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 Process of Obtaining and Processing Raw Materials
Field of study:	Gathering 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 24, 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. 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 for finding 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

Listings

Introduction

Our experimental project will be 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 space during lectures.

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. Application we'll be developing is called MathworldVR. 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 the Node.js, NPM, initialization of the project with NPM and installing 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 Theoretical analysis

Undoubtedly VR has attracted a lot of interest of people in last few years. Being a new paradigm of user interface it offers great benefits in many application areas. It provides an easy, powerful, 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 of course in the entertainment area.

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).

3 Analysis of current state

3.1 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.

3.2 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 2017. This has numerous very interesting applications, from virtual product tours and interactive training apps to immersive first person games. Unity, 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.

4 Technologies and tools for development

4.1 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

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

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)

4.2 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.

4.3 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.

4.4 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:

```
<html>
  <head>
    <script src="https://aframe.io/releases/0.5.0/aframe.min.js"></script>
  </head>
  <body>
    <a-scene>
      <a-box color="#6173F4" opacity="0.8" depth="2"></a-box>
      <a-sphere radius="2" src="texture.png" position="1 1 0"></a-sphere>
      <a-sky color="#ECECEC"></a-sky>
    </a-scene>
  </body>
</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)

4.4.1 JavaScript and Aframe

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

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

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)

4.4.2 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 A-Frame (2017) :

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

