreshed, which is called frames per second (FPS) rendering. In addition, the a-scene provides a default camera and light entities, which could be overridden by custom entities. (33)

A-Frame provides a number of primitives out of the box. Primitives are entity-component wrappers initiated by semantic tags like *<a-box>*, *<a-light>*, *<a-camera>* and others that have a preset bundle of components with default values.

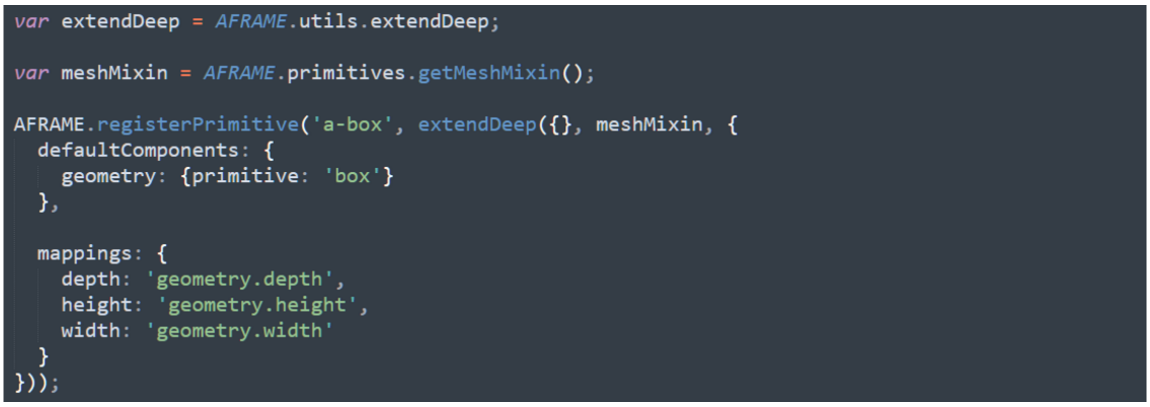
The components are attributes, which configurations affect the appearance of an entity, its color, texture, animations and physical properties. Editing component properties change related primitive. That allows a quick and flexible scene customizing. The example of scene initializing in an HTML template can be seen below in Figure 6. (34)



```html
<html>
  <head>
    <script src="https://aframe.io/releases/0.9.0/aframe.min.js"></script>
  </head>
  <body>
    <a-scene>
      <a-box position="-1 0.5 -3" rotation="0 45 0" color="#4CC3D9"></a-box>
      <a-sphere position="0 1.25 -5" radius="1.25" color="#EF2D5E"></a-sphere>
      <a-cylinder position="1 0.75 -3" radius="0.5" height="1.5" color="#FFC65D"></a-cylinder>
      <a-plane position="0 0 -4" rotation="-90 0 0" width="4" height="4" color="#7BC8A4"></a-plane>
      <a-sky color="#ECECEC"></a-sky>
    </a-scene>
  </body>
</html>
```

FIGURE 6. Example of initialising a-scene element and primitives with components.

Developers can create custom primitives by registering them through JavaScript. To do that, it is necessary to define default components and mappings for a new primitive. Default components describe set of components of a primitive, which are always applied unless they are overridden in the DOM. Mappings define custom components used as HTML attributes that relate to original property of a component. The code snippet presented below in Figure 7 demonstrates an example of mapping the custom components, which are going to be treated as properties of eponymous geometry component. (34)



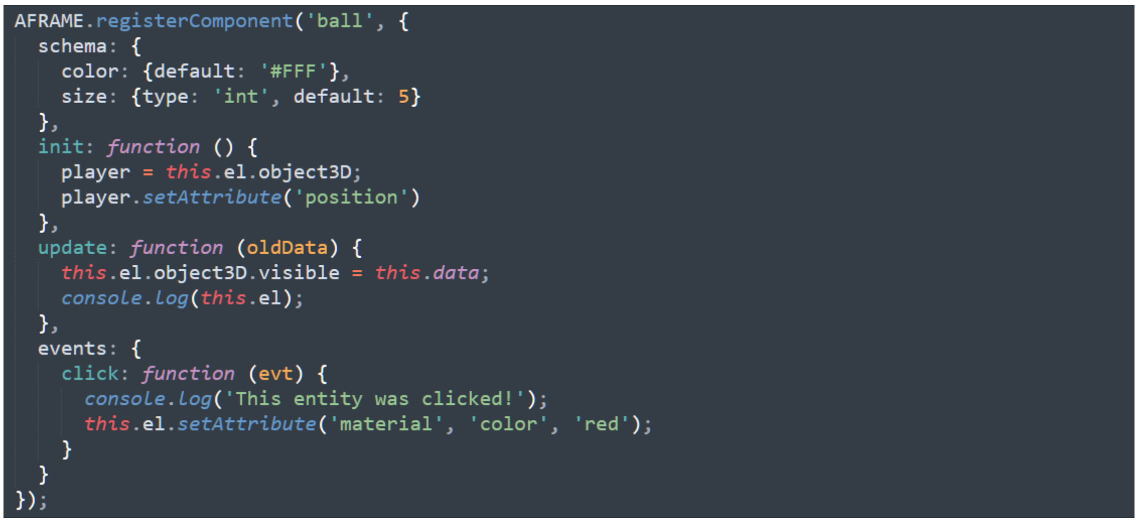
```javascript
var extendDeep = AFRAME.utils.extendDeep;

var meshMixin = AFRAME.primitives.getMeshMixin();

AFRAME.registerPrimitive('a-box', extendDeep({}, meshMixin, {
  defaultComponents: {
    geometry: {primitive: 'box'}
  },

  mappings: {
    depth: 'geometry.depth',
    height: 'geometry.height',
    width: 'geometry.width'
  }
}));
```

FIGURE 7. Example of registration for the <a-box> primitive.

