Technical University of Košice
Faculty of Mining, Ecology, Process Control and Geotechnologies

Application of Virtual and Augmented Reality in Education

Master's Thesis

2017 **Bc. Michal Takáč**

Technical University of Košice
Faculty of Mining, Ecology, Process Control and Geotechnologies

Application of Virtual and Augmented Reality in Education

Master's Thesis

Study Programme:	Informatization of Processes of Raw Materials Extraction and Processing
Field of study:	Recovering and processing of earth resources
Department:	Institute of Control and Informatization of Production Processes (ÚRaIVP)
Supervisor:	RNDr. Andrea Mojžišová, PhD.
Consultant(s):	RNDr. Jana Pócsová, PhD.

Košice 2017 **Bc. Michal Takáč**

Abstract

In this thesis we set to explore the possibilities of virtual and augmented reality (VR/AR) through the use of modern web technologies and development practices with the goal of finding novel approaches and applications in higher education with focus on mathematics. First a brief history of VR and AR is described followed by overview regarding the current problems and use cases of VR and AR systems and overview of current commercially available headsets. Then the novel, experimental Graphical User Interface (GUI) is presented, utilizing six degrees of freedom in room-scale virtual world within web page with use of interactions with VR hand controlelrs to provide strong visual representations and visualizations of parametrized functions, helping students to approach learning differently and understand difficult math concepts faster. In the end of thesis we propose future possibilities and how the technology could shape student's knowledge.

Keywords

Virtual reality, Augmented reality, Education, Mathematics

Assign Thesis

Namiesto tejto strany vložte naskenované zadanie úlohy. Odporúčame skenovať s rozlíšením 200 až 300 dpi, čierno-bielo! V jednej vytlačenej ZP musí byť vložený originál zadávacieho listu!

Declaration

I hereby declare that this thesis is my own work and effort. Where other sources of information have been used, they have been acknowledged.

Košice, April 21, 2017 .....

Signature

Acknowledgement

I would like to express my sincere thanks to RNDr Andrea Mojžišová, PhD, the main Supervisor, for her constant and constructive guidance throughout the study, valuable feedback and brainstorming sessions. Special mention should go to RNDr Jana Pócsová, PhD. for her interest in new approaches to teaching and her ability to listen and give feedback to novel ideas when others are not interested and also to prof. Steven Abbott for helping me out with Oculus Touch support. Thanks to *Sleighdogs, GmbH* for sponsoring the project. To all other who gave a hand, I say thank you very much.

Preface

Mathematical knowledge is often fundamental when solving real life problems. Especially, problems situated in the three-dimensional domain that require spatial skills are sometimes hard to understand for students. Many students have difficulties with spatial imagination and lack spatial abilities. Recently, a number of training studies have shown the usefulness of virtual reality in training spatial ability. Therefore I set myself a goal to find inresections between virtual reality and higher education by building experimental virtual user interface(s) and testing them in educational environments, e.g. colleges and universities. In this thesis I'll focus on mathematics.

Contents

List of Figures

List of Terms

DoF Degrees of freedom (6DoF = six degrees of freedom)

VR Virtual Reality

WebVR Virtual Reality on the web

API Application Programming Interface

ES6 ECMAScript version 6

DOM Document Object Model

HTML Hyper Text Markup Language

WebGL Web-based Graphics Library

HMD Head-mounted display

VLE Virtual Learning Environment

Listings

Introduction

In this thesis we're introducing novel learning environment with the use of virtual reality on the web that can be used as possible way of teaching mathematics.

Our experimental project is called MathworldVR, which sets to explore the possibilities and introduce novel methods of using web technologies for creating room-scale, immersive learning environment in virtual reality for helping students to explore, learn about and experiment with various parametrized functions. It's also a practical tool for teachers to showcase abstract concepts in concrete 3D environment during lectures.

Thesis starts with the "Problem expression" section, where we describe our goal and methodology of how we're approaching the development of MathworldVR. In "Analysis of current state" section, current state of virtual reality is described with introduction to the technology, examples of different VR applications, what development platform and tools are used for their development and advantages of VR on the web, called WebVR in short.

In "Technologies and tools for development", we provide detailed information about technologies used for building the MathworldVR application. the selection of technologies has it's meaning - all of them are used in modern web development by big companies and/or was created by them. All of them are also open-source, because MathworldVR will be open-sourced as well.

Design and functionality planning is explained in "Conceptualisation and design" section, describing what decisions were made when architecting and designing MathworldVR and what functionality user can expect.

In "Implementation of virtual reality application", project setup, development process, testing and deployment is explained with detailed description of individual

components that together makes the MathworldVR experience possible.

Findings and future project roadmap is discussed in "Conclusion" section.

1 Problem expression

The main goal of master's thesis with the name "Application of Virtual and Augmented Reality in Education" is to conceptualize, design and implement a virtual learning environment (VLE) for higher education, primarily focused on mathematics. This environment will be available on the web in form of client-side, web application and accessible through desktop and mobile web browser. Main emphasis will be put into desktop platform, since currently, it's the only platform that supports interactive VR capabilities of six degrees of freedom, head-mounted display (HMD) and hand controllers. We're specificaly focused on consumer version of HTC Vive (HMD) since we have one at our disposal and on which we'll be testing the implementation of application.

1.1 Methodology of the thesiss

Development of MathworldVR web application will include these steps:

- At the beginning it's necessarry to become familiar with current state of VR support on the web and WebVR frameworks. This will require us to study the available WebVR API and it's state of implementation in modern, popular browsers.
- After acknowledgement of limitations and possibilities of web browsers we can proceed to conceptualization and design. We need to define the functionality of MathworldVR, criteria for application and select the approach to development.
- Then we can proceed to selecting a WebVR framework.
- Before start of the development, we need select the technologies. MathworldVR will be client-side WebVR application and since WebVR is a novel

technology, we need to look into modern development tools and approaches.

- After selecting the development tools we need to prepare the development PC for work by installing Node.js, NPM, initialize the project with NPM, install needed development tools and project dependencies in form of Node modules. For our needs, we'll go with code editor/IDE "VSCode" by Microsoft, which includes tools for efficient programming and static code analysis.

- MathworldVR will be deployed to web server and tested manually in different web browsers that support WebVR API with use of HTC Vive HMD to ensure compatibility and good user experience.

- In the end, MathworldVR is tested privately by students at Technical University in Košice and lecturers from the department of mathematics at FBERG.

2 Analysis of current state

Undoubtedly, VR has attracted a lot of interest of people in the last few years. Being a new paradigm of user interface it offers great benefits in many application areas. It provides an easy, powerful and intuitive way of human-computer interaction. the user can watch and manipulate the simulated environment in the same way we act in the real world, without any need to learn how the complicated (and often clumsy) user interface works. Therefore many applications like flight simulators, architectural walkthrough or data visualization systems were developed relatively fast. Later on, VR has was applied as a teleoperating and collaborative medium, and also in the entertainment area, which now became de-facto the main interest of VR developers and emerging VR companies.

One can say that virtual reality established itself in many disciplines of human activities, as a medium that allows easier perception of data or natural phenomena appearance. Therefore the education purposes seem to be the most natural ones. the intuitive presentation of construction rules (virtual Lego-set), visiting a virtual museum, virtual painting studio or virtual music playing (Loeffler, 1995) are just a few examples of possible applications.

Virtual environments are inherently three-dimensional. They can provide interactive playgrounds with a degree of interactivity that goes far beyond what is possible in reality. If using VR as a tool for mathematics education, it ideally offers an added benefit to learning in a wide range of mathematical domains (Kaufmann, 2011).

In recent months, many VR applications in the education area started to emerge, notable one being *The Body VR*. It is an educational VR experience that takes the user inside the human body. User travels through the bloodstream and discovers how blood cells work to spread oxygen throughout the body or learn how the organelles work together to fight deadly viruses.

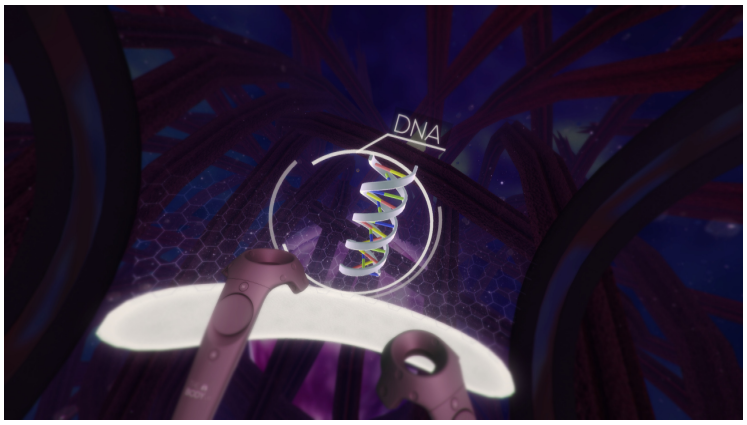

Figure 2 – 1 The Body VR. Steam, The Body VR (2017)

Most popular VR game nowadays is *Job Simulator*, an experience for HTC Vive, taking place in a virtual world where robots have replaced all human jobs. Game's purpose is to teach, in a funny way, what it was like to have a job in the past. It is the most profitable VR experience on the market, grossing more than $3 million in sales. thesixthaxis.com (2017)

Figure 2 – 2 Job simulator. Steam, Job Simulator (2017)

In field of mathematics, VR application named *Calcflow* is making serious progress. It's developed by *Nanome, Inc.* - a company that was started by students of *University of California San Diego* and stays affiliated with it. *Calcflow*'s features include visualizations of vector addition, cross product, parametrized functions, mobius strip, spherical coordinate mapping, double and surface integrals.

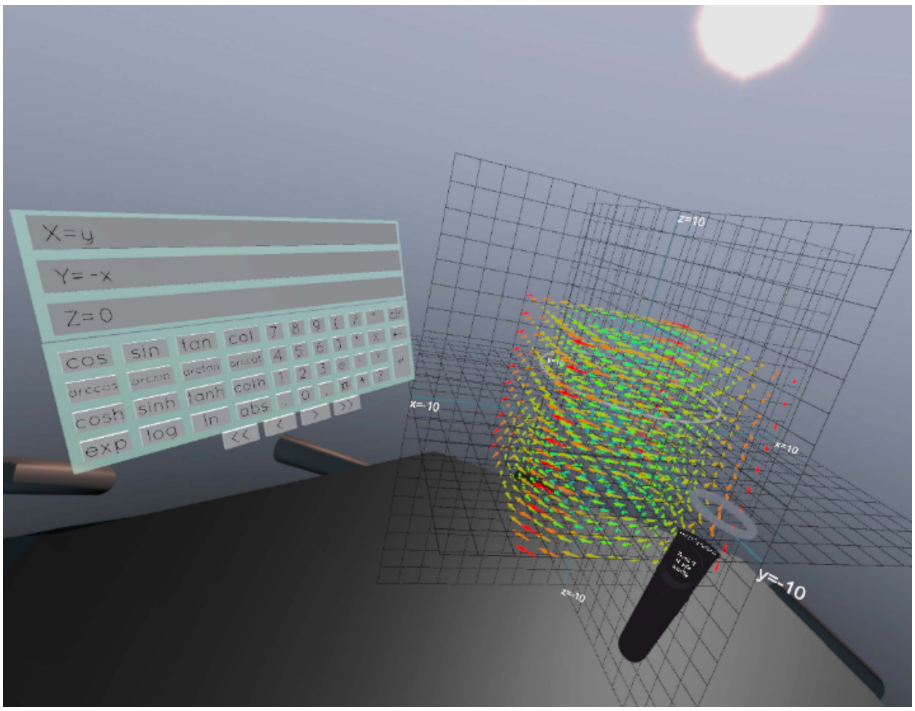

Figure 2 – 3 Calcflow. Steam, Calcflow (2017)

2.1 Building virtual reality applications and experiences

The leading platform for building VR experiences today is the game engine Unity, both because the company had the foresight to add support for the Oculus Rift development kit early on, but also simply because the early use cases from when Oculus Rift was still just a very successful Kickstarter project centered around video games.

Unreal Engine 4 is a complete suite of development tools made for anyone working

with real-time technology. From enterprise applications and cinematic experiences to high-quality games across PC, console, mobile, VR and AR, Unreal Engine 4 gives developers everything they need to implement and deploy their games and experiences. Unreal Engine (2017)

2.2 Virtual reality on the web

WebVR provides support for exposing virtual reality devices — for example head-mounted displays like the HTC Vive or Oculus Rift — to web apps, enabling developers to translate position and movement information from the display into movement around a 3D scene in browser. As of today, support for both head-mounted displays is available in experimental or development builds of Chrome and Firefox, with official release planned for third quarter of 2017. This has numerous very interesting applications, from virtual product tours and interactive training apps to immersive first person games. Unity game engine, for instance, is able to make native builds for all major platforms from the same code base, including PC, Mac, Linux, iOS, Android and more. When made by professionals, such native builds will undoubtedly look better and run faster than a comparable VR experience built with WebGL and WebVR (at least AAA games or other experiences where high fidelity and performance are paramount).

The major advantage of WebVR over natively built experiences is the same as the web has always had over desktop apps and mobile apps today - no need to download and install anything. User just needs to click a link, type in a url, and the application runs directly in her browser. There's no app store needed. Web developers can also take advantage of many open source libraries available on the internet.