4.4.3 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)

4.4.4 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)

4.4.5 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

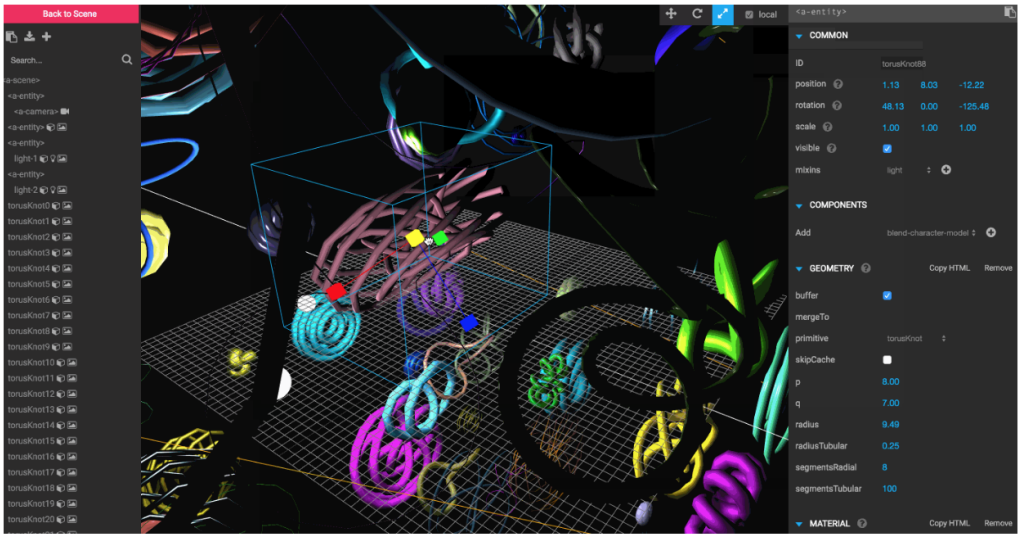

Figure 4 – 1 A-Frame Inspector window.

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)

4.4.6 Hardware specification

To use 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

4.5 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.

4.6 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.

4.7 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.

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.

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)

4.7.1 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)

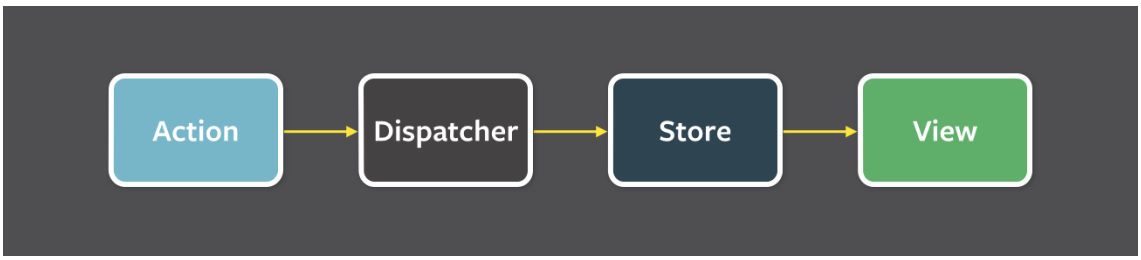

Figure 4 – 2 Data in a Flux application flows in a single direction.

The views may cause a new action to be propagated through the system in response to user interactions.

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.

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.

We found that two-way data bindings led to cascading updates, where changing one

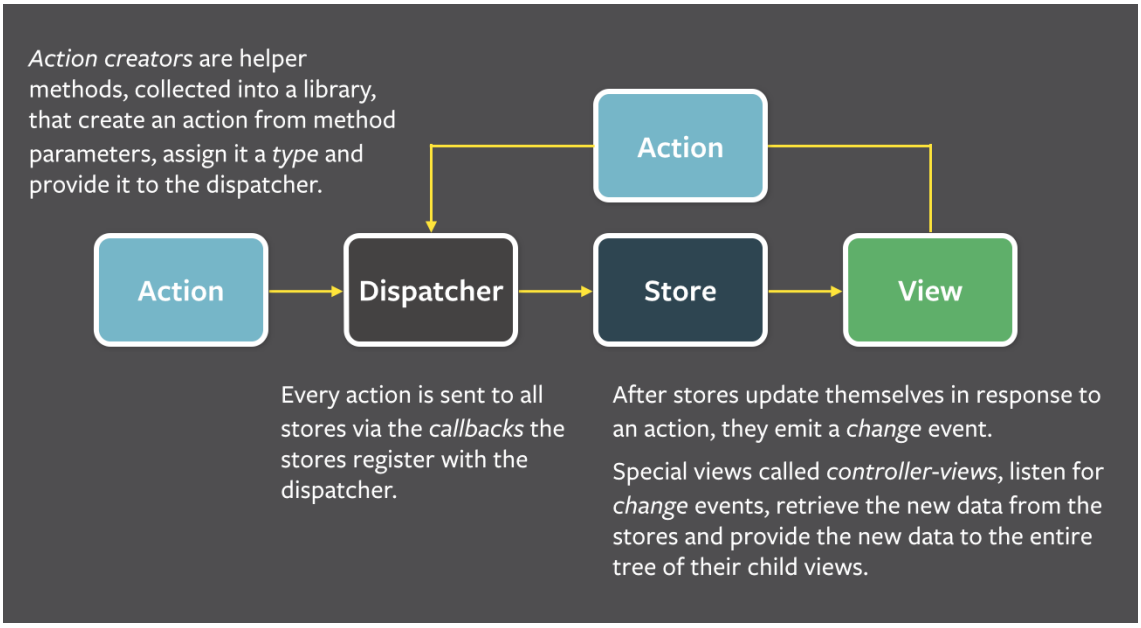

Figure 4 – 3 Detailed Flux application architecture diagram.

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)

4.8 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)

4.8.1 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.

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.

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.

4.9 Webpack

Webpack is a module bundler for modern JavaScript applications. It takes modules with dependencies and generates static assets representing those modules [4 – 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.

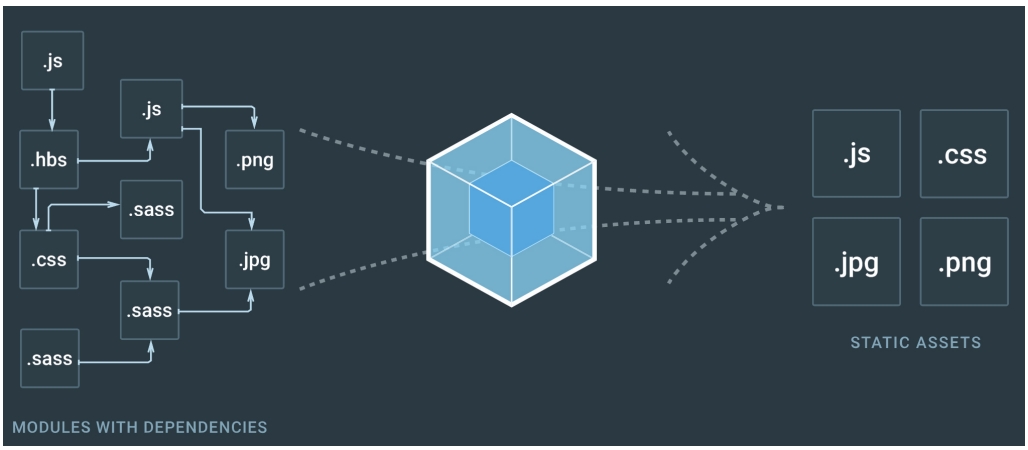

Figure 4 – 4 Bundling dependencies and static assets with Webpack

5 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.

5.1 Inputs and objections

5.2 Functionality planning

...zatial len par poznamok... - moznost pohybovat sa volne v 3D priestore - funkcia vykreslovania 3D parametrickej funkcie vo virtualnom priestore - potrebujeme moznost zapojenia ruk (ovladace HTC Vive/Oculus Touch) - nadvazujeme na to vyssie - potrebujeme 3D funkciu uchopit, otacat, zvacsovat/zmensovat - potrebujeme vediet menit/upravovat premenne funkcie - potrebujeme menit/upravovat constrains funkcie - potrebujeme prijmat user input - potrebujeme prijaty user input spojit s 3D parametrickou funkciou

5.3 Virtual user interface

6 Implementation of virtual reality application

6.1 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 }
```