In order to use an existing primitive in JS, it should have a custom component, which has to be registered with a dedicated *AFRAME.registerComponent()* method that provides a set of handlers allowing to use a primitive in different periods of its lifecycle. Totally, there are eight handlers available for a component: *init()*, *update()*, *remove()*, *pause()*, *play()*, *tick()*, *tock()* and *updateSchema()*. At least one defined handler is necessary for a component registration, others can be used if needed. Accessing an entity of a component is possible via "*this*" property. Another important component concept in A-Frame is the schema that describes properties of the component. The component registration example is provided in Figure 8. (*35*)

The *init()* function runs only once to initiate a custom component when a content is set to an entity in the HTML and a page is loaded, or when the component is added to a scene entity via *setAttribute()*, or when the entity is appended to a scene via *appendChild()*. The initiating handler is mostly used to set up an initial state and variables, bind methods and attach event listeners. The *update()* function is called right after *init()* and whenever component properties get updated. When a component gets removed from an entity, or an entity gets detached from a scene, A-Frame calls the *remove()* function. If an entity or a scene were paused, then a *pause()* function is called. It is also called before the related entity is removed. Each time when the entity or the scene is resumed, a *play()* function runs. The *tick()* and *tock()* handlers call a respond on an each tick or frame of the scene's render loop, with a difference that the *tick()* called before scene is loaded and the *tock()* is called after. Finally, *updateSchema()* is called when entities schema gets modified. (35)

Besides handlers, the component could have definition properties. The "*dependencies*" property controls the order in which entities should be initialized. The "*multiple*" property allows the component to have multiple instances of itself in an entity. It is set to false by default. The instances are defined by the name of a component with a unique suffix of a double underscore and an instance identifier. That is needed for an easy switching between a set of predefined properties. The "*events*" object property is intended for a component related event handlers like "*click*". (35)



```
AFRAME.registerComponent('ball', {
  schema: {
    color: {default: '#FFF'},
    size: {type: 'int', default: 5}
  },
  init: function () {
    player = this.el.object3D;
    player.setAttribute('position')
  },
  update: function (oldData) {
    this.el.object3D.visible = this.data;
    console.log(this.el);
  },
  events: {
    click: function (evt) {
      console.log('This entity was clicked!');
      this.el.setAttribute('material', 'color', 'red');
    }
  }
});
```

FIGURE 8. Registering of a ball component example.

Systems are also possible to be registered. They are mostly needed to assemble and manage entities with the appropriate components. The system helps to separate the logic and behavior from the data and must hold the logic that belongs to the collection of entities. They are registered in the same way as components and scheme and callbacks, except that their set consists of *init()*, *play()*, *pause()*, *tick()* and *tock()*. Registered system prototypes can be accessed through *AFRAME.systems*. (36)

The A-Frame framework also provides a number of developer tools, such as 3D inspector, motion capture and graphical user interface (GUI) tools based on the framework. The 3D inspector visual tool allows editing a scene by moving around it and change, add and remove entities. A result of performed actions is seen in real time, and every entity can be copied as a code snippet. Motion capture, in its turn, brings an opportunity to record movements and actions, which perform in a WebVR application, and then replays the recording in order to develop the application without a manual continuous repeating of the same actions. This approach is handy for Quality Assurance (QA) testing as well because all console logs and bug reproduction steps are available. (37)

Based on all mentioned methods, it is possible to create a 3D platform with objects that represents a virtual basketball hall. Objects can be mapped with textures to make them look more realistic. A-Frame has an asset management system, which holds all assets at one place and improves performance by preloading and caching assets. A scene will not be loaded until a browser fetches all assets, and the loading will fail if some asset is not found during some defined timeout. Assets can store textures and object models, sounds and videos. Textures are images that are superimposed on a geometric surface. They are configured by *material* component properties such as dimensions, opacity, color and others. Ready 3D objects can be used with models. Each asset has own *ID* attribute by which it can be used in the scene. A path to the files is defined with a *src* attribute. (38)