3 Technologies and tools for development

3.1 JavaScript

JavaScript is an interpreted programming language with object-oriented (OO) capabilities. Syntactically, the core JavaScript language resembles C, C++ and Java, with programming constructs such as the if statement, the while loop, and the operator. the similarity ends with the syntactic resemblance, however. JavaScript is a loosly typed language, which means that variables do not need to have a type specified. Object in JavaScript map property names to arbitrary property values. In this way they are more like hash tables or associative arrays (in Perl) then they are structs (in C) or objects (in C++ or Java). the OO inheritance mechanism of JavaScript is prototype-based. This is different from inheritance in C++ and Java. Flanagan (2006)

JavaScript is mostly used in web browsers. It allows scripts to interact with the user, control the web browser and alter the document content that appears within the web browser window. This embedded version of JavaScript runs scripts embedded within HTML web pages. It's called client-side JavaScript to distinguish them from scripts that run on the server, since JavaScript can also be used on server-side with Node.js. Flanagan (2006)

JavaScript programming language is standardized in the ECMA-262 specification and ISO/IEC 16262 by standards organization European Computer Manufacturers Association (ECMA) and is often refered to as ECMAScript. Most popular browsers currently supports ECMAScript in version 5 (ES5). ECMAScript 6 (ES6) is the sixth version seventh edition of ECMAScript language which was introduced to improve JavaScript and ensure that developers no longer needed to use abstractions or other techniques to write quality code.Narayan (2015)

3.2 NPM

NPM is the package manager for JavaScript and the world's largest software registry, which makes it easy for JavaScript developers to share the code that they've created to solve particular problems, and for other developers to reuse that code in their own applications.NPM (2017)

The bits of reusable code are called packages, or sometimes modules. A package is just a directory with one or more files in it, that also has a file called "package.json" with some metadata about this package. A typical application, such as a website, will depend on dozens or hundreds of packages. These packages are often small. the general idea is to create a small building block which solves one problem and solves it well. This makes it possible for developers to compose larger, custom solutions out of these small, shared building blocks.NPM (2017)

There's lots of benefits to this. It makes it possible for development teams to draw on expertise outside of the organization they are working in by bringing in packages from people who have focused on particular problem areas. But even if they don't reuse code from people outside of the organization, using this kind of module based approach can actually help the team work together better, and can also make it possible to reuse code across projects.NPM (2017)

We can find packages to help us build our application by browsing the npm website. When we're browsing the website, we'll find different kinds of packages and node modules. NPM started as the node package manager, so we'll find lots of modules which can be used on the server side. There are also packages which add commands for us to use in the command line.NPM (2017) And there are a number of packages which can be used in the browser, on the front end, which we're going to use in our project, MathworldVR.

3.3 Three.js

Three.js is cross-browser JavaScript library/API used to create and display animated 3D computer graphics in a web browser. It uses WebGL.

3.4 A-Frame

A-Frame is a web framework for building virtual reality experiences. It was started by Mozilla to make WebVR content creation easier, faster, and more accessible. A-Frame lets you build scenes with just HTML while having unlimited access to JavaScript, Three.js, and all existing Web APIs. It uses an entity-component-system pattern that promotes composition and extensibility. It is free and open source with growing community and a ecosystem of tools and components. A-Frame (2017)

There are no build steps or boilerplate required to install, we just need an HTML file:

```
1  <html>
2    <head>
3      <script src="https://aframe.io/releases/0.5.0/aframe.min.js"
           ></script>
4    </head>
5    <body>
6      <a-scene>
7        <a-box color="#6173F4" opacity="0.8" depth="2"></a-box>
8        <a-sphere radius="2" src="texture.png" position="1 1 0"></
             a-sphere>
9        <a-sky color="#ECECEC"></a-sky>
10     </a-scene>
11   </body>
12 </html>
```

`<a-scene>` contains all of the objects in 3D scene. It also handles all of the setup

that is traditionally required for 3D: setting up WebGL, the canvas, camera, lights, renderer, render loop as well as out of the box VR support on platforms such as HTC Vive, Oculus Rift, Samsung GearVR, and smartphones (Google Cardboard). We can place objects within our scene using assorted primitive elements that come with A-Frame such as `<a-box>` or `<a-sphere>`. This approach helps with keeping the codebase readable. We could copy and paste this HTML to any other scene and it would behave in the same way. A-Frame (2017)

3.4.1 JavaScript and Aframe

We can use traditional JavaScript DOM APIs to manipulate A-Frame scenes to add logic, behavior, and functionality:

```
1 var box = document.querySelector('a-box');
2 box.getAttribute('position');
3 box.addEventListener('click', function () {
4   box.setAttribute('color', 'red');
5 });
```

A-Frame components are being based on the DOM, so most existing client-side libraries and frameworks such as React, Vue.js, d3.js, jQuery, or Angular work on top of A-Frame. the existing web ecosystem of tools were built on top of the notion of manipulating plain HTML and are thus compatible with A-Frame. A-Frame (2017)

3.4.2 Entity-Component System

A-Frame at its core is an entity-component-system framework. Entity-component-system (ECS) is a pattern popular in game development and is prominent in game engines like Unity. ECS favors composition over inheritance. Every single object

in the scene is an entity. an entity is an empty placeholder object that by itself does nothing. We plug in reusable components to attach appearance, behavior or functionality. We can also put different components together and configure them in order to define different types of objects. A-Frame (2017)

Object-oriented and hierarchical patterns have well-suited the 2D web, where we lay out elements and components that have fixed behavior on a web page. 3D and VR is different; there are infinite types of objects with endless complexity. A-Frame provides an easy way to build up different kinds of objects without having to create a special class for each one. A-Frame (2017)

In A-Frame, an entity is simply an HTML tag:

```
<a-entity></a-entity>
```

A-Frame components are reusable modules that can be plugged into any entity. They are allowed to do anything and have full access to JavaScript, Three.js, and Web APIs. the structure of a basic component may look like this:

```
AFRAME.registerComponent('foo', {
  schema: {
    bar: {type: 'number'},
    baz: {type: 'string'}
  },
  init: function () {
    // Do something when component is plugged in.
  },
  update: function () {
    // Do something when component's data is updated.
  }
});
```

Then once defined, we can plug this bundle of appearance, behavior, or functionality into an entity straight from an HTML attribute:

```
1 <a-entity foo="bar: 5; baz: qux"></a-entity>
```

3.4.3 Component Ecosystem

A-Frame ships with several components, but since A-Frame is fully extensible at its core, the community provided the ecosystem with lot's of components such as physics, particle systems, audio visualizations, and Leap Motion controls. This ecosystem is the driving force behind the popularization of A-Frame. Developer can build a component, publish it, and then someone else can take that component and use it straight from HTML without having to know JavaScript. A-Frame (2017)

These components are curated and collected into the A-Frame Registry. This is similar to the collection of components and modules on the Unity Asset Store or NPM, but free and open source. A-Frame maintainers and core developers make sure they work well and from there they are easily searchable and installable through multiple channels, one of which is through the A-Frame Inspector. A-Frame (2017)

3.4.4 A-Frame Inspector

The A-Frame Inspector is a visual tool for inspecting and editing A-Frame scenes. Similar to the browser's DOM Inspector, you can go to any A-Frame scene, local or on the Web, and hit `<ctrl>` + `<alt>` + `i` on your keyboard. This will open the visual Inspector where you can make changes and return to the scene with the reflected changes. You can visually move and place objects, change properties of the components, or pan the camera around to see a different view of the scene. It's similar to viewing the source code in an interactive way. the A-Frame Inspector is integrated

with the A-Frame Registry. From the Inspector, you can install components from the Registry and attach them to objects in the scene. A-Frame (2017)

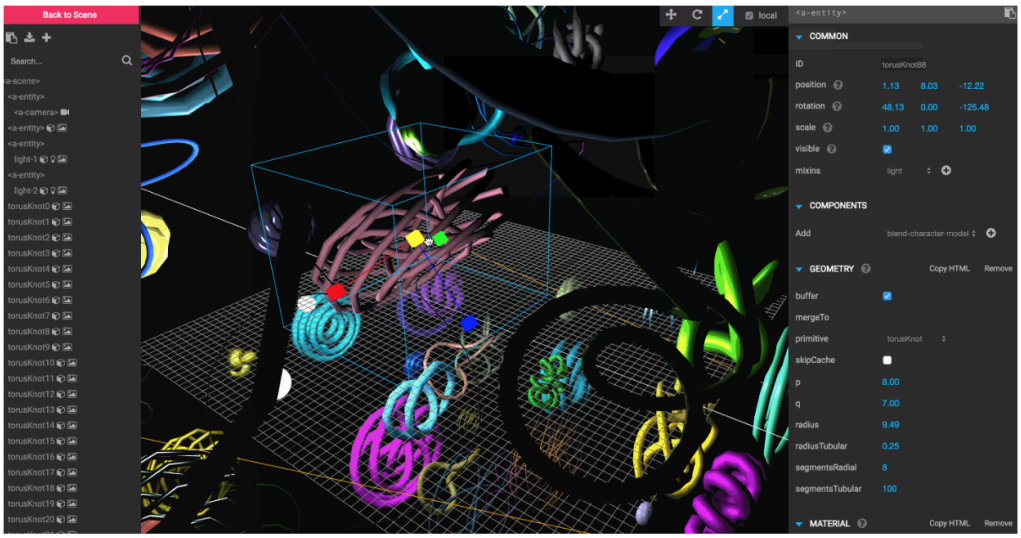

Figure 3 – 1 A-Frame Inspector window.

3.4.5 Device and Platform Support

Devices and platform support depends on how well the browsers support certain devices and APIs. A-Frame supports both flat (3D on a normal screen) and WebVR experiences, though its focus is heavily VR. Support for flat experiences primarily depends on a browser's WebGL support. We can see which browsers support WebGL at `http://caniuse.com/#feat=webgl`. Support for VR experiences, along with WebGL support, depends on a browser's support for the WebVR API. You can see which browsers currently support the WebVR API at `https://webvr.rocks`. A-Frame supports version 1.0 of the WebVR API. the exception to this is for mobile devices, which are supported through the WebVR Polyfill. A-Frame currently supports controllers with six degrees of freedom (6DoF) using experimental build of Chromium browser with experimental controller support (e.g., Vive controllers). A-Frame (2017)

3.5 Math.js

Math.js is an extensive math library for JavaScript and Node.js. It features a flexible expression parser with support for symbolic computation, comes with a large set of built-in functions and constants, and offers an integrated solution to work with different data types like numbers, big numbers, complex numbers, fractions, units, and matrices. Math.js (2017)

3.6 React

ReactJS is a javascript library for building user interfaces, originally created by engineers at Facebook to solve the challenges involved when developing complex user interfaces with datasets that change over time. It provides a way to write encapsulated components that manage their own state, then compose them to make complex user interfaces. It doesn't make assumptions about the rest of the technology stack, because it's just library. Since component logic is written in JavaScript instead of templates, we can easily pass rich data through our app and keep state out of the DOM. Gackenheimer (2015)

3.7 Flux

Flux is an application architecture created by Facebook for building client-side web applications. It complements the React in a way that displaces the standard Model-View-Controller (MVC) framework. It was designed in React in mind and aims to help developers to create more efficient, maintainable code when dissecting the application into multiple components, which according to Facebook engineers was hard if the traditional MVC pattern was followed when building the application with React.(Gackenheimer, 2015)

Flux is utilizing a unidirectional data flow. It's more of a pattern rather than a formal framework, and developers can start using Flux immediately without a lot of new code. (Facebook, Inc., 2015)

Flux applications have three major parts: the dispatcher, the stores, and the views (React components). These should not be confused with Model-View-Controller. Controllers do exist in a Flux application, but they are controller-views - views often found at the top of the hierarchy that retrieve data from the stores and pass this data down to their children. Additionally, action creators - dispatcher helper methods - are used to support a semantic API that describes all changes that are possible in the application. (Facebook, Inc., 2015)

Flux abandons MVC in favor of a uni-directional data flow. When a user interacts with a React view, the view propagates an action through a central dispatcher, to the various stores that hold the application's data and business logic, which updates all of the views that are affected. This works especially well with React's declarative programming style, which allows the store to send updates without specifying how to transition views between states. (Facebook, Inc., 2015)

3.7.1 Structure and Data Flow in Flux architecture

A uni-directional data flow is the main consequence of using the Flux pattern, and the diagram below should be the primary mental model for the Flux programmer. the dispatcher, stores and views are independent nodes with distinct inputs and outputs. the actions are simple objects containing a action type that identifies the property and the new data that is passed along with the action. (Facebook, Inc., 2015)

The views may cause a new action to be propagated through the system in response to user interactions. (Facebook, Inc., 2015)

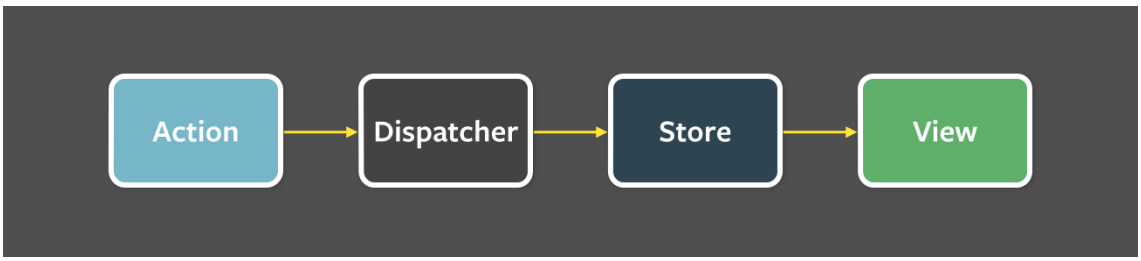

Figure 3 – 2 Data in a Flux application flows in a single direction. (Facebook, Inc., 2015)

All data flows through the dispatcher as a central hub. Actions are provided to the dispatcher in an action creator method, and most often originate from user interactions with the views. the dispatcher then invokes the callbacks that the stores have registered with it, dispatching actions to all stores. Within their registered callbacks, stores respond to whichever actions are relevant to the state they maintain. the stores then emit a change event to alert the controller-views that a change to the data layer has occurred. Controller-views listen for these events and retrieve data from the stores in an event handler. the controller-views call their own setState() method, causing a re-rendering of themselves and all of their descendants in the component tree. (Facebook, Inc., 2015)

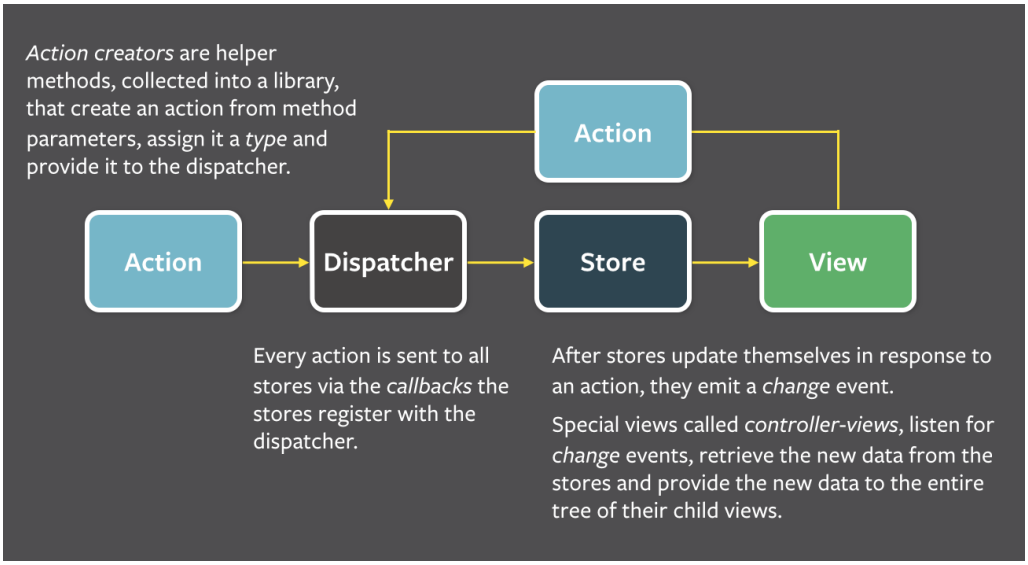

Figure 3 – 3 Detailed Flux application architecture diagram. (Facebook, Inc., 2015)

This structure allows us to reason easily about our application in a way that is reminiscent of functional reactive programming, or more specifically data-flow programming or flow-based programming, where data flows through the application in a single direction — there are no two-way bindings. Application state is maintained only in the stores, allowing the different parts of the application to remain highly decoupled. (Facebook, Inc., 2015)

We found that two-way data bindings led to cascading updates, where changing one object led to another object changing, which could also trigger more updates. As applications grew, these cascading updates made it very difficult to predict what would change as the result of one user interaction. When updates can only change data within a single round, the system as a whole becomes more predictable. (Facebook, Inc., 2015)

3.8 Redux

Redux is a predictable state container for JavaScript applications, inspired by several important qualities of Flux architecture. It helps developers write applications that behave consistently, run in different environments (client, server, and native), and are easy to test. On top of that, it provides a great developer experience, such as live code editing combined with a time traveling debugger.(Redux.org, 2017) We are using Redux together with React, but in general it can be used with any other view library.