6.2 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:

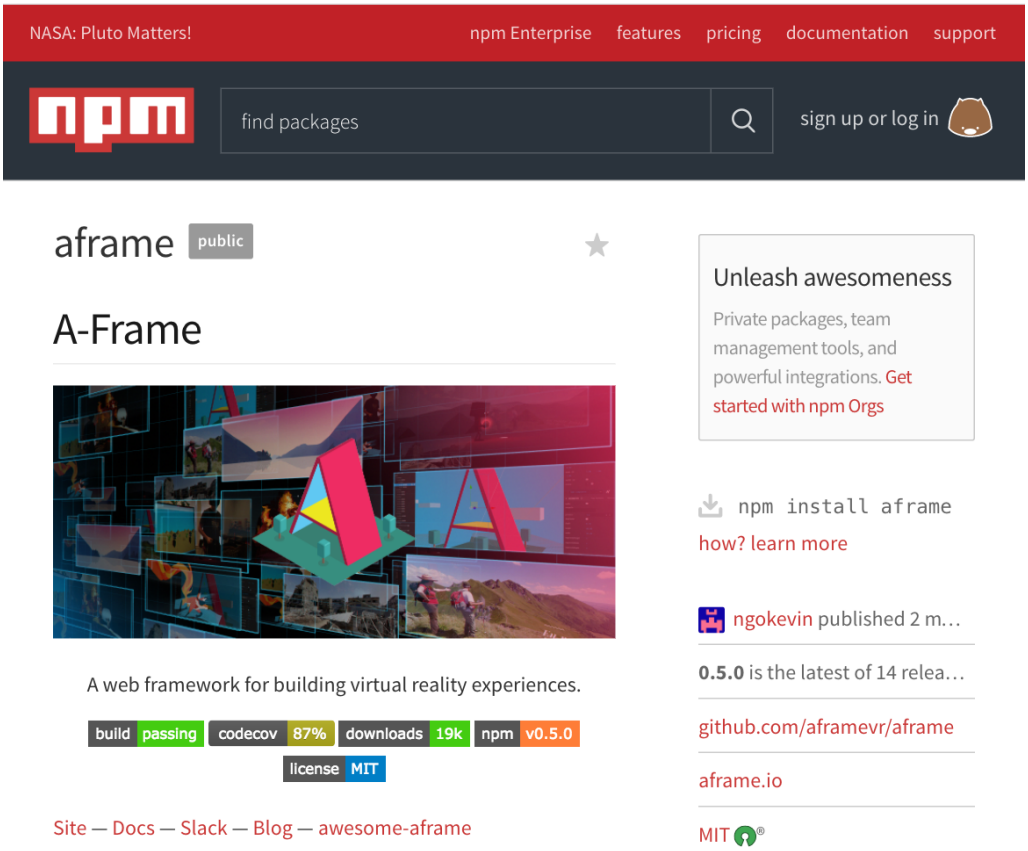

Figure 6 – 1 Detail page of JavaScript package in NPM registry with the script to install highlighted.

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.

6.3 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.