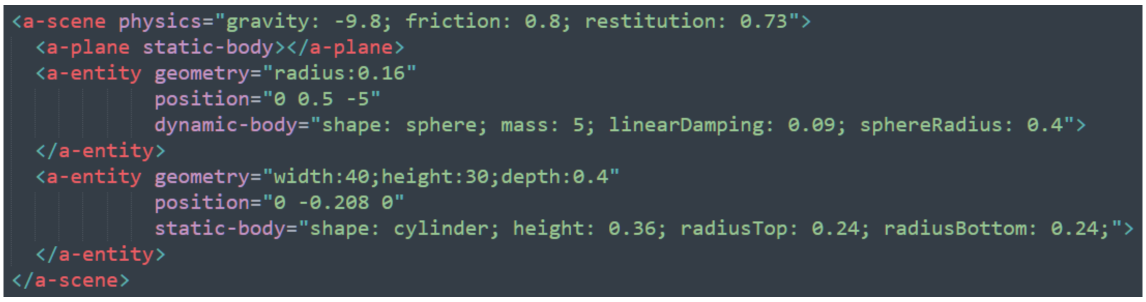
4.2 Adding physics

A physics engine is not included in the default A-Frame JavaScript build. It is separated and belongs to the additional extra features of the framework developed by the A-Frame team or community, which intend to add more functionality, primitives, components and tolls. The “aframe-physics-system” library is one of such extra features made by the A-Frame team. It is based on Cannon.js and adds physics to scene entities using body components like dynamic-body and static-body, and extends scene customization with gravity, friction, restitution, iterations and other additional properties. (39)

Cannon.js is an open source engine built originally on JavaScript and its main advantages over the rest are its compactness with regard to its performance and operability. It includes common physics features such as rigid-body dynamics, collision detection and calculation of the acceleration vector, and additionally such extra features as cloth simulation (40). A-frame combines own syntax with the physics API providing new components and properties and allowing to use the events of the library on the entities (39).

The entities, which have static-body or dynamic-body components, imply solidity relative to other objects, so they acquire a hard surface. The difference between them is that the static body has no effect from gravity and collisions. The dynamic body, on the contrary, can have a mass and a velocity vector, making it responsive in any applied force. In the demo application for this research, a basketball should be in the dynamic state in order to have motion acceleration and velocity, while remaining objects e.g. floor, walls and basket backboard are static and serve as physical obstacles for a ball. (39)

The dynamic-body and static-body components have a set of properties that consist of *shape*, *mass*, *linearDamping*, *angularDamping*, *sphereRadius* and *cylinderAxis*, from which the static-body component has only *shape* and *cylinderAxis*. Physics components HTML syntax can be seen in Figure 9. The *shape* defines a collision geometry of an object, depending on which one specific entity object behaves differently in collisions with dynamic entity objects. There are such available shapes as box, cylinder, sphere, hull (convex), mesh (custom geometry), auto and none. Sphere and cylinder shapes provide the best performance regarding collisions. The shape configured with a *none* property does not apply any collision geometry to the object. Other property, the *mass*, simulates a mass of an object and defines an impact force of an object in collisions. The *linearDamping* and *angularDamping* define a dynamic resistance of the object to movement and rotation. The *sphereRadius* and *cylinderAxis* work only in cases of sphere and cylinder shapes respectively, and define a restrictive radius of a collision geometry of the sphere and axis pf the cylinder. (39)



```
<a-scene physics="gravity: -9.8; friction: 0.8; restitution: 0.73">
  <a-plane static-body></a-plane>
  <a-entity geometry="radius:0.16"
            position="0 0.5 -5"
            dynamic-body="shape: sphere; mass: 5; linearDamping: 0.09; sphereRadius: 0.4">
  </a-entity>
  <a-entity geometry="width:40;height:30;depth:0.4"
            position="0 -0.208 0"
            static-body="shape: cylinder; height: 0.36; radiusTop: 0.24; radiusBottom: 0.24;">
  </a-entity>
</a-scene>
```

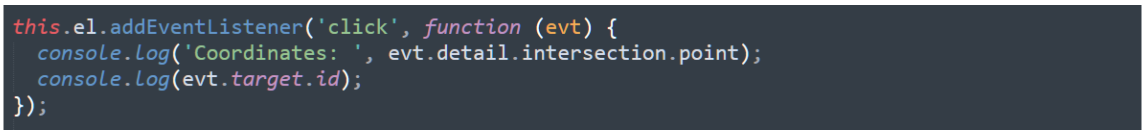
FIGURE 9. Example of physics components and properties.

Another library called “aframe-extras” provides a kinematic-body component, which is another state of the physical body of an entity. This kinematic state is similar to static, though it can collide with other static and dynamic objects. It does not take into account the reasons of object motion such as mass, impact force or applied impulse. That component is mainly used for a camera entity, more precisely as a “rig” of the camera, and it has one *radius* property defining the radius of camera boundaries representing collision geometry. In this way, the camera cannot go through the objects, limiting the movements of a player around the scene and simulating a virtual room-scale. (41)

4.3 Configuring events, actions and controls

Scene events on entities and between them are handled in usual way through an *addEventListener()* method with a common set of events such as *click*, *fusing*, *mouseenter*, *mouseleave*, *keydown*, *keyup*, *touchstart*, *touchend*. The example can be seen in Figure 10 below. Since some events relate to a cursor, there is an obvious question about the availability of such in the WebVR application. To define a cursor in A-Frame and enable an emitting mouse and hover events, the camera entity has to contain an additional entity inside with a *cursor* component attribute. The cursor can take different forms depending on geometry and material properties of components entity. A position of the cursor can also be overridden, for example to put it farther or closer from a viewpoint of a camera and keep it in a visible area. In case if HMD is used in combination with VR hand controllers, those are also using the *addEventListener()* method for interacting with a 3D environment. (42)