The whole state of application is stored in an object tree inside a single store. the only way to change the state tree is to emit an action, an object describing what happened. To specify how the actions transform the state tree, we write pure reducers.(Redux.org, 2017)

Instead of mutating the state directly, we specify the mutations we want to happen

with plain objects called actions. Then we write a special function called a reducer to decide how every action transforms the entire application's state.(Redux.org, 2017)

Unlike Flux, Redux does not have the concept of a Dispatcher. This is because it relies on pure functions instead of event emitters, and pure functions are easy to compose and don't need an additional entity managing them.

The beauty and strength of this pattern is how well it scales to large and complex apps. It also enables very powerful developer tools, because it is possible to trace every mutation to the action that caused it. We can record user sessions and reproduce them just by replaying every action.(Redux.org, 2017)

3.8.1 Three Principles of Redux

1. Single source of truth

The state of your whole application is stored in an object tree within a single store. This makes it easy to create universal apps, as the state from your server can be serialized and hydrated into the client with no extra coding effort. A single state tree also makes it easier to debug or inspect an application; it also enables you to persist your app's state in development, for a faster development cycle. Some functionality which has been traditionally difficult to implement - Undo/Redo, for example - can suddenly become trivial to implement, if all of your state is stored in a single tree. (Redux.org, 2017)

2. State is read-only

The only way to change the state is to emit an action, an object describing what happened. This ensures that neither the views nor the network callbacks will ever write directly to the state. Instead, they express an intent to transform the state. Because all changes are centralized and happen one by one in a strict order, there

are no subtle race conditions to watch out for. As actions are just plain objects, they can be logged, serialized, stored, and later replayed for debugging or testing purposes. (Redux.org, 2017)

3. Changes are made with pure functions

To specify how the state tree is transformed by actions, you write pure reducers. Reducers are just pure functions that take the previous state and an action, and return the next state. Remember to return new state objects, instead of mutating the previous state. You can start with a single reducer, and as your app grows, split it off into smaller reducers that manage specific parts of the state tree. Because reducers are just functions, you can control the order in which they are called, pass additional data, or even make reusable reducers for common tasks such as pagination. (Redux.org, 2017)

3.9 Webpack

Webpack is a module bundler for modern JavaScript applications. It takes modules with dependencies and generates static assets representing those modules [3 – 4]. Its strength is that it's configurable and developers using it in their applications should know the four core concepts used when configuring Webpack:

- Entry - Webpack creates a graph of all of your application's dependencies and the starting point of this graph is called entry point. It tells Webpack where to start and follows the graph of dependencies to know what to bundle.
- Output - it tells Webpack where to bundle our application.
- Loaders - Webpack treats every file (.css, .html, .scss, .jpg, etc.) as a module. However, webpack only understands JavaScript, so loaders transform these files into modules as they are added to dependency graph.

- Plugins - loaders only execute transform on per-file basis, but plugins are mostly used to perform actions and custom functionality on combilations or chunks of bundled modules.

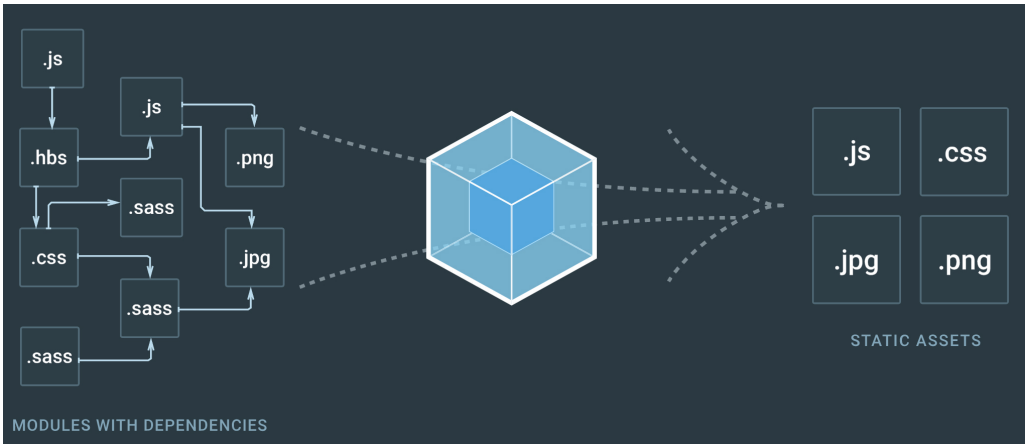

Figure 3 – 4 Bundling dependencies and static assets with Webpack

4 Conceptualisation and design

At the heart of user interface design is the "user" and the user is in the "driver's seat". Therefeore it is critical that users of any virtual environment should be able to use the interface intuitively, regardless of cultural diversities. the goal of user interface design is to make the user's interaction as simple and efficient as possible, in terms of accomplishing user goals. This is called user-centered design. Badrul (2011)

The most important goal for designing virtual environment in MathworldVR is to reach learners more effectively through seamless integration of content and its organization, together with the navigational and interactive controls that user use to work with the content.

4.1 Inputs and objections

MathworldVR is designed as a client-side, single page WebVR application. That means user doesn't have to install any additional software besides the supported experimental browser (this will change in future and user's would not need to install any additional browsers since Mozilla Firefox will introduce full WebVR support in version 55 and Google Chrome should introduce WebVR support in version 59). To be able to use MathworldVR properly, user needs to have on of HTC Vive or Oculus Touch headset installed and prepared. It can be used also on any device through any browser supporting WebVR API in version 1.0 and up. If accessed through the mobile device, user can look around the 3D environment but is not allowed to move from current position.

Code of the application will be fully open-sourced under the MIT license, hosted on `GitHub.com`, an online version control repository. That will allow more skilled de-

velopers from around the world contribute to the project and provide easier working conditions with various universities.

When user opens MathworldVR website, she should be informed about supported browsers and provided with the links to check her browser's WebVR support.

4.2 Functionality planning

MathworldVR will help students to explore various parametrized functions in 3D environment with use of VR hand controllers interactions. It will contain this functionality:

- Visualization of various parametrized functions from user input. User can explore the function, move around it, make it very big to see parts of it in detail or make it small for convenient movement and rotate it.
- Freedom of movement in room-scale, 3D environment, with six degrees of freedom. User can walk freely in a room or use teleport as a form of movement walk around the virtual components of MathworldVR, explore the world, parametrized function and components from different angles.
- Ability to change and update parametrized function's variables and constraints through the interactive settings panel. Function responds in real-time, giving the user immediate feedback to see what impact the change of variable has on parametrized function.
- Ability to leverage VR hand controllers' interactions to grab and scale the parametrized function. User's intuition expe
- Virtual calculator-like 3D interface with clickable buttons to serve as a panel for user input. User can choose from multiple variables, numbers from 0 to

9, operators and mathematical symbols to construct complex functions with It will include button for updating the parametrized function, because user input needs to be parsed with Math.js library first.

5 Implementation of virtual reality application

5.1 Project initialization

Before project can be initialized, Node.js has to be installed on PC used for development. To start a new JavaScript project, `npm init` command is used through command line, on Windows operating system it's usually PowerShell and on Unix systems, terminal. The multi-step wizard appears to guide developer through the project initialization process. It will generate the `package.json` file that includes these parts:

- Name - title of the project
- Version - project version, using semantic versioning
- Description - short project description
- Entry point - central file of our JavaScript code, usually index.js
- Test command - NPM script for testing
- Git repository - link to version control repository where application's code is hosted
- Keywords - used to better allocate the project within NPM packages search
- Author - project author
- License - project license

In the end, MathworldVR's `package.json` file looks like this:

```
1 {
2   "name": "mathworldvr",
3   "version": "0.0.1",
```

```
4   "description": "Math world in WebVR, powered by A-frame.",
5   "main": "src/index.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1"
8   },
9   "repository": {
10    "type": "git",
11    "url": "git+https://github.com/michaltakac/mathworld.git"
12  },
13  "keywords": [
14    "webvr",
15    "math",
16    "aframe",
17    "mathworld",
18    "vr",
19    "room-scale"
20  ],
21  "author": "Michal Tak   ",
22  "license": "MIT",
23  "bugs": {
24    "url": "https://github.com/michaltakac/mathworld/issues"
25  },
26  "homepage": "https://github.com/michaltakac/mathworld#readme"
27 }
```

5.2 Installing JavaScript packages

In modern JavaScript development, applications are build by using many packages together. To install and use a package, developer first needs to search for one in NPM registry on `https://www.npmjs.com/`. Our application will need multiple packages. They are installed through command line with command `npm install <package-name>` found at the right side of every package detail web page:

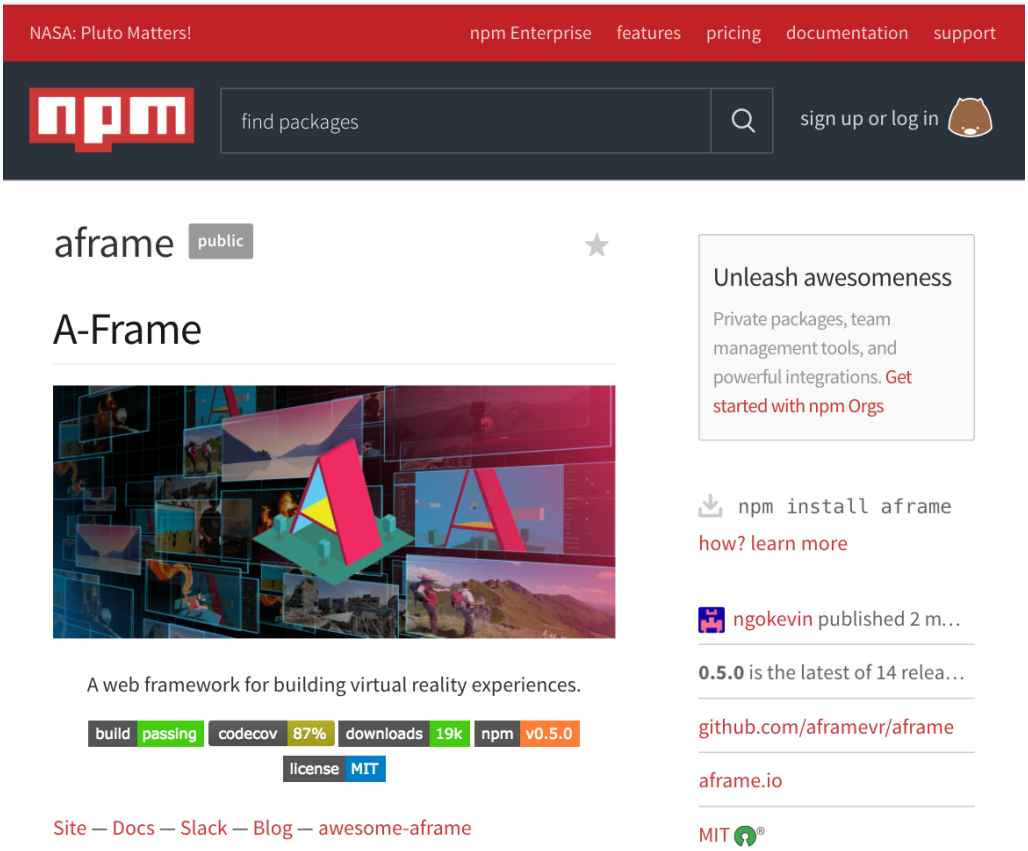

Figure 5 – 1 Detail page of JavaScript package in NPM registry with the script to install highlighted.

To save a package into `package.json` file as a dependency, `-save` option has to be added to `npm install` command. To save it as a development dependency (only used for local development, doesn't get included in final build), we need to add `-save-dev` option instead. Packages with their respective version tag will be added to `package.json` file into specific location:

```
"devDependencies": {
  "babel-cli": "^6.24.1",
  // ...
},
"dependencies": {
  "aframe": "^0.5.0",
  // ...
```

```
8 }
```

The list of used packages is included in System Manual.