6.4 Project structure

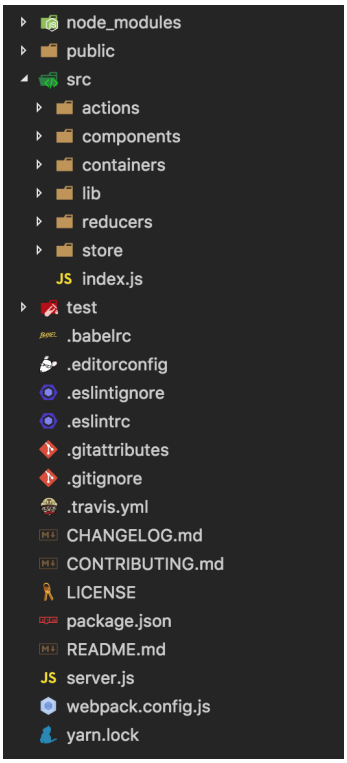

Figure 6 – 2 Project folder structure

6.5 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.

6.6 Webpack development server

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

```
1  const ip = '127.0.0.1';
2  const port = 3000;
3
4  new WebpackDevServer(webpack(config), {
5    publicPath: '/',
6    hot: true,
7    host: ip,
8    stats: false,
9    historyApiFallback: true,
10   contentBase: 'public',
11 }).listen(port, ip, (err) => {
12   if (err) {
13     return console.log(err);
14   }
15
16   console.log(`Listening at http://${ip}:${port}`);
17 });
```

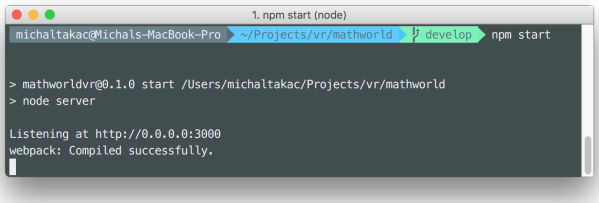

Figure 6 – 3 Development server is running.

6.7 Setting up Redux for application state management

- Actions
- Reducers
- Store

6.7.1 Initial state

In Redux, all the application state is stored as a single object. It's a good idea to think of its shape before writing any code.

6.7.2 Actions

Actions are payloads of information that send data from your application to your store. They are the only source of information for the store. You send them to the store using store.dispatch().

Actions are plain JavaScript objects. Actions must have a type property that indicates the type of action being performed. Types should typically be defined as string constants.

6.7.3 Action creators

Action creators are functions that create actions. In Redux, action creators return an action. It makes them portable and easy to test.

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

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

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

```
export const functionBoxSetPosition = (position) => ({
    type: FUNCTION_BOX_SET_POSITION,
    position
})
```

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

```
export const settingsSetXMin = (xMin) => ({
    type: SETTINGS_SET_X_MIN,
    xMin
})

export const settingsSetYMin = (yMin) => ({
    type: SETTINGS_SET_Y_MIN,
    yMin
```

```
9  })
10
11 export const settingsSetZMin = (zMin) => ({
12     type: SETTINGS_SET_Z_MIN,
13     zMin
14 })
15
16 export const settingsSetXMax = (xMax) => ({
17     type: SETTINGS_SET_X_MAX,
18     xMax
19 })
20
21 export const settingsSetYMax = (yMax) => ({
22     type: SETTINGS_SET_Y_MAX,
23     yMax
24 })
25
26 export const settingsSetZMax = (zMax) => ({
27     type: SETTINGS_SET_Z_MAX,
28     zMax
29 })
30
31 export const settingsSetSegments = (segments) => ({
32     type: SETTINGS_SET_SEGMENTS,
33     segments
34 })
35
36 export const settingsSetFunctionColor = (functionColor) => ({
37     type: SETTINGS_SET_FUNCTION_COLOR,
38     functionColor
39 })
```

```
1 export const uiAttentionboxToggle = () => ({
2     type: UI_ATTENTIONBOX_TOGGLE
3 })
```

```
export const userSetPosition = (position) => ({
    type: USER_SET_POSITION,
    position
})
```

6.7.4 Reducers

Actions describe the fact that something happened, but don't specify how the application's state changes in response. This is the job of reducers.