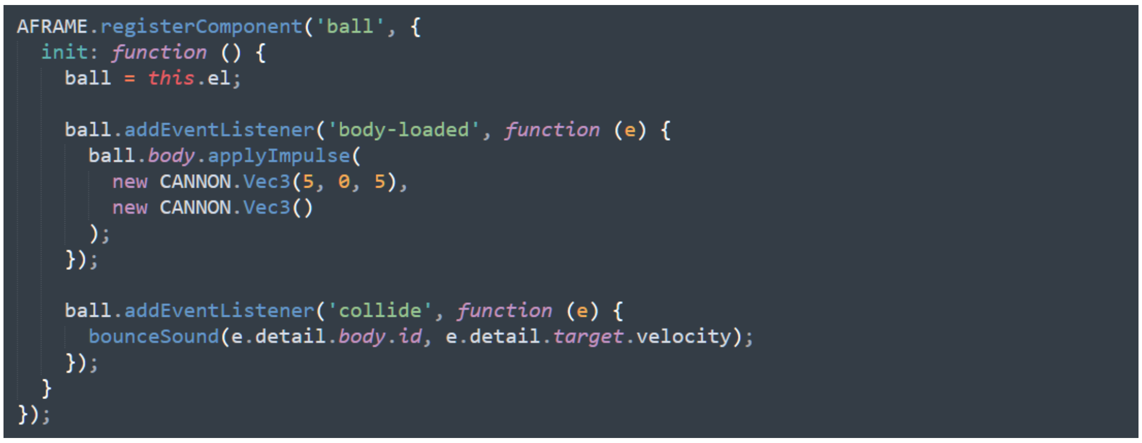
The *cursor* is based on a raycaster component. The raycaster is a Three.js method, which draws a line from a defined point to some direction and detects if the line intersects entities. Thus, the cursor represents a ray that is tending from the camera towards the user’s gaze, and the raycaster component application emits a mouse event when the ray hits into an object (43). In A-Frame, every raycaster event can contain event related details with a *detail.intersection* data attribute of events property. The *distance* attribute stores the value of a distance between the cursor and the used entity. The *point* attribute shows the coordinates of a ray point intersection in a 3D world coordinate system. Intersection faces, their index and indices of vertices comprising of intersected faces are available with *faces*, *faceIndex* and *indices* attributes. A full data set of the intersected objects is gathered in the *object* attribute. The correspondence between coordinates on the surface of a three-dimensional object and coordinates on the texture are stored in the *uv* attribute. (42)



```
this.el.addEventListener('click', function (evt) {
  console.log('Coordinates: ', evt.detail.intersection.point);
  console.log(evt.target.id);
});
```

FIGURE 10. Example of cursor click event usage. The “evt.target” attribute is equal to “evt.detail.intersection.object”.

To apply forces to an object or listen to a physics specific event, it is possible to use a related Cannon.js API library as can be seen in Figure 11. The *body-loaded* and *collide* events are available to listen for a moment when a targeting component is loaded and collided with other physical body components. The collision even provides a motion vector, velocity in the moment of a collision and collided entities data. The forces appliance performed by Cannon.js methods, for instance *applyImpulse()*, which sets two 3-dimensional vectors to an entities body physical object using a *Vec3(x, y, z)* constructor, where *x*, *y* and *z* represent the dimensional coordinates of a scene. The first vector represents a direction and acceleration force, and the second vector is responsible for rotation. (39)



```
AFRAME.registerComponent('ball', {
  init: function () {
    ball = this.el;

    ball.addEventListener('body-loaded', function (e) {
      ball.body.applyImpulse(
        new CANNON.Vec3(5, 0, 5),
        new CANNON.Vec3()
      );
    });

    ball.addEventListener('collide', function (e) {
      bounceSound(e.detail.body.id, e.detail.target.velocity);
    });
  }
});
```

FIGURE 11. Example of usage of “body-loaded” and “collide” event listeners, and applying impulse on component with a “ball” unique identifier.

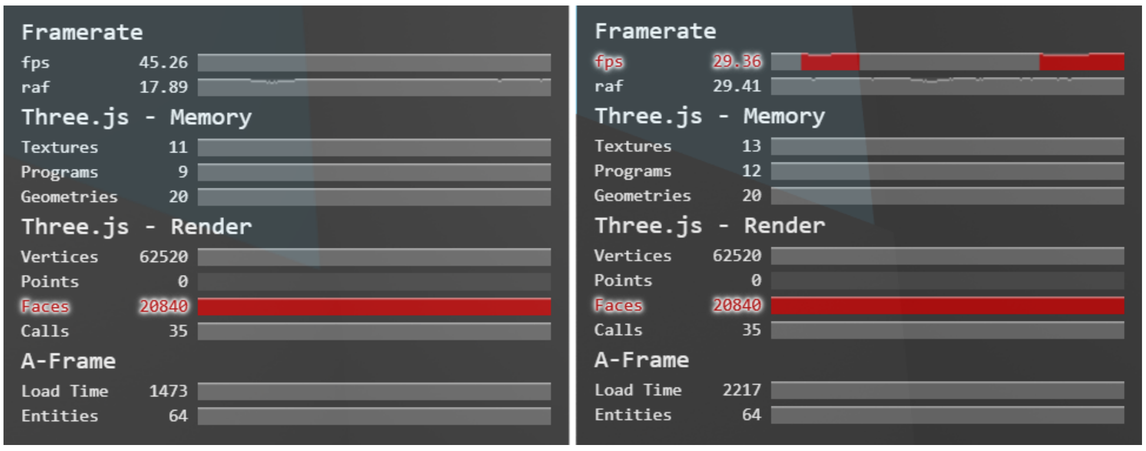
In this way, using the *cursor* component and physic engine API, implementation of a ball related actions, such as picking it up or throwing can be handled with the few events. To modify the properties of the ball, it is possible to use *setAttribute()* or *removeAttribute()* methods. Such methods that relate to entity configurations are used on the entities, while methods related to Three.js *Object3D* representation are used on entity objects controlling position, velocity and rotation.