5.3 NPM scripts

All of development, testing, bundling and deployment tasks are automated with `NPM scripts` defined in `package.json`.

```
1  "scripts": {
2    "start": "node server",
3    "test": "jest",
4    "test:watch": "npm test -- --watch",
5    "coverage": "npm test -- --coverage && opn coverage/lcov-
       report/index.html",
6    "lint": "eslint 'src/**/*.js' webpack.config.js server.js",
7    "clean": "del 'build/!(.git*|Procfile)**'",
8    "build:copy": "copyfiles -u 1 public/* public/**/* build",
9    "build:clean": "rimraf \"build/!(.git*|Procfile)**\"",
10   "prebuild": "npm run build:clean && npm run build:copy",
11   "build": "cross-env NODE_ENV=production webpack"
12 },
```

- "start" - Starts the development server.
- "test" - Initialize Jest test runner that searches for all files that have `*.test.js` or `*.spec.js` in their file name and triggers all tests.
- "test:watch" - Test runner will keep watching for file changes and will restart tests after every change.
- "coverage" - Generates detailed test coverage report.

- "lint" - Runs eslint for static code analysis testing according to preconfigured eslint rules in `.eslintrc`.
- "clean" - Deletes the `/build` folder.
- "build:copy" - Copies folders and files from `/public` into `/build`.
- "build:clean" - Similar to "clean" command.
- "prebuild" - Chain of NPM scripts that will be executed before actual "build" script.
- "build" - Production-ready build that can be deployed to server.

5.4 Project structure

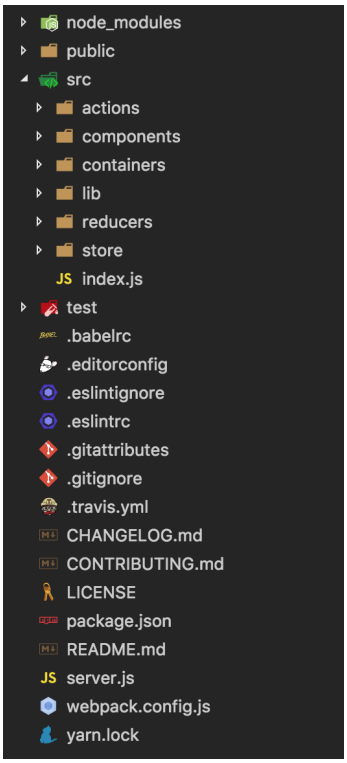

Figure 5 – 2 Project folder structure

5.5 Webpack configuration

Webpack is used as a tool to build JavaScript modules in MathworldVR application. It simplifies development workflow by quickly constructing a dependency graph of JavaScript application and bundling them in the right order. Our webpack configuration is defined in `webpack.config.js` and includes optimisations to the code like minification, obfuscation or splitting vendor/CSS/JavaScript code for production build. It also includes the configuration of Babel transpiler, which transpiles ES6/ES7 code to ES5, supported by all modern browsers.

5.6 Webpack development server

Development server configuration is defined in `server.js`. To start the server, `npm start` command is used from command line.

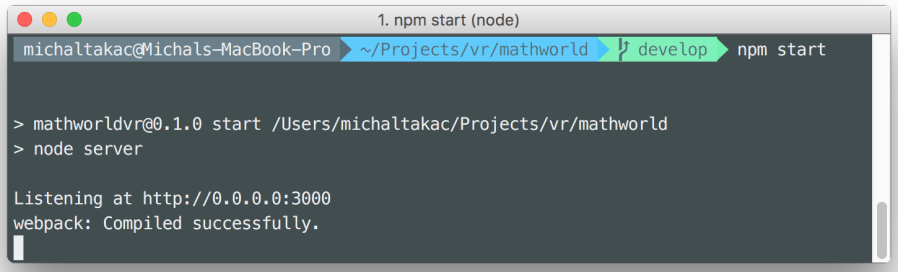

Figure 5 – 3 Started development server.

5.7 Setting up Redux for application state management

In Redux, all the application state is stored as a single object. We need to think of its shape before writing any code. In MathworldVR, there are 5 main "features" - `calculator`, interactable `function box` with grid, inside which resides

the `parametric function` handling 3D visualization and `settings` for this visualization, represented by interactive settings panel. Each of those has it's own state. These are combined together into one, global state, including additional `user` state that represents user position and `ui` state that represents traditional 2D user interface behaviour, like the visibility of information panel displayed after MathworldVR page is rendered in browser. For each feature and additional state, one Reducer function exists and by using reducer composition, all reducers are combined together into one, root reducer. Code for reducer composition is defined in `src/reducers/index.js`.

```
1  import { combineReducers } from 'redux'
2  import calculator from './calculator'
3  import functionBox from './functionBox'
4  import parametricFunction from './parametricFunction'
5  import settings from './settings'
6  import ui from './ui'
7
8  const rootReducer = combineReducers({
9    calculator,
10   functionBox,
11   parametricFunction,
12   settings,
13   ui
14 })
15
16 export default rootReducer
```

Listing 1 Reducers composition code.

`combineReducers()` method generates a function that calls all reducers with the slices of state selected according to their keys, and combining their results into a single object again.

Each reducer is defined in it's own file that also includes the `initialState` object,

which is passed into reducer function as a first argument, alongside action as a second argument. The reducer is a pure function that takes the previous state and an action, and returns the next state.

Actions are payloads of information that send data from application to Redux store. describe the fact that something happened, but don't specify how the application's state changes in response - this is the job of reducers. They are the only source of information for the store. To trigger an action, they must be dispatched using `store.dispatch()` method. Actions are plain JavaScript objects that must have a type property, indicating the type of action being performed. Types should typically be defined as string constants. Action creators are functions that create actions and returns them which makes them portable and easy to test. They are defined in `src/actions/index.js`.

The Store is the object that brings actions and reducers together. The store has the following responsibilities:

- Holds application state.
- Allows access to state via `getState()`.
- Allows state to be updated via `dispatch(action)`.
- Registers listeners via `subscribe(listener)`.
- Handles unregistering of listeners via the function returned by (listener).

It's important to note that Redux applications only have a single store. When we want to split the data handling logic, we'll use reducer composition instead of multiple stores.

Store configuration code for development is defined in `src/store/configureStore.dev.js`, which includes hot-reloading of reducers and action logger For production

build, different store configuration is defined in `src/store/configureStore.prod.js`. Usage of correct configuration is determined by `NODE_ENV` environment variable.

```
1 if (process.env.NODE_ENV === 'production') {
2   module.exports = require('./configureStore.prod')
3 } else {
4   module.exports = require('./configureStore.dev')
5 }
```

Listing 2 Loading store configuration dynamically.

5.8 Combining A-Frame and React

A-Frame framework is built on top of the DOM, thus web library such as React can be put cleanly on top of A-Frame. aframe-react (2017)

A-Frame is an entity-component-system (ECS) framework exposed through HTML. ECS is a pattern used in game development that favors composability over inheritance, which is more naturally suited to 3D scenes where objects are built of complex appearance, behavior, and functionality. aframe-react (2017)

In A-Frame, HTML attributes map to components which are composable modules that are plugged into `<a-entity>`s to attach appearance, behavior, and functionality. aframe-react (2017)

MathworldVR is using `aframe-react` module as a thin layer on top of A-Frame to bridge with React. It passes React props directly to A-Frame using refs and `.setAttribute()`, bypassing the DOM. This works since A-Frame's `.setAttribute()` is able to take non-string data such as objects, arrays, or elements and synchronously modify underlying 3D scene graph. aframe-react (2017)

With `aframe-react`, we get the the 3D and VR architecture of A-Frame, and the view and state ergonomics of React. React can be used to bind application and

state data to the values of A-Frame components. And we still have access to all the features and performance of A-Frame as well as A-Frame's community component ecosystem. aframe-react (2017)

5.9 A-Frame scene

3D scene initialization code is defined in `src/components/VRScene/index.js` by implementing the `<a-scene>` primitive from A-Frame API. It creates `THREE.Scene()` instance, sets the WebGL renderer by creating new `THREE.WebGLRenderer()` instance and combines them into a render loop. Abstractions like these makes the A-Frame framework a great WebVR development tool.

```
export default class VRScene extends React.Component {
  render() {
    return (
      <a-scene>
        {this.props.children}
      </a-scene>
    )
  }
}
```

Listing 3 *VRScene* component.

Additional A-Frame-specific libraries are loaded at the start of VRScene file:

```
// A-frame Components by community
import 'aframe'
import 'aframe-teleport-controls'
import 'super-hands'
import physics from 'aframe-physics-system'

// Libraries used by MathworldVR (Three.js, A-Frame, etc.)
import 'lib'
```

Here, we're loading `aframe-physics-system` which has specific initialization requirement - we need to call a function `physics.registerAll()` from within VRcene after it's loaded. That means we need to put it inside `componentWillMount()` function, which is React's standard lifecycle method:

```
1 componentWillMount() {
2   // Initialize aframe-physics-system
3   physics.registerAll()
4 }
```

To use `aframe-physics-system`, `physics="gravity: 0"` is added to `<a-scene>` primitive:

```
1 <a-scene physics="gravity: 0">
2   {this.props.children}
3 </a-scene>
```

5.10 Camera component

Camera component is implementing the `camera` from A-Frame API which is an abstraction over creation of a new `THREE.PerspectiveCamera` instance. Any additional properties are then passed into the main entity. `Camera` component is defined in `src/components/Camera/index.js`.

```
1 import React from 'react'
2 import { Entity } from 'aframe-react'
3
4 const Camera = (props) => {
5   return (
6     <Entity camera="" {...props} />
7   )
8 }
9
```

```
10 export default Camera
```

Listing 4 *Camera* component code.

5.11 Sky component

Sky component is implementing the `geometry` and `material` from A-Frame API. It's used to achieve the feeling of a sky around the user. `geometry.primitive` property is set to `'sphere'` with `geometry.radius` of 30 meters and `sky.jpg` texture prepared for a 360-degree image viewer is added to `material.src` property, pointing to an URL where image is stored. Any additional properties are then passed into the main entity. `Sky` component is defined in `src/components/Sky/index.js`.

```
1  import React from 'react'
2  import { Entity } from 'aframe-react'
3
4  const Sky = (props) => {
5    return (
6      <Entity
7        geometry={{ primitive: 'sphere', radius: 30, phiLength:
             360, phiStart: 0, thetaLength: 90 }}
8        material={{ shader: 'flat', src: 'url(sky.jpg)', side: '
             back', height: 2048, width: 2048 }}
9        {...props}
10     />
11   )
12 }
13
14 export default Sky
```

Listing 5 *Sky* component code.

5.12 Plane component

Plane component is implementing the `geometry` and `material` from A-Frame API. It's used as a floor under the user. `geometry.primitive` property is set to `'circle'` with `geometry.radius` of 12 meters and `floor.jpg` texture is added to `material.src` property, pointing to an URL where image is stored. Main entity is rotated 90-degrees around the X-axis. To accept light from Lights component, `material.shader` property is set to `'flat'`. Any additional properties are then passed into the main entity. `Plane` component is defined in `src/components/Plane/index.js`.

```
1  import React from 'react'
2  import { Entity } from 'aframe-react'
3
4  const Plane = (props) => {
5    return (
6      <Entity
7        geometry={{ primitive: 'circle', radius: 12 }}
8        material={{ src: 'url(floor.jpg)', shader: 'flat',
             roughness: 0 }}
9        rotation="-90 0 0"
10       static-body
11       {...props}
12     />
13   )
14 }
15
16 export default Plane
```

Listing 6 *Plane* component code.

5.13 Lights component

`Lights` component is implementing the `light` from A-Frame API. It's used for lighting up the scene and main points of interest, such as FunctionBox, Calculator and SettingsPanel components. MathworldVR uses two types of light - `'point'` and `'hemisphere'`, grouped into single component. Light intensity is set with `intensity` property. `Lights` component is defined in `src/components/Lights/index.js`.

```
1  import React from 'react'
2  import { Entity } from 'aframe-react'
3
4  const Lights = (props) => {
5    return (
6      <Entity {...props}>
7        <Entity light={{ type: 'point', color: '#fff', intensity:
             0.6 }} position={{ x: 3, y: 10, z: 1 }} />
8        <Entity light={{ type: 'point', color: '#fff', intensity:
             0.2 }} position={{ x: -3, y: -10, z: 1 }} />
9        <Entity light={{ type: 'hemisphere,', groundColor: '#888',
              intensity: 0.8 }} />
10     </Entity>
11   )
12 }
13
14 export default Lights
```

Listing 7 *Lights* component code.

5.14 Text component

`Text` component is implementing the `text` from A-Frame API. It's used for displaying 2D text inside 3D environment. `Text` component is defined in `src/components`

/Text/index.js.

```
1  import React from 'react'
2  import { Entity } from 'aframe-react'
3
4  const Text = ({ align, color, letterSpacing, lineHeight, opacity
       , value, width, zOffset, ...props }) => {
5    return (
6      <Entity
7        text={{ value, width, align, letterSpacing, lineHeight,
             color, opacity, zOffset }}
8        {...props}
9      >
10       {props.children}
11     </Entity>
12   )
13 }
14
15 export default Text
```

Listing 8 *Lights* component code.

5.15 Calculator component

5.15.1 Action types

Calculator action types are defined in `src/actions/index.js` as string constants and exported individually to ensure they can be imported in other files and also tested.

```
1 export const CALCULATOR_WRITE_TEXT = 'CALCULATOR_WRITE_TEXT'
2 export const CALCULATOR_BACKSPACE = 'CALCULATOR_BACKSPACE'
3 export const CALCULATOR_CLEAR_TEXT = 'CALCULATOR_CLEAR_TEXT'
```

Listing 9 `calculator` action types.

5.15.2 Action creators

Calculator's action creators are pure functions that return payloads of information about type of `CALCULATOR` action being dispatched and, if needed, also the data we want to send to Redux store. They are defined in `src/actions/index.js`

```
export const calculatorWriteText = (text) => ({
    type: CALCULATOR_WRITE_TEXT,
    text
})
```

Listing 10 Action for writing text to calculator display.

```
export const calculatorBackspace = () => ({
    type: CALCULATOR_BACKSPACE
})
```

Listing 11 Action to remove character from calculator display.

```
export const calculatorClearText = () => ({
    type: CALCULATOR_CLEAR_TEXT
})
```

Listing 12 Action to completely clear the calculator display.

5.15.3 Initial state

Calculator component's initial state is defined in `src/reducers/calculator/index.js` as a `initialState` object.

```
const initialState = {
  displayText: 'x^2 + y^2',
}
```

Listing 13 Initial state of a *calculator*.