The reducer is a pure function that takes the previous state and an action, and returns the next state.

```
(previousState, action) => newState
```

It's called a reducer because it's the type of function you would pass to Array.prototype.reduce(redu initialValue). It's very important that the reducer stays pure. Things you should never do inside a reducer:

- Mutate its arguments.
- Perform side effects like API calls and routing transitions.
- Call non-pure functions, e.g. Date.now() or Math.random().

Reducer must be pure. Given the same arguments, it should calculate the next state and return it. No surprises, side effects, API calls or mutations, just a calculation.

```
export default (state = initialState, action) => {
  switch (action.type) {
    case ActionTypes.FUNCTION_BOX_SET_POSITION:
      return { ...state, position: action.position }
    default: return state
  }
```

```
}
```

Listing 1 *functionBox* reducer.

parametricFunction reducer:

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

settings reducer:

```
export default (state = initialState, action) => {
  switch (action.type) {
    case ActionTypes.SETTINGS_SET_X_MIN:
      return { ...state, xMin: action.xMin }
    case ActionTypes.SETTINGS_SET_Y_MIN:
      return { ...state, yMin: action.yMin }
    case ActionTypes.SETTINGS_SET_Z_MIN:
      return { ...state, zMin: action.zMin }
    case ActionTypes.SETTINGS_SET_X_MAX:
      return { ...state, xMax: action.xMax }
    case ActionTypes.SETTINGS_SET_Y_MAX:
      return { ...state, yMax: action.yMax }
    case ActionTypes.SETTINGS_SET_Z_MAX:
      return { ...state, zMax: action.zMax }
    case ActionTypes.SETTINGS_SET_Z_MAX:
      return { ...state, zMax: action.zMax }
    case ActionTypes.SETTINGS_SET_SEGMENTS:
      return { ...state, segments: action.segments }
    case ActionTypes.SETTINGS_SET_FUNCTION_COLOR:
      return { ...state, functionColor: action.functionColor }
    default: return state
```

```
22    }
23  }
```

ui reducer:

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

user reducer:

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

—

Note that:

- We don't mutate the state. We create a copy with ES6 spread operator "...".
- We return the previous state in the default case. It's important to return the previous state for any unknown action.

Then we join all reducers into one, root reducer which we export from 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
```

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

6.7.5 Store

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 subscribe(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.

Redux Store used in development:

```
import { createStore, applyMiddleware, compose } from 'redux'
import thunk from 'redux-thunk'
import createLogger from 'redux-logger'
import rootReducer from '../reducers'

const logger = createLogger({
  collapsed: true,
  duration: true,
})

const configureStore = initialState => {
  const hasWindow = typeof window !== 'undefined'

  const store = createStore(
    rootReducer,
    initialState,
    compose(
      applyMiddleware(thunk, logger),
      hasWindow && window.devToolsExtension ?
          window.devToolsExtension() : f => f
    )
  )

  if (module.hot) {
    // Enable Webpack hot module replacement for reducers
    module.hot.accept('../reducers', () => {
      const nextRootReducer = require('../reducers').default
      store.replaceReducer(nextRootReducer)
    })
```

```
29    }
30
31    return store
32  }
33
34  export default configureStore
```

Redux Store used in production:

```
1  import { createStore, applyMiddleware } from 'redux'
2  import thunk from 'redux-thunk'
3  import rootReducer from '../reducers'
4
5  const configureStore = initialState => createStore(
6    rootReducer,
7    initialState,
8    applyMiddleware(thunk)
9  )
10
11 export default configureStore
```

We'll also create additional configureStore.js file in which we'll iimplement conditional switching between Store for development and production according to NODE$_ENVvariableu$

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

6.8 A-Frame scene

```
1  export default class VRScene extends React.Component {
2    render() {
3      return (
```

```
4       <a-scene>
5         {this.props.children}
6       </a-scene>
7     )
8   }
9 }
```

We need to import additional A-Frame-specific libraries at the start of VRScene file:

```
1 // A-frame Components by community
2 import 'aframe'
3 import 'aframe-teleport-controls'
4 import 'super-hands'
5 import physics from 'aframe-physics-system'
6
7 // Libraries used by MathworldVR (Three.js, A-Frame, etc.)
8 import 'lib'
```

Here, we're loading aframe-physics-system which has specific initialization requirement - we need to call a function physics.registerAll() from within our scene 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, we add physics="gravity: 0" to <a-scene>:

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

6.9 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 2 *Camera* component code.

6.10 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 3 *Sky* component code.

6.11 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 4 *Plane* component code.

6.12 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 5 *Lights* component code.

6.13 Text component

6.14 CalcButton component

6.15 Calculator component

6.15.1 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 6 `calculator` action types.

6.15.2 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`

```
1 export const calculatorWriteText = (text) => ({
```

```
2       type: CALCULATOR_WRITE_TEXT,
3       text
4   })
```

Listing 7 Action for writing text to calculator display.