Moving in a virtual environment ability is configured by components related to the camera and its “rig” entities. A-Frame supports a number or additional components, allowing developers to use all the necessary tools to control a player in the game world. The *look-controls* component enable the camera moving with a use of mouse or devise accelerometer sensors. The *wasd-controls* adds support to use keyboard layout of common controlling keys to move the camera entity (44). Extra framework libraries provide a wider range of controls options, the *movement-controls* component for instance, which combines locomotion controls of *keyboard-controls*, *touch-controls*, *trackpad-controls*, and most important *gamepad-controls* that enables an appliance of a gamepad. With its help, any gamepad, joystick and controller could be used in the A-Frame application. (45)

The *gamepad-controls* component is using Gamepad API, which allows applying a connected gamepad using JavaScript. The *getGamepads*() method is used to get available gamepads. It is used in a user agent interface, which is same as in case with WebVR API – the *navigator*. Each gamepad returned by *getGamepads()* have objects, such as *axes*, *buttons*, *connected*, *ID*, *index*, *mapping* and *timestamp*. Defining connectivity is done by the *connected* object, which is the Boolean data type and stores true if a gamepad is connected. The *axes* object stores an array of numbers that represent the state of each analogue stick or button. Numbers are a floating point value in the range from -1.0 to 1.0. The *buttons* object is an array representing the buttons that are present on a used gamepad. Each button has a *pressed* and a *value* property, where the first one is a Boolean and belongs to buttons, and the second one is a floating point value and belongs to analog buttons, such as triggers. The *mapping* represents a layout of the gamepad, which can be defined by a browser if it is empty. The *timestamp* stores the last time when the data for the used gamepad was updated, for example if *axes* or *buttons* data values change. The *ID* object contains some information about the controller, such as a USB vendor ID, a product ID and a name. The *index* holds a unique integer data type value for every connected gamepad. (46)

5.4 Light and shadows

The light entities are similar to other A-Frame entities. They have components and properties and can be extended and overridden in JS. However, without a light, there is nothing to be seen in the scene, only a black screen. The light makes the scene visible. There are few light options provided by the framework: *ambient*, *directional*, *point*, *spot* and *hemisphere*. Each option produces different kind of a light, derived from a related option type name. The light itself does not greatly affect the performance of the application, but it could become very resource demanding, if shadow casting is enabled. Three light types *spot*, *point* and *directional* can produce shadows by configuring *light* component properties *castShadow*, *shadowMapHeight* and *shadowMapWidth*. The last two properties define a resolution of a shadow in pixels. The entities have own *cast* property of a *shadow* component that is responsible for controlling shadow casting of a used entity. The application has to calculate the geometry boundaries of an each entity object with a *cast* property set to true, located in the coverage area of a light. The more objects casting shadows, the more processing power is needed. The *directional* light type provides the best performance compared to other types, but the FPS value still drops significantly on around 15 frames in average, which entails a lower user experience. Figure 12 below demonstrates results in performance with and without shadows. (47)



```
Framerate
fps           45.26
raf           17.89
Three.js - Memory
Textures         11
Programs          9
Geometries       20
Three.js - Render
Vertices      62520
Points            0
Faces         20840
Calls            35
A-Frame
Load Time      1473
Entities         64
```

```
Framerate
fps           29.36
raf           29.41
Three.js - Memory
Textures         13
Programs         12
Geometries       20
Three.js - Render
Vertices      62520
Points            0
Faces         20840
Calls            35
A-Frame
Load Time      2217
Entities         64
```

FIGURE 12. Comparison of application performance statistics without (left table) and with (right table) shadows casting from two directional light sources.

5 CONCLUSION

WebVR is an interesting concept, a powerful tool providing many opportunities to developers. The technology definitely brings a desired simplicity, since any user is able to experience WebVR applications in a moment by only opening them in the browser on a computer or smartphone. The API library gets developing, its functionality keeps extending and improving. It provides a wide set of tools that allow developers to be flexible in creativity. Using Three.js and VR-oriented frameworks, it is possible to create a WebVR application without having deep knowledge in HTML and JavaScript. However, web-based VR is unpopular and known mostly in the circles of people who work in the VR marketing and developing fields.

The VR and AR market itself is growing each year with increasing sales of related software and hardware. It attracts more individual consumers of different age, because more device and application options become available on the market. The enterprises are interested in these technologies as well as individuals. VR is integrated in many areas such as movies, sports broadcasts and shows, social networks, education, medicine, trade and real estate. According to data from the Digi-Capital consulting and analytics company, investments in spheres of mixed reality technology appliances amounted to $3 billion in 2017. The annual worldwide revenue from the sale of games, applications and devices for 2018 year was over $20 billion as can be seen in charts in Appendix 1. Concerning the future, the VR/AR marketing field growth has a good forecast for the upcoming 3-4 years.