5.15.4 Reducer function

Reducer function for `Calculator` component is defined in `src/reducers/calculator/index.js`. From this file it's exported as a default function that takes state (with initialState as a default value) and action as its arguments and returns new state according to `CALCULATOR` actions. Possible `ActionTypes` are conditionally checked with JavaScript's `switch` statement for a better code readability.

```
export default (state = initialState, action) => {
  switch (action.type) {
    // ...
    case ActionTypes.CALCULATOR_WRITE_TEXT:
      return { ...state, displayText: `${state.displayText}${
          action.text}` }
   // ...
    default: return state
  }
}
```

Listing 14 Addition of text to `state.calculator.displayText`.

```
export default (state = initialState, action) => {
  switch (action.type) {
    // ...
    case ActionTypes.CALCULATOR_BACKSPACE:
      return { ...state, displayText: state.displayText.slice(0,
          -1) }
   // ...
    default: return state
  }
}
```

Listing 15 Remove one character from `state.calculator.displayText`.

```
export default (state = initialState, action) => {
  switch (action.type) {
    // ...
```

```
4       case ActionTypes.CALCULATOR_CLEAR_TEXT:
5         return { ...state, displayText: '' }
6     // ...
7       default: return state
8     }
9   }
```

Listing 16 Clear the `state.calculator.displayText`.

5.15.5 Higher-order component

State and actions that Calculator component needs to function properly are mapped to properties in Calculator container, also called "higher-order component", because it's "aware" of application's state that is mapped into it. Code for Calculator higher-order component creation is defined in `src/containers/Calculator/index.js`.

```
1 const mapStateToProps = (state) => ({
2   displayText: state.calculator.displayText,
3 })
```

Listing 17 Function to map `calculator` state to component properties.

```
1 const mapDispatchToProps = (dispatch) => ({
2   writeText: (text) => dispatch(calculatorWriteText(text)),
3   backspace: () => dispatch(calculatorBackspace()),
4   clearText: () => dispatch(calculatorClearText()),
5   updateEquation: (equation) => dispatch(
      parametricFunctionSetEquation(equation)),
6 })
```

Listing 18 Function to map dispatchable `calculator` action creators to component properties.

```
1 import { connect } from 'react-redux'
2 import { Calculator } from 'components'
3 // ...
```

```
4 export default connect(mapStateToProps, mapDispatchToProps)(
    Calculator)
```

Listing 19 Creation of `Calculator` higher-order component.

5.15.6 Presentational component

Presentational Calculator component is implementing the `geometry` and `material` from A-Frame API. It's shape is determined by `geometry.primitive` which is set to `'box'` with width of `0.88` meters, height of `0.65` meters and depth of `0.01` meters. By default it's not moveable to ensure its position central to the MathworldVR experience. Properties passed into the component as function attributes includes state and dispatchable action that was mapped in it's higher-order component. `Calculator` code is defined in `src/components/Calculator/index.js`. Code preview displayed here doesn't include the CalcButton components, but they're utilizing the `writeText`, `backspace` and `clearText` actions. They are explained in next section.

```
1  const Calculator = ({ displayText, writeText, backspace,
     clearText, updateEquation }) => {
2    return (
3      <Entity
4        geometry="primitive: box; width: 0.88; height: 0.65;
           depth: 0.01;"
5        material="shader: flat; side: double; color: #8d8547;"
6      >
7        { /* Calculator display */ }
8        <Text value={displayText} />
9        { /* --- BUTTONS --- */ }
10       { /* ... */ }
11     </Entity>
12   )
13 }
```

Listing 20 Presentational *Calculator* component code.

5.16 CalcButton component

`CalcButton` is interactable component that is reacting to VR hand controllers interactions. When hand controller intersects with CalcButton in 3D environment, it fires an `hover-start` event, which starts the chain reaction of another two events by calling a `startInteraction()` method that fires Redux action passed to CalcButton as a property attribute and at the same time changes the depth and opacity of CalcButton. This is providing a visual cue - simulation of an actual click of a button on real calculator. After hand controller leaves the area of CalcButton, `hover-end` event is fired, initializing `endInteraction()` method which sets the depth and opacity back to default values. `CalcButton` component is defined in `src/components/CalcButton/index.js`.

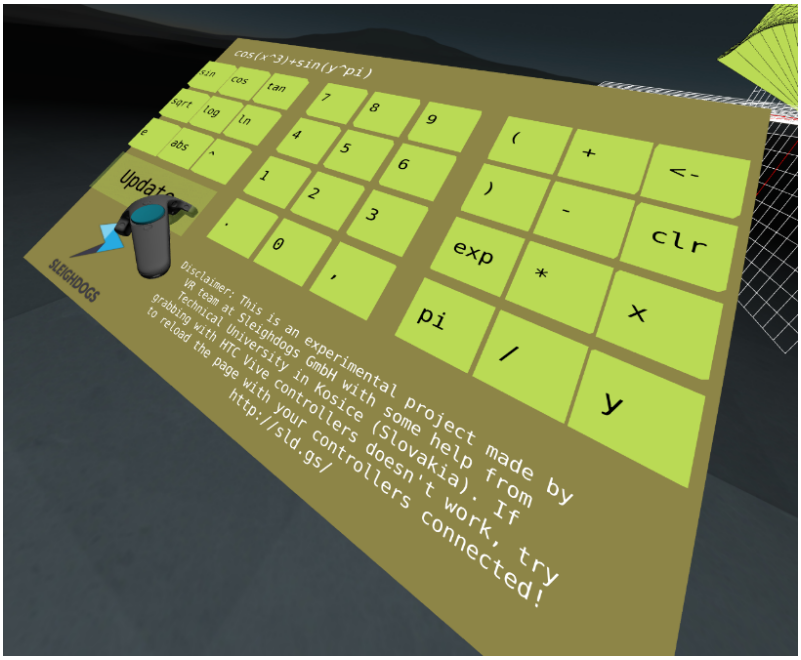

Figure 5 – 4 *Calculator* with multiple *CalcButton* components.

5.17 ParametrizedFunction component

First, the A-Frame API had to be extended with `parametricfunction` component:

```
AFRAME.registerComponent('parametricfunction', {
  schema: {/* ... */},
  init: function() {/* ... */},
  update: function() {/* ... */},
  remove: function() {/* ... */},
})
```

Listing 21 Registering new A-Frame component `'parametricfunction'`.

A-Frame components should have schema defined to ensure what data is passed into them. `parametricfunction`'s schema includes `equation`, `segments`, `xMin`, `xMax`, `yMin`, `yMax`, `zMin`, `zMax` and `functionColor`, with their respective default values.

```
schema: {
  equation: { type: 'string', default: '' },
  segments: { type: 'number', default: 20 },
  xMin: { type: 'number', default: -5 },
  xMax: { type: 'number', default: 5 },
  yMin: { type: 'number', default: -5 },
  yMax: { type: 'number', default: 5 },
  zMin: { type: 'number', default: -5 },
  zMax: { type: 'number', default: 5 },
  functionColor: { type: 'string', default: '#bada55' }
}
```

Listing 22 `parametricfunction` A-Frame component schema with default values.

Main functions of `parametricfunction` A-Frame component are:

- Parsing the equation passed into it with use of `Math.js` module and visualizing it in 3D environment, inside FunctionBox component that provides the grid.
- Updating the 3D visualization after settings in `SettingsPanel` changes.

```
var equation = 'f(x,y) = ' + this.data.equation;
var parser = math.parser();

try {
  parser.eval(equation);
} catch (error) {
  return;
}
```

Listing 23 Parsing of the equation.

To get the parsed value as a function, `parser.get('f')` method is called:

```
const f1 = parser.get('f');
parser.clear();
```

The code of `parametricfunction` A-Frame component implementation is defined in `src/lib/components/parametricfunction.js`

5.17.1 Action types

ParametricFunction action types are defined in `src/actions/index.js` as string constants and exported individually to ensure they can be imported in other files and also tested.

```
export const PARAMETRIC_FUNCTION_SET_EQUATION = '
    PARAMETRIC_FUNCTION_SET_EQUATION'
```

Listing 24 `parametricFunction` action types.

5.17.2 Action creators

ParametricFunction's action creators are pure functions that return payloads of information about type of `PARAMETRIC_FUNCTION` action being dispatched and, if

needed, also the data we want to send to Redux store. They are defined in `src/actions/index.js`

```
export const parametricFunctionSetEquation = (equation) => ({
  type: PARAMETRIC_FUNCTION_SET_EQUATION,
  equation,
})
```

Listing 25 Action for setting the function for 3D visualization.

5.17.3 Initial state

ParametricFunction component's initial state is defined inside the `src/reducers/parametrizedFunction/index.js` file as a `initialState` object.

```
const initialState = {
  equation: 'x^2 + y^2',
}
```

Listing 26 Initial state of a *calculator*.

5.17.4 Reducer function

Reducer function for `ParametricFunction` component is defined in `src/reducers/parametricFunction/index.js`. From this file it's exported as a default function that takes state (with initialState as a default value) and action as its arguments and returns new state according to `PARAMETRIC_FUNCTION` actions. Possible `ActionTypes` are conditionally checked with JavaScript's `switch` statement for a better code readability.

```
export default (state = initialState, action) => {
  switch (action.type) {
    // ...
    case ActionTypes.PARAMETRIC_FUNCTION_SET_EQUATION:
```

```
5         return { ...state, equation: action.equation }
6       // ...
7         default: return state
8     }
9   }
```

Listing 27 Update the `state.parametricFunction.equation` value.

5.17.5 Higher-order component

State and actions that ParametricFunction component needs to function properly are mapped to properties in ParametricFunction container - higher-order component. Code for ParametricFunction higher-order component creation is defined in `src/containers/ParametricFunction/index.js`.

```
1  const mapStateToProps = (state) => ({
2    equation: state.parametricFunction.equation,
3    segments: state.settings.segments,
4    xMin: state.settings.xMin,
5    xMax: state.settings.xMax,
6    yMin: state.settings.yMin,
7    yMax: state.settings.yMax,
8    zMin: state.settings.zMin,
9    zMax: state.settings.zMax,
10   functionColor: state.settings.functionColor,
11 })
```

Listing 28 Function to map `parametricFunction` and `settings` state to component properties.

```
1 import { connect } from 'react-redux'
2 import { ParametricFunction } from 'components'
3 // ...
4 export default connect(mapStateToProps)(ParametricFunction)
```

Listing 29 Creation of `ParametricFunction` higher-order component.

5.17.6 Presentational component

Presentational ParametricFunction component is implementing the `parametric-function` from A-Frame API we extended. Properties passed into the component as function attributes includes state and dispatchable action that was mapped in it's higher-order component. `ParametricFunction` code is defined in `src/components/ParametricFunction/index.js`.

```
1  const ParametricFuntion = ({ equation, segments, xMin, xMax,
       yMin, yMax, zMin, zMax, functionColor }) => {
2    return (
3      <Entity
4        id="function-mesh"
5        parametricfunction={{
6          equation,
7          segments,
8          xMin,
9          xMax,
10         yMin,
11         yMax,
12         zMin,
13         zMax,
14         functionColor,
15       }}
16       grid="size: 2; step: 20"
17     />
18   )
19 }
```

Listing 30 Presentational `ParametricFunction` component code.

5.18 FunctionBox component

5.18.1 Action types

FunctionBox action types are defined in `src/actions/index.js` as string constants and exported individually to ensure they can be imported in other files and also tested.

```
1 export const FUNCTION_BOX_SET_POSITION = '
    FUNCTION_BOX_SET_POSITION'
```

Listing 31 `functionBox` action types.

5.18.2 Action creators

FunctionBox's action creators are pure functions that return payloads of information about type of `FUNCTION_BOX` action being dispatched and, if needed, also the data we want to send to Redux store. They are defined in `src/actions/index.js`

```
1 export const functionBoxSetPosition = (position) => ({
2   type: FUNCTION_BOX_SET_POSITION,
3   position,
4 })
```

Listing 32 Action for setting function box position.

5.18.3 Initial state

`FunctionBox` component's initial state is defined in `src/reducers/functionBox/index.js` as a `initialState` object.

```
1 const initialState = {
2   position: { x: 0.65, y: 1.45, z: -1.03 },
```

```
3 }
```

Listing 33 Initial state of a *functionBox*.

5.18.4 Reducer function

Reducer function for `FunctionBox` component is defined in `src/reducers/functionBox/index.js`. From this file it's exported as a default function that takes state (with `initialState` as a default value) and `action` as its arguments and returns new state according to `FUNCTION_BOX` actions. Possible `ActionTypes` are conditionally checked with JavaScript's `switch` statement for a better code readability.

```
1 export default (state = initialState, action) => {
2   switch (action.type) {
3     case ActionTypes.FUNCTION_BOX_SET_POSITION:
4       return { ...state, position: action.position }
5     default: return state
6   }
7 }
```

Listing 34 Updating the `state.functionBox.position` value.

5.18.5 Higher-order component

State and actions that `FunctionBox` component needs to function properly are mapped to properties in Calculator container - higher-order component. Code for FunctionBox higher-order component creation is defined in `src/containers/FunctionBox/index.js`.

```
1 const mapStateToProps = (state) => ({
2   position: state.functionBox.position,
3 })
```

Listing 35 Function to map `functionBox` state to component properties.

```
import { connect } from 'react-redux'
import { FunctionBox } from 'components'
// ...
export default connect(mapStateToProps, null)(FunctionBox)
```

Listing 36 Creation of `FunctionBox` higher-order component.

5.18.6 Presentational component

Presentational FunctionBox component is implementing the `geometry` and `material` from A-Frame API. It's shape is determined by `geometry.primitive` which is set to `'box'` with width of `4` meters, height of `4` meters and depth of `4` meters. By default it's scaled down by 80%, but since this component is interactive, it can be stretched with hand controller interactions. It can also be moved around the 3D environment. `Position` property is passed into the component from higher-order component. `FunctionBox` code is defined in `src/components/FunctionBox/index.js`.

```
const FunctionBox = ({ position, ...props }) => (
  <Entity
    id="function-box"
    className="interactive"
    geometry="primitive: box; width: 4; height: 4; depth: 4;"
    material="transparent: true; opacity: 0; shader: standard"
    scale="0.2 0.2 0.2"
    position={position}
    grabbable
    stretchable
    dynamic-body
    stop-flying
  >
    { props.children }
  </Entity>
```

```
16 )
```

Listing 37 Presentational *FunctionBox* component code.