```
1   export const calculatorBackspace = () => ({
2       type: CALCULATOR_BACKSPACE
3   })
```

Listing 8 Action to remove character from calculator display.

```
1   export const calculatorClearText = () => ({
2       type: CALCULATOR_CLEAR_TEXT
3   })
```

Listing 9 Action to completely clear the calculator display.

6.15.3 Initial state

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

```
1   const initialState = {
2     displayText: 'x^2 + y^2',
3   }
```

Listing 10 Initial state of a *calculator*.

6.15.4 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 11 Addition of text to `state.calculator.displayText` when action of type $\texttt{CALCULATOR}_{WRITE_{TEXT}}$ *is dispatched.*

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

Listing 12 Remove one character from `state.calculator.displayText` when action of type $\texttt{CALCULATOR}_{BACKSPACE}$ *is dispatched.*

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

```
9 }
```

Listing 13 Clear the `state.calculator.displayText` when action of type `CALCULATOR`$_{C}LEAR_{T}EXT is dispatched$.

6.15.5 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 14 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 15 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 16 Creation of `Calculator` higher-order component.

6.15.6 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 will be 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
10       { /* --- BUTTONS --- */ }
11       { /* ... */ }
12
13     </Entity>
14   )
15 }
```

Listing 17 Presentational `Calculator` component code.

6.16 CalcButton component

```
1  import React from 'react'
2  import PropTypes from 'prop-types'
3  import { Entity } from 'aframe-react'
4  import { Text } from 'components'
5
6  export default class CalcButton extends React.Component {
7    constructor(props) {
8      super(props)
9
10     this.state = {
11       depth: 0.02,
12       opacity: 1,
13     }
14
15     this.startIntersection = this.startIntersection.bind(this)
16     this.endIntersection = this.endIntersection.bind(this)
17   }
18
19   startIntersection() {
20     const { actionToTrigger, value } = this.props
21     this.setState({ depth: 0, opacity: 0.2 })
22     actionToTrigger(value)
23   }
24
25   endIntersection() {
26     this.setState({ depth: 0.02, opacity: 1 })
27   }
28
29   render() {
30     const { id, position, width, height, buttonColor, text,
           textColor, textWidth } = this.props
31     const { depth, opacity } = this.state
32     return (
```

```
33        <Entity
34          id={id}
35          className="interactive"
36          geometry={{ primitive: 'box', height, width, depth }}
37          material={{ shader: 'flat', side: 'double', color:
                buttonColor, opacity }}
38          scale={{ x: 0.5, y: 0.5, z: 0.5 }}
39          position={position}
40          hoverable
41          clickable
42          events={{
43            'hover-start': this.startIntersection,
44            'hover-end': this.endIntersection,
45          }}
46        >
47          <Text
48            value={text}
49            color={textColor}
50            align="center"
51            zOffset={0.02}
52            width={textWidth}
53          />
54        </Entity>
55      )
56    }
57  }
```

6.17 Adding components together

6.18 ParametrizedFunction component

6.18.1 Extending A-Frame API with parametrizedfunction component

```
1  AFRAME.registerComponent('parametricfunction', {
```