Considering that WebVR is a part of the VR/AR market, its development is inevitable. For now, it is an experimental technology and does not carry any revolutionary value. As the idea involves a simplification of production and use of content, it is going to take its place in the future of VR marketing field. Presently, it is possible to assume that WebVR will be mainly used for special-purpose applications such as virtual real estate or sale of tourist services. Building a VR application in the web requires less time and financial resources than native application development, thus it is also suitable for mocking up a VR application for testing. WebVR cannot be considered necessary and useful at this point, but it just needs a bit more time to unleash its potential.

The current thesis work can be considered as a successful evaluation of WebVR API usage and application operability based on it. During the work, the topic was studied gradually as the application was developed. The nuances of the technologies were studied deeply and described in details. The gained experience was very valuable from the programming skills and theoretical skills developing perspectives. The developed demo application shows capabilities of web-based VR and gives an idea of its current status in the field of entertainment and other possible marketing spheres.

REFERENCES

(1) Wikipedia. WebVR. Date of retrieval: 23.8.2018
https://en.wikipedia.org/wiki/WebVR

(2) Kidwell Essa. Everything you need to know about WebVR. Date of retrieval: 23.8.2018
https://www.windowscentral.com/everything-you-need-know-about-webvr

(3) Wade Nicholas J. 2002. “Perception, volume 31”, pages 265-272. Date of retrieval: 12.5.2019
journals.sagepub.com/doi/pdf/10.1068/p3103ed

(4) Barnard Dom. 2017. “History of VR - Timeline of Events and Tech Development”. Date of retrieval: 5.9.2019
https://virtualspeech.com/blog/history-of-vr

(5) Ware Justin. 2018. “Virtual reality: a (very) brief history, part 1”. Date of retrieval: 12.5.2019
finfeed.com/opinion/ctrl-alt-del/virtual-reality-very-brief-history-part-1/

(6) Heilig Morton L. 1969. “Experience theatre patent”. Date of retrieval: 11.5.2019
www.mortonheilig.com/Experience_Theater_Patent.pdf

(7) Parkin Simon. 2014. “Virtual Reality Startups. Look Back to the Future”. Date of retrieval: 10.5.2019
www.technologyreview.com/s/525301/virtual-reality-startups-look-back-to-the-future/

(8) Corning Anne. 2018. “Blast From the Past: Virtual Reality”. Date of retrieval: 10.5.2019
https://www.radiantvisionsystems.com/blog/blast-past-virtual-reality

(9) Lowood Henry E. 1998. ”Virtual reality”. Date of retrieval: 5.9.2019
www.britannica.com/technology/virtual-reality

(10) Vincent James. 2018. ”Haptic feedback is making VR surgery feel like the real thing”. Date of retrieval: 7.9.2019
https://www.theverge.com/2018/8/14/17670304/virtual-reality-surgery-training-haptic-feedback-fundamentalvr

(11) Shilova Margarita. 2017. “Virtual reality technology: main trends, statistics & startups”. Date of retrieval: 25.8.2018
https://apiumhub.com/tech-blog-barcelona/virtual-reality-technology/

(12) Sheikh Knvul. 2016. ”10 Other Fascinating Uses for Virtual-Reality Tech”. Date of retrieval: 25.8.2018
www.livescience.com/53392-virtual-reality-tech-uses-beyond-gaming.html

(13) Gardonio Scottie. 2017. “Retailers Beware: AR/VR is Coming to Change Shopping”. Date of retrieval: 4.05.2019
https://medium.com/iotforall/retailers-beware-ar-vr-is-coming-to-change-shopping-d207f9e5be63

(14) Mikhalchuk Dimitri. 2017. “VR Gaming: How Industry Will Propel the Technology”. Date of retrieval: 4.5.2019
https://teslasuit.io/blog/virtual-reality/vr-gaming-how-industry-push-technology/

(15) Iwaniuk Phil. 2016. “VR is 7 times more demanding of your PC than 1080p gaming, say NVIDIA”. Date of retrieval: 9.9.2018
https://www.pcgamesn.com/eve-valkyrie/vr-requires-a-pc-7-times-more-powerful-than-1080p-gaming-say-nvidia

(16) Program-Ace. 2018. “How Mobile VR Is Going to Change Gaming Market”. Date of retrieval: 9.9.2018
https://dev.to/program_ace_ltd/how-mobile-vr-is-going-to-change-gaming-market-5c8o

(17) MiddleVR. “The challenges of creating a VR application”. Date of retrieval: 2.9.2018
https://www.middlevr.com/resources/the-challenges-of-creating-a-vr-application/

(18) Krishna Sai V. K. 2018. “WebVR is still not easy. What if we could change that?”. Date of retrieval: 14.12.2018
https://medium.com/scapic/webvr-is-still-not-easy-what-if-we-could-change-that-a1c1b5afd1ea