5.19 SettingsPanel component

First, the A-Frame API has to be extended with `datgui` component. Inside its `init` method, the `dat.guiVR` instance is created and injected into the component element property:

```
1 const gui = dat.GUIVR.create( this.data.name );
2 this.el.setObject3D('gui', gui );
3 this.el.gui = gui;
```

To allow VR hand controllers interactions, `bindInput` method is used for creation of event listeners for `triggerdown`, `triggerup`, `gripdown` and `gripup` events:

```
1  function bindInput(el, input) {
2    el.addEventListener('triggerdown', function() {
3      input.pressed(true);
4    });
5    el.addEventListener('triggerup', function() {
6      input.pressed(false);
7    });
8    el.addEventListener('gripdown', function() {
9      input.gripped(true);
10   });
11   el.addEventListener('gripup', function() {
12     input.gripped(false);
13   });
14 }
```

5.19.1 Action types

SettingsPanel action types are defined in `src/actions/index.js` as string constants and exported individually to ensure they can be imported in other files and also tested.

```
1 export const SETTINGS_SET_X_MIN = 'SETTINGS_SET_X_MIN'
2 export const SETTINGS_SET_Y_MIN = 'SETTINGS_SET_Y_MIN'
3 export const SETTINGS_SET_Z_MIN = 'SETTINGS_SET_Z_MIN'
4 export const SETTINGS_SET_X_MAX = 'SETTINGS_SET_X_MAX'
5 export const SETTINGS_SET_Y_MAX = 'SETTINGS_SET_Y_MAX'
6 export const SETTINGS_SET_Z_MAX = 'SETTINGS_SET_Z_MAX'
7 export const SETTINGS_SET_SEGMENTS = 'SETTINGS_SET_SEGMENTS'
8 export const SETTINGS_SET_FUNCTION_COLOR = '
    SETTINGS_SET_FUNCTION_COLOR'
```

Listing 38 `settings` action types.

5.19.2 Action creators

SettingsPanel's action creators are pure functions that return payloads of information about type of `SETTINGS` action being dispatched and, if needed, also the data we want to send to Redux store. They are defined in `src/actions/index.js`

```
1 export const settingsSetSegments = (segments) => ({
2   type: SETTINGS_SET_SEGMENTS,
3   segments,
4 })
```

Listing 39 Action for setting fnction segments.

5.19.3 Initial state

SettingsPanel component's initial state is defined in `src/reducers/settings/index.js` as a `initialState` object.

```
1  const initialState = {
2    xMin: -1,
3    yMin: -1,
4    zMin: -4,
5    xMax: 1,
6    yMax: 1,
7    zMax: 4,
8    segments: 30,
9    functionColor: '#bada55',
10 }
```

Listing 40 Initial state of a *calculator*.

5.19.4 Reducer function

Reducer function for `SettingsPanel` component is defined in `src/reducers/settings/index.js`. From this file it's exported as a default function that takes state (with initialState as a default value) and action as its arguments and returns new state according to `SETTINGS` actions. Possible `ActionTypes` are conditionally checked with JavaScript's `switch` statement for a better code readability.

```
1  export default (state = initialState, action) => {
2    switch (action.type) {
3      // ...
4      case ActionTypes.SETTINGS_SET_SEGMENTS:
5        return { ...state, segments: action.segments }
6     // ...
7      default: return state
8    }
```

```
9 }
```

Listing 41 Update of `state.settings.segments` value.

5.19.5 Higher-order component

State and actions that SettingsPanel component needs to function properly are mapped to properties in SettingsPanel container - higher-order component. Code for SettingsPanel higher-order component creation is defined in `src/containers/SettingsPanel/index.js`.

```
1  const mapDispatchToProps = (dispatch) => {
2    return {
3      setXMin: (xMin) => dispatch(settingsSetXMin(xMin)),
4      setYMin: (yMin) => dispatch(settingsSetYMin(yMin)),
5      setZMin: (zMin) => dispatch(settingsSetZMin(zMin)),
6      setXMax: (xMax) => dispatch(settingsSetXMax(xMax)),
7      setYMax: (yMax) => dispatch(settingsSetYMax(yMax)),
8      setZMax: (zMax) => dispatch(settingsSetZMax(zMax)),
9      setSegments: (segments) => dispatch(settingsSetSegments(
           segments)),
10     setFunctionColor: (color) => dispatch(
           settingsSetFunctionColor(color)),
11   }
12 }
```

Listing 42 Function to map dispatchable `settings` action creators to component properties.

```
1 import { connect } from 'react-redux'
2 import { SettingsPanel } from 'components'
3 // ...
4 export default connect(null, mapDispatchToProps)(SettingsPanel)
```

Listing 43 Creation of `SettingsPanel` higher-order component.

5.19.6 Presentational component

Presentational SettingsPanel component is implementing the `datgui` from A-Frame API that we extended. Individual settings are handled by `SettingsController` component, which triggers the action when user changes the value by pointing to the settings with VR hand controller and pressing the trigger button. `SettingsPanel` code is defined in `src/components/SettingsPanel/index.js`.

```
1  const SettingsPanel = ({
2    name,
3    controllerLeft,
4    controllerRight,
5    setSegments,
6    ...props
7  }) => {
8    return (
9      <Entity datgui={{ name, controllerLeft, controllerRight }}
           {...props}>
10       <SettingsController type="slider" name="segments" step={1}
             min={1} max={50} initialState={30} actionToTrigger={
           setSegments} />
11       { /* ...other settings... */ }
12     </Entity>
13   )
14 }
```

Listing 44 Presentational *Calculator* component code.

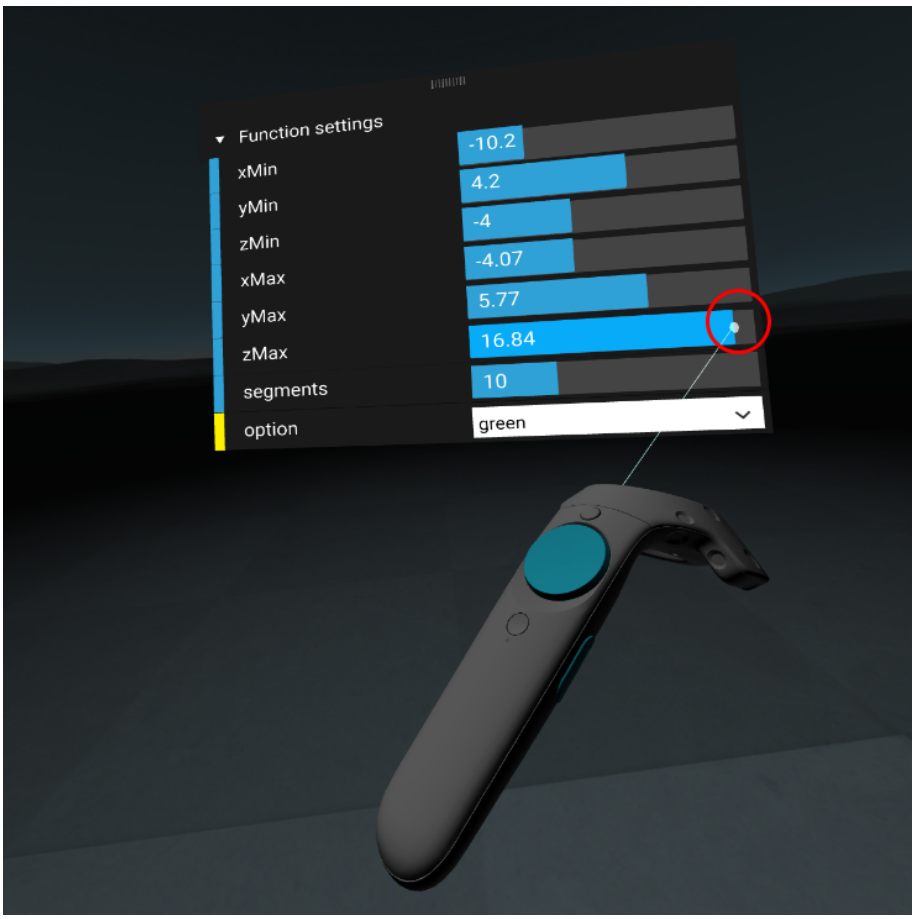

Figure 5 – 5 `SettingsPanel` component

5.20 Hand controller components

VR hand controllers implement the `super-hands` A-Frame API that was extended by importing the `aframe-super-hands-component` in MathworldVR's `VRScene` component. `super-hands` A-Frame component adds natural, intuitive hand controller interactions, interprets input from tracked controllers and collision detection into interaction gestures and communicates those gestures to target entities for them to respond.

The currently implemented gestures are:

- Hover - Holding a controller in the collision space of an entity.

- Grab - Pressing a button while hovering an entity, potentially also moving it.
- Stretch - Grabbing an entity with two hands and resizing.
- Drag-drop - Dragging an entity onto another entity.

`super-hand` includes components for typical reactions to the implemented gestures: `hoverable`, `grabbable`, `stretchable`, and `drag-droppable`. Code for hand controllers implementation used in MathworldVR is defined in `src/components/LeftController/index.js` and `src/components/LeftController/index.js` respectively.

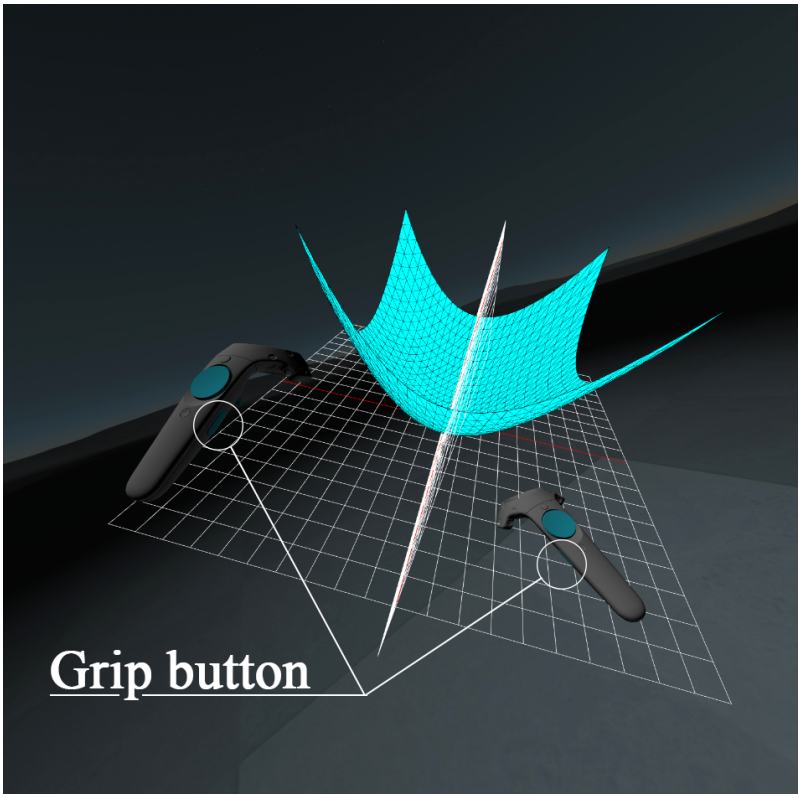

Figure 5 – 6 Grabbing functionality

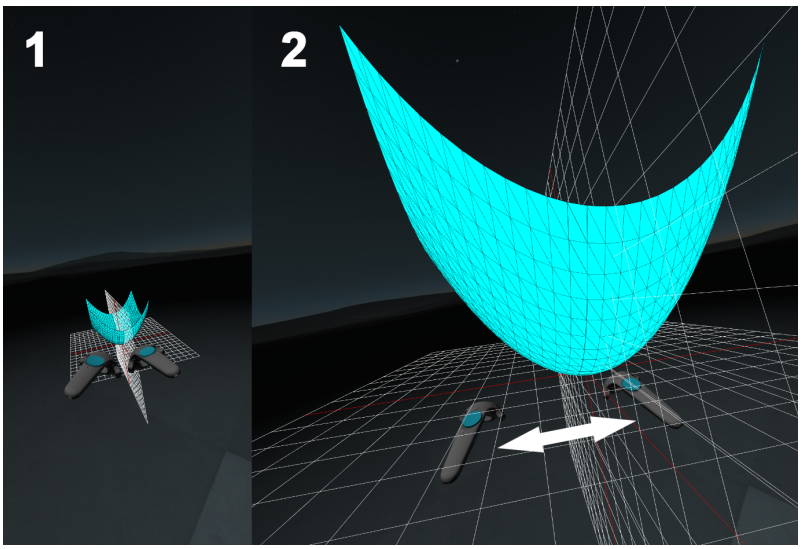

Figure 5 – 7 Scaling functionality

5.21 AttentionBox component

`AttentionBox` component is used for displaying semi-transparent rectangle with MathworldVR project information in the middle of the browser screen. `AttentionBox` component is defined in `src/components/AttentionBox/index.js`.

5.22 Adding components into A-Frame scene

To add MathworldVR components into the 3D environment, they need to be included as a children components of `VRScene` component:

```
1 <VRScene>
2   <AttentionBox />
3   <LeftController />
4   <RightController />
5
6   <FunctionBox>
7     <ParametricFunction />
8   </FunctionBox>
```

```
9
10    <Calculator />
11
12    <SettingsPanel
13      name="Function settings"
14      position={{ x: -0.37, y: 1.93, z: -0.34 }}
15      rotation={{ x: 10, y: 30, z: 0 }}
16      scale={{ x: 0.5, y: 0.5, z: 0.5 }}
17    />
18
19    <Sky />
20    <Lights />
21    <Plane />
22  </VRScene>
```

5.23 Building and deploying the application to web hosting

Application's JavaScript code is bundled together by using NPM script `npm run build`, which produces production-ready bundle. Build script triggers pre-build task, which gets executed before the build and consists of `npm run build:clean` (for removing all files from `/build` folder) and `npm run build:copy` (for copying static files like `index.html`, assets, images, CSS stylesheets into cleaned `/build` folder) scripts, called synchronously.

To deploy the application, `/build` folder contents are copied to the server, hosted on `https://www.fortrabbit.com/`. Hosting is sponsored by *Sleighdogs, GmbH* company, which author is working for during writing of this thesis. Server us hosted under `http://vr.sld.gs/mathworldvr/` domain.

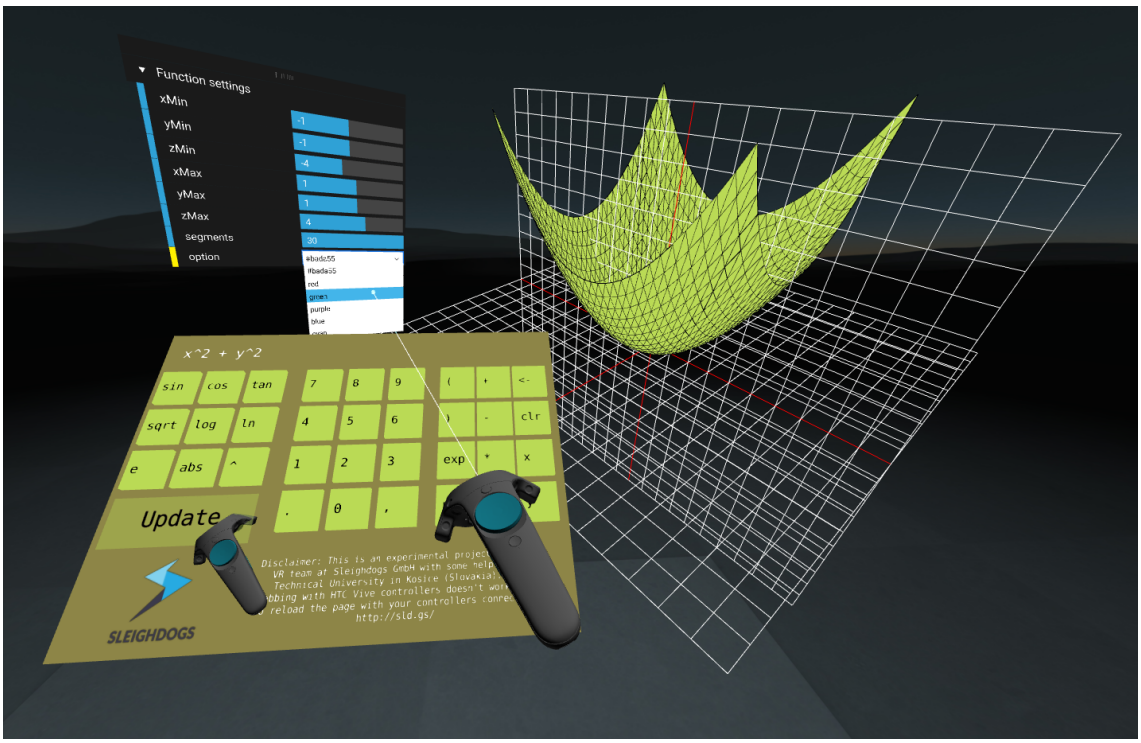

Figure 5 – 8 Finalized MathworldVR application, deployed on live server.

6 Conclusion

Goal of the thesis was to explore possibilities of virtual reality on the web, design and implement VR application called MathworldVR that could help students explore, experiment and learn about various parametrized functions, allowing them to build an intuition of how such functions behave according to changes of their variables.

We managed to not only successfuly implement the WebVR application, but also push forward the whole ecosystem of web-based virtual reality provided by A-Frame framework. We decided to open-source MathworldVR so more ideas can "flow" freely into the project through contributors from all around the world.

Many components were created from scratch for the project and we also inspired other A-Frame developers by sharing the progress along the way on social media websites Facebook, Twitter and Instagram.

MathworldVR cought an eye of prof. Steven Abbott (`https://www.stevenabbott.co.uk/`), former Visiting Professor at the School of Mechanical Engineering at University Leeds, who helped us with the Oculus Touch support.

We also joined Virtuleap's online, worldwide WebVR hackathon (`http://www.virtuleap.com/`), where we managed to move into finals from 33 participanting projects and ended in 10th place overall. This triggered the public interest, helping get MathworldVR mentioned in A-Frame weekly blog and VentureBeat, *"the leading source for news, events, groundbreaking research and perspective on technology innovation"* (`https://venturebeat.com/2017/01/10/virtuleap-hackathon-generates-a-bunch-of-webvr-projects/`).

Current functionality of MathworldVR is very limited, but interest and recognition this project received shown that it's definitely worth to take it to the next level in near future. This will be also easier since we open-sourced it.

Next step will be building the infrastructure, adding the multi-user support and creating a back-end with database. We're looking into Elixir programming language and Phoenix framework, because it looks very promising. It leverages the ability of Erlang's virtual machine to handle millions of connections alongside Elixir's beautiful syntax and productive tooling for building fault-tolerant systems. This will make creating different (public or private) "rooms" and persisting the position, rotation and application state, possible.

Another planned features include: more examples in form of "mathematical rooms" of different kind, better calculator component, intuitive menu added to VR hand controller incorporating multiple options and layers, 2D graphs support, double and surface integrals, vectors and many more. We plan to create a public roadmap of planned features to serve as a guide for developers, universities and others who are interested about the progress.

References

BADRUL, K. 2011. *User Interface Design for Virtual Environments: Challenges and Advances.* USA: IGI Global, 2011. 375 s. ISBN 9781613505175

FLANAGAN, D. 2006. *JavaScript: The Definitive Guide (5th edition).* USA: O'Reilly, 2006. 1032 s. ISBN 978-0-596-10199-2

NARAYAN, P. 2015. *Learning ECMAScript 6.* USA: Packt Publishing, 2015. 202 s. ISBN 9781785884443

LOEFFLER, C. 1995. *Distributed Virtual Reality: Applications for Education, Entertainment and Industry.* `http://www.wiumlie.no/1993/telektronikk-4-93/Loeffler_C_E.html`

KAUFMANN, H. 2011. *Virtual Environments for Mathematics and Geometry Education.* In: Themes In Science and Technology Education. Vienna : Klidarithmos Computer Books, 2011, Special Issue, pp. K 131–152

NPM. [online]. Retrieved April 2, 2017 from `https://docs.npmjs.com/all`

A-Frame. [online]. Retrieved April 2, 2017 from `https://aframe.io/docs/0.5.0/introduction/`

Math.js. [online]. Retrieved April 2, 2017 from `http://mathjs.org/`

GACKENHEIMER, C. 2015. *Introduction to React.* USA: Apress, 2015 129 s. ISBN 978-1-4842-1245-5

FACEBOOK, Inc. 2015. *Flux - In Depth Overview.* [online]. Retrieved April 13, 2017 from `https://facebook.github.io/flux/docs/in-depth-overview.html`

Redux. [online]. Retrieved April 13, 2017 from `https://aframe.io/docs/0.5.0/introduction/`

aframe-react. [Online]. Retrieved April 13, 2017 from https://github.com/aframevr/aframe-react/blob/master/README.md

Job Simulator revenue. [Online]. Retrieved April 17, 2017 from http://www.thesixthaxis.com/2017/01/09/job-simulators-3-million-in-sales-show-that-vr-can-be-profitable/

Unreal Engine 4. [Online]. Retrieved April 17, 2017 from https://www.unrealengine.com/unreal-engine-4

HTC Vive User Guide. [Online]. Retrieved April 19, 2017 from https://dl4.htc.com/Web_materials/Manual/Vive/Vive_User_Guide.pdf

The Body VR. [Online]. Retrieved April 19, 2017 from http://cdn.edgecast.steamstatic.com/steam/apps/451980/ss_e3e4e6850f6b76e1974aad3f65ea4a0eb04785d4.jpg

Job Simulator. [Online]. Retrieved April 19, 2017 from http://cdn.edgecast.steamstatic.com/steam/apps/448280/ss_ff7151cab14752f2b6501c31c3c79235b79cc45a.jpg

Calcflow. [Online]. Retrieved April 19, 2017 from http://cdn.edgecast.steamstatic.com/steam/apps/547280/ss_072b60fc7903197ffcb747600b90a3ccd1108e85.jpg

Appendices

Appendix A CD with MathworldVR code and Chromium browser

Appendix B System manual

Appendix C User manual

Appendix A

System manual

Hardware specification

To use HTC Vive, your computer must meet the following minimum system requirements.

- GPU: NVIDIA® GeForce® GTX 970, AMD Radeon™ R9 290 equivalent or better
- CPU: Intel® Core™ i5-4590/AMD FX™ 8350 equivalent or better
- RAM: 4 GB or more
- Video output: HDMI 1.4, DisplayPort 1.2 or newer
- USB port: 1x USB 2.0 or better port
- Operating system: Windows® 7 SP1, Windows® 8.1 or later, Windows® 10

HTC Vive installation

Detailed installation manual for HTC Vive virtual reality headset can be found at `https://dl4.htc.com/Web_materials/Manual/Vive/Vive_User_Guide.pdf` or by searching the Support page at `https://www.vive.com/us/support/`.

HTC Vive comes in a set including headset, base stations called *Lighthouses* for creating the laser field for precision tracking, hand controllers, USB and power cables.

Steam and SteamVR installation

For HTC Vive to work, user need to install Steam and SteamVR.

System requirements:

- Windows XP, Vista, 7, 8, 8.1 or 10.
- 512MB RAM minimum.
- 1GHz CPU or better.
- 1GB of free space on disk.
- Internet connection.

To download Steam, go to `http://store.steampowered.com/about/` and click on green button shown on the image below:

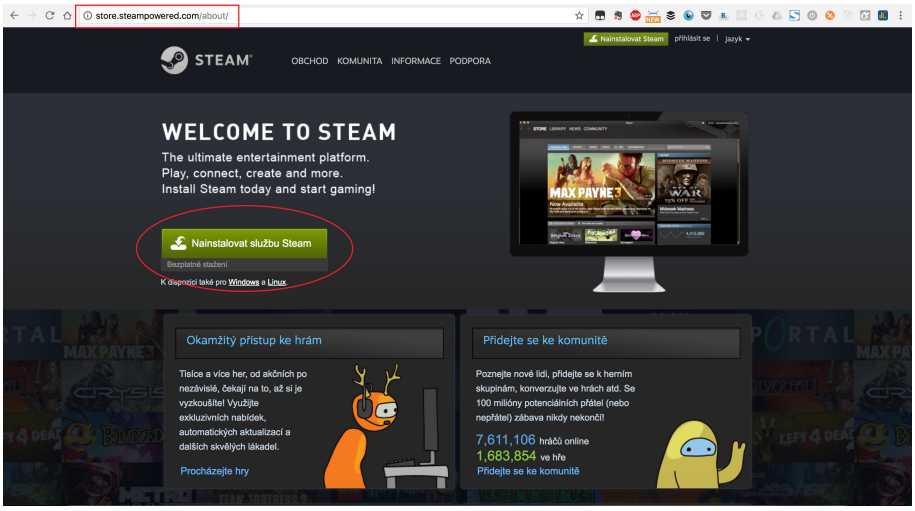

Figure 6 – 1 Steam download page.

After Steam download is finished, you can open the installer and follow instructions.

For SteamVR, open installed Steam, navigate to Library and search for "steamvr". It should find the installer and show it in list view under Tools. Clicking on it will show

a button in the right panel. Click on it to start SteamVR download, installation will begin automatically afterwards.

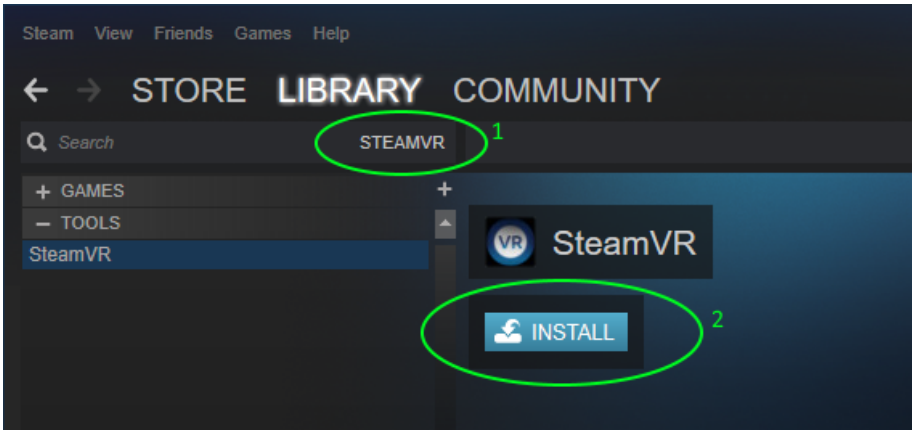

Figure 6 – 2 SteamVR installation button.

Room-scale tracking setup

After SteamVR is installed, room setup window will pop up automatically. For MathworldVR to work correctly, you need to select *Room-Scale* option and follow the tutorial.

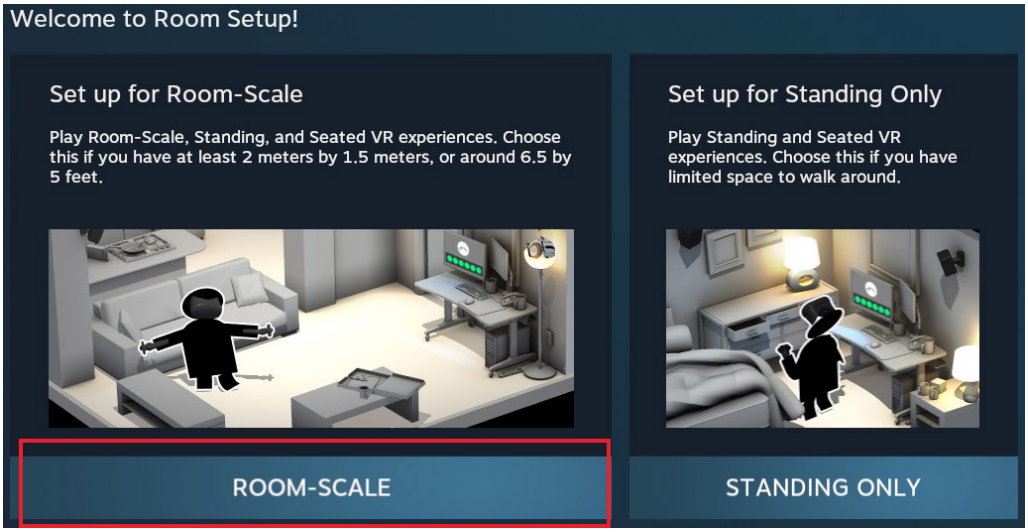

Figure 6 – 3 Room setup window.