(19) Developers. “Introduction to the WebVR API”. Date of retrieval: 14.12.2018
https://developer.oculus.com/documentation/oculus-browser/latest/concepts/browser-webvr-api/

(20) Tavares Gregg. “WebGL, how it works”. Date of retrieval: 15.12.2018
https://webglfundamentals.org/webgl/lessons/webgl-how-it-works.html

(21) Google developers. 2019. “Getting Started with WebVR”. Date of retrieval: 14.12.2018
https://developers.google.com/web/fundamentals/vr/getting-started-with-webvr/

(22) Can I Use. WebVR browsers support table. Date of retrieval: 17.11.2018
https://www.caniuse.com/#search=WebVR

(23) WebVR info for developers. “Bringing Virtual Reality to the Web”. Date of retrieval: 17.11.2018
https://webvr.info/developers/

(24) Wikipedia. Polyfill. Date of retrieval: 17.11.2018
https://en.wikipedia.org/wiki/Polyfill_(programming)

(25) Bozorgzadeh Amir. 2018. “VR browsers are key to a more immersive web”. Date of retrieval: 16.11.2018
www.venturebeat.com/2018/02/13/vr-browsers-are-key-to-a-more-immersive-web/

(26) Terdiman Daniel. 2015. “Oculus Super Bowl party could be the future of social sports broadcasting”. Date of retrieval: 16.11.2018
https://venturebeat.com/2015/01/30/oculus-super-bowl-party-could-be-the-future-of-social-sports-broadcasting/

(27) Siebert Charles, Berlin Branden, Song Yipeng. 2017. “Optimizing Performance of A-Frame Scenes for Mobile Devices”. Date of retrieval: 30.3.2019
hacks.mozilla.org/2017/07/optimizing-performance-of-a-frame-scenes-for-mobile-devices/

(28) Marinacci Josh. 2018. “Performance-Tuning a WebVR Game”. Date of retrieval: 30.3.2019
https://hacks.mozilla.org/2018/09/performance-tuning-webvr-game/

(29) Github. 2017. WebVR Editor’s Draft. Date of retrieval: 8.2.2019
https://immersive-web.github.io/webvr/spec/1.1/

(30) MDN web docs. 2019. WebVR API. Date of retrieval: 20.10.2018
developer.mozilla.org/en-US/docs/Web/API/WebVR_API

(31) VR Software Wiki. 2018. “WebVR comparison: A-Frame vs. React-360”. Date of retrieval: 24.10.2018
https://sites.google.com/view/brown-vr-sw-review-2018/vr-development-software/comparisons/webvr-comparison-a-frame-vs-react-360

(32) A-Frame. Entity-Component-System. Date of retrieval: 8.9.2018
www.aframe.io/docs/0.8.0/introduction/entity-component-system.html

(33) A-Frame. Building a Basic Scene. Date of retrieval: 26.8.2018
www.aframe.io/docs/0.8.0/core/scene.html

(34) A-Frame. HTML & Primitives. Date of retrieval: 8.9.2018
www.aframe.io/docs/0.8.0/introduction/html-and-primitives.html

(35) A-Frame. Component. Date of retrieval: 2.11.2019
www.aframe.io/docs/0.8.0/core/component.html

(36) A-Frame. System. Date of retrieval: 2.2.2019
www.aframe.io/docs/0.8.0/core/systems.html

(37) A-Frame. Visual Inspector & Dev Tools. Date of retrieval: 26.8.2018
www.aframe.io/docs/0.8.0/introduction/visual-inspector-and-dev-tools.html

(38) A-Frame. Asset Management System. Date of retrieval: 2.2.2019
www.aframe.io/docs/0.8.0/core/asset-management-system.html\

(39) Github. Physics for A-Frame VR. Date of retrieval: 18.11.2018
https://github.com/donmccurdy/aframe-physics-system

(40) Wikipedia. Cannon.js. Date of retrieval: 24.10.2018
https://en.wikipedia.org/wiki/Cannon.js

(41) Github. A-Frame Extras. Date of retrieval: 12.01.2019
https://github.com/donmccurdy/aframe-extras

(42) A-Frame. Interactions & Controllers. Date of retrieval: 25.1.2019
www.aframe.io/docs/0.8.0/introduction/interactions-and-controllers.html

(43) A-Frame. Raycaster. Date of retrieval: 25.1.2019
www.aframe.io/docs/0.8.0/components/raycaster.html

(44) A-Frame. Camera. Date of retrieval: 14.10.2018

www.aframe.io/docs/0.8.0/components/camera.html

(45) Github. A-Frame Extras, controls. Date of retrieval: 12.01.2019
https://github.com/donmccurdy/aframe-extras/tree/master/src/controls

(46) Walter Charlie. 2015. “Using The Gamepad API In Web Games”. Date of retrieval: 2.3.2019
www.smashingmagazine.com/2015/11/gamepad-api-in-web-games/

(47) A-Frame. Light. Date of retrieval: 25.1.2019
https://aframe.io/docs/0.8.0/components/light.html

(48) Digi-Capital. VR/AR analytics. Date of retrieval: 13.4.2019
https://www.digi-capital.com/news/2018/01/record-over-3b-ar-vr-investment-in-2017-1-5b-in-q4/

(49) Merel Tim. 2017. “The reality of VR/AR growth”. Date of retrieval: 14.5.2019
https://techcrunch.com/2017/01/11/the-reality-of-vrar-growth/

APPENDIX

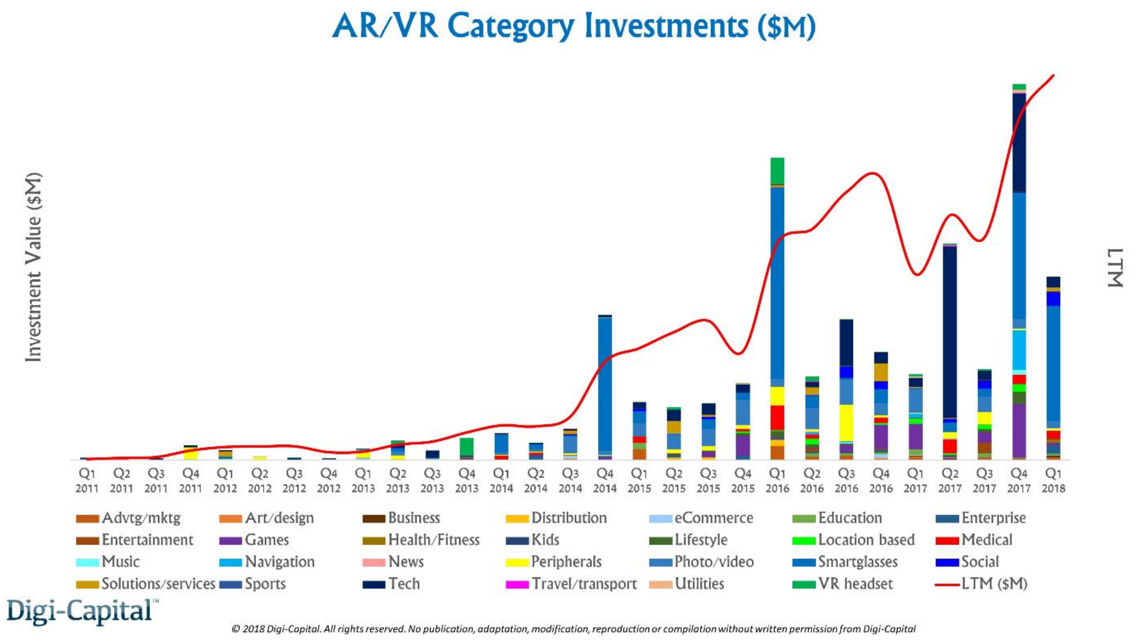


Chart 1. VR/AR investments in 2011-2018 period. (48)

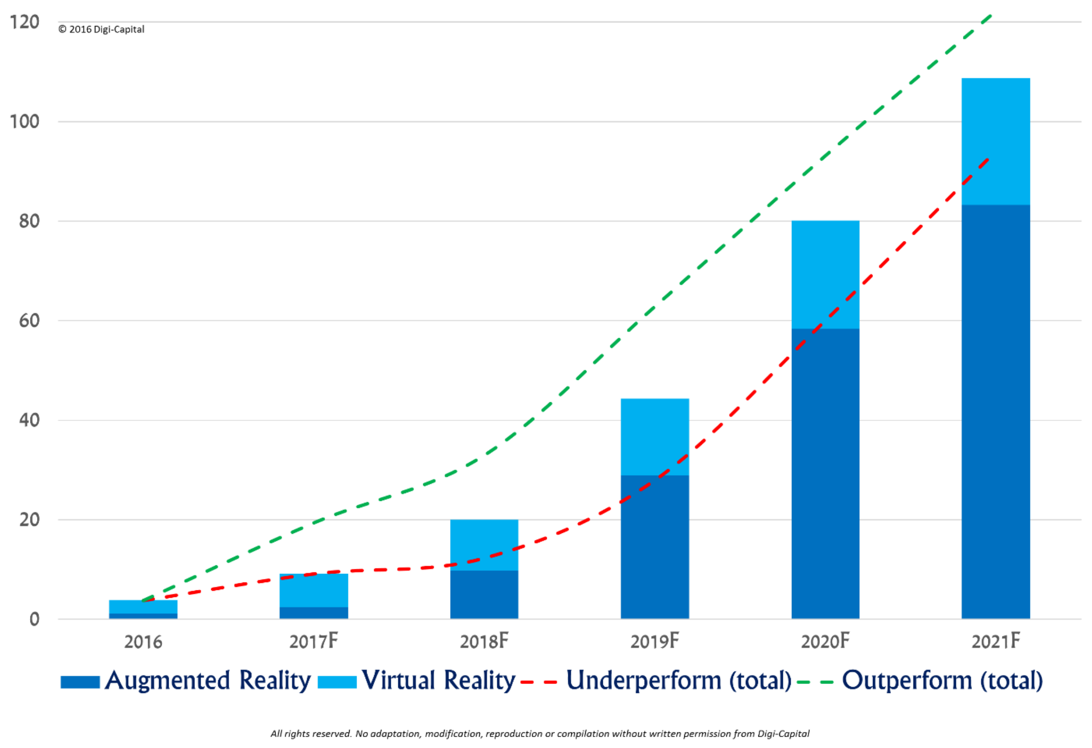


Chart 2. VR/AR revenue ($B). (49)