WebVR-supported browser

During the time of writing of this thesis, only Chromium (experimental version of Google Chrome) and Mozilla Firefox Nightly browsers supported WebVR API. We included Chromium browser with MathworldVR application on the appended CD.

Chromium doesn't need to be installed. To open it, user just needs to double-click on `chrome.exe` file. In order to enable access to the WebVR APIs in Chromium, user must select the "Enabled" option from the drop-down menu for the "Enable WebVR" flag (enter `chrome://flags/enable-webvr` in the URL bar) and the "Gamepad Extensions" flag (enter `chrome://flags/enable-gamepad-extensions` in the URL bar), or launch Chromium from the command line with the `-enable-webvr` and

`-enable-gamepad-extensions` options.

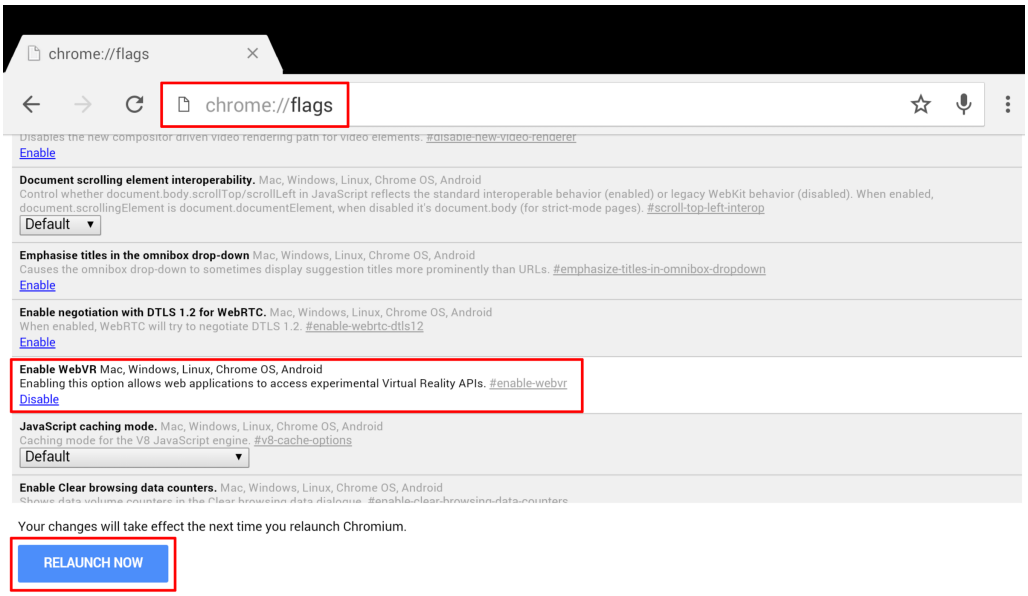

Figure 6 – 4 Enabling the WebVR API in Chromium browser.

Appendix B

User manual

MathworldVR user manual gives an assistance to people using it. It includes described images of VR headset, controllers usage and screenshots of virtual user interface.

HTC Vive headset

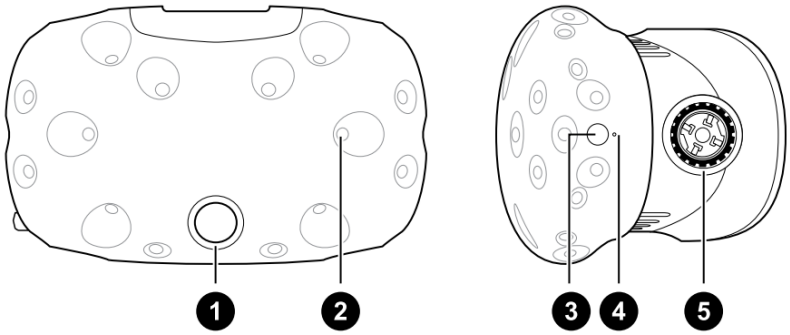

Figure 6 – 5 HTC Vive headset - technical detail. HTC Vive User Guide (2017)

1. Camera lens - can be used for viewing the "real world" from within the headset.
2. Tracking sensor - allowing Lighthouse base stations to "see" the headset.
3. Headset button - used for opening the SteamVR menu from within virtual reality view.
4. Status light - if the light produces red color, headset is not ready; if the light produces green color, headset is ready to be used.
5. Lens distance knob - used for changing the distance between lenses.

Hand controllers

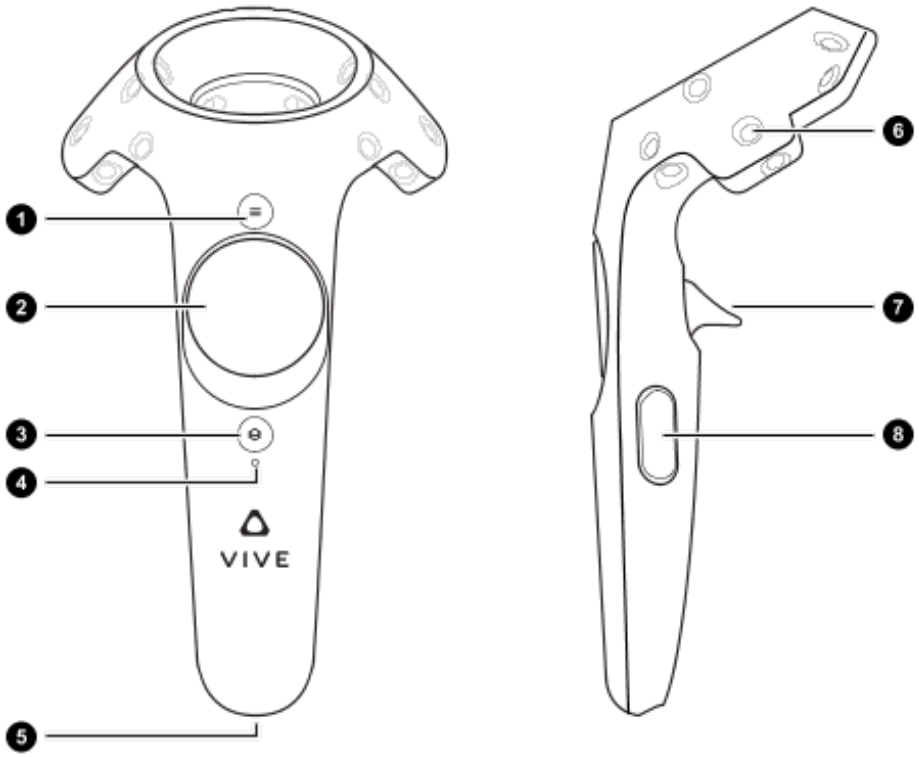

Figure 6 – 6 HTC Vive hand controllers - technical detail. HTC Vive User Guide (2017)

1. Menu button - This button doesn't have any effect in MathworldVR.

2. Trackpad - Allows user to move in scrolling menus, just like double-finger gesture on notebook trackpads. It's also clickable. Clicking and holding the trackpad on the left controller will trigger teleportation beam. User can point the beam where he wants to be teleported and then release the trackpad.

3. System button - Used for opening the SteamVR menu.

4. Status light - Indicates the status of a controller. Green color means it's ready, blinking orange color means it needs to be recharged and blue color means it needs to be synced with the Lighthouse base stations, because they cannot see

the controller.

5. Micro-USB port - Used for recharging the controller and also to upgrade its firmware.

6. Tracking sensor - Thanks to this sensor, Lighthouse base stations can see the controller. Tracking of HTC Vive controllers is very precise, because Lighthouse base stations emits laser beam that maps the space in front of them.

7. Trigger button - Used for selecting the options and moving the sliders in SettingsPanel component.

8. Grip button - Used for grabbing and scaling the parmetrized function. For grabbing interaction, only one of two controller's grip buttons needs to be pressed. For scaling interaction, both left and right controller's grip buttons needs to be pressed, scaling the grabbed object up and down according to movement of both controllers.

Interacting with parametrized function

Parametrized function can be grabbed, scaled and its variables changed in settings panel. To grab the function, you need to press the grip button on one of your VR controllers. You can move and rotate the function freely while grabbing it, just like objects in real life. To scale the function, you need to press grip buttons on both VR controllers imultaneously and move them fro or to each other. Moving controllers to each other will make the function smaller. Moving controllers from each other will make the function bigger.

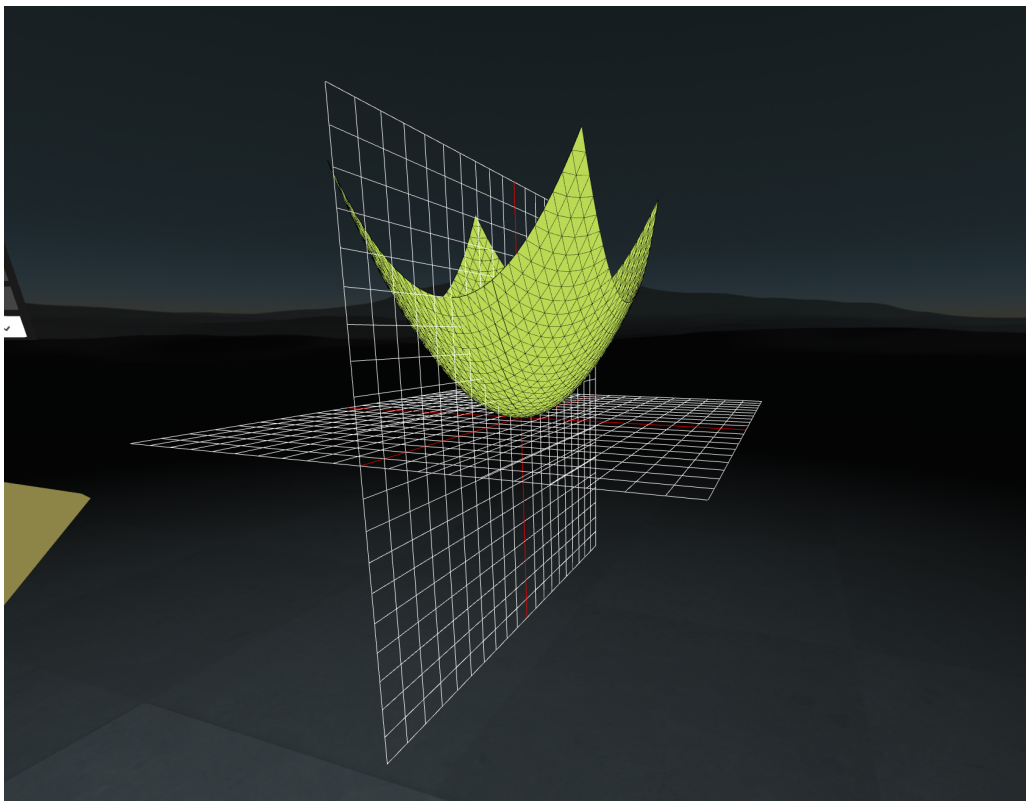

Figure 6 – 7 ParametrizedFunction component.

Changing function variables and color

To change the function variables or its color, you can use the SettingsPanel component. Pointing the VR controller at this panel will initiate the laser pointer. Move the pointer at any slider and press the trigger button. Holding and moving the controller will change the value of selected variable. You can also press and hold the grip button while pointing at SettingsPanel - this will automatically move the panel on top of your controller.

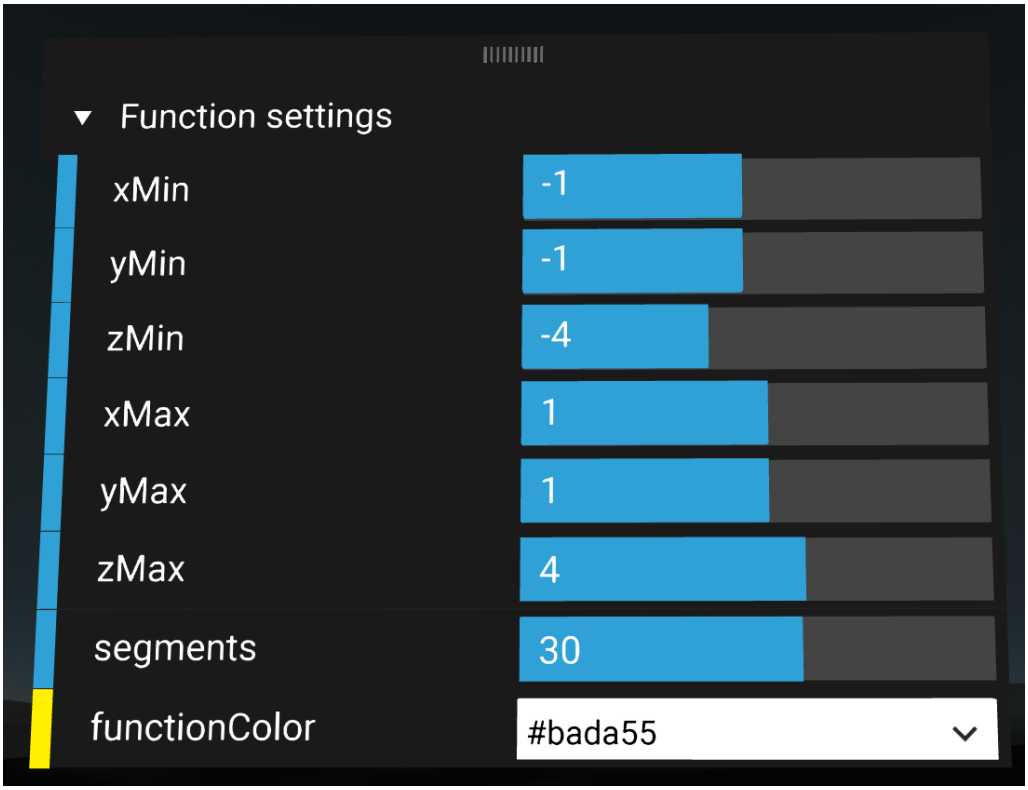

Figure 6 – 8 Settings component.

User input

You can change right side of the equation by simply clicking the buttons on virtual calculator. This component was designed to resemble real calculator and make user interaction with it more intuitive. After you write the function, you can update the 3D visualization by clicking the *Update* button on virtual calculator.

Figure 6 – 9 Calculator component.