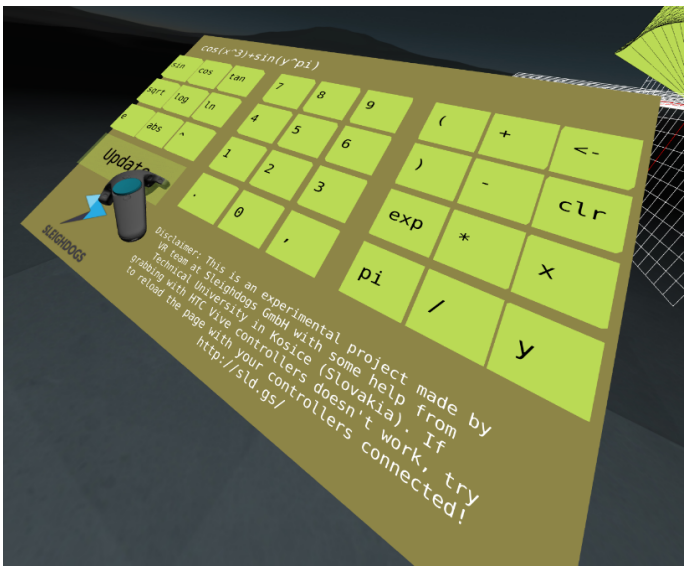

Figure 6 – 4 Calculator component

```
  schema: {/* ... */},
  init: function() {/* ... */},
  update: function() {/* ... */},
  remove: function() {/* ... */},
})
```

Listing 18 Registering new A-Frame component `'parametricfunction'`.

```
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 19 `parametricfunction` A-Frame component schema with default values.

```
init: function() {
```

```
2     const el = this.el;
3     const canvas = el.sceneEl.canvas;
4     // Wait for canvas to load.
5     if (!canvas) {
6       el.sceneEl.addEventListener('render-target-loaded',
            this.init.bind(this));
7       return;
8     }
9   }
```

Listing 20 `parametricfunction` A-Frame component init function.

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

Listing 21 `parametricfunction` A-Frame component init function.

```
1 try {
2   parser.eval(equation);
3 } catch (error) {
4   return;
5 }
```

Listing 22 `parametricfunction` A-Frame component init function.

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

Listing 23 `parametricfunction` A-Frame component init function.

```
1 this.mainMaterial = new THREE.MeshBasicMaterial( { color:
      this.data.functionColor, side: THREE.DoubleSide } );
2 this.wireframeMaterial = new THREE.MeshBasicMaterial( { color: 0
      x00008, wireframe: true, transparent: true } );
3 this.functionMaterial = [ this.mainMaterial,
      this.wireframeMaterial ];
```

Listing 24 `parametricfunction` A-Frame component init function.

6.18.2 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.

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

Listing 25 `parametricFunction` action types.

6.18.3 Action creators

ParametricFunction's action creators are pure functions that return payloads of information about type of $\texttt{PARAMETRIC}_{F}UNCTIONactionbeingdispatchedand, if needed, alsotheda$

```
1 export const parametricFunctionSetEquation = (equation) => ({
2   type: PARAMETRIC_FUNCTION_SET_EQUATION,
3   equation,
4 })
```

Listing 26 Action for setting the function for 3D visualization.

6.18.4 Initial state

`ParametricFunction component's initial state is defined in src/reducers/parametr`
`as a initialState object.`

```
1 const initialState = {
2   equation: 'x^2 + y^2',
3 }
```

Listing 27 Initial state of a *calculator*.

6.18.5 Reducer function

Reducer function for ParametricFunction component is defined in src/reducers/par
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$_F$$UNCTIONactions.Possible$ActionTypes$areconditio$

```
1 export default (state = initialState, action) => {
2   switch (action.type) {
3     // ...
4     case ActionTypes.PARAMETRIC_FUNCTION_SET_EQUATION:
5       return { ...state, equation: action.equation }
6    // ...
7     default: return state
8   }
9 }
```

Listing 28 Update the `state.parametricFunction.equation` value with the new one from action payload when action of type PARAMETRIC$_F$$UNCTION_S$$ET_E$$QUATIONisdispatched$.

6.18.6 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 29 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 30 Creation of `ParametricFunction` higher-order component.

6.18.7 Presentational component

Presentational ParametricFunction 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/compon Code preview displayed here doesn't include the CalcButton components, but they're utilizing the writeText, backspace and clearText actions. They will be explained in next section.

```
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 31 Presentational `ParametricFunction` component code.

6.19 SettingsPanel component

We will register the datgui A-Frame component first. Inside the init method, we're creating the dat.guiVR instance and injecting it 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, we'll create bindInput method that will create 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  }
```

Datgui component doesn't know about the VR hand controllers yet, thus we need to add them as an input objects into dat.guiVR instance:

```
1  const controllerRightEl = this.data.controllerRight;
2  if (controllerRightEl){
3    const right = dat.GUIVR.addInputObject(
       controllerRightEl.object3D);
4    scene.add(right);
5    bindInput(controllerRightEl, right);
6  }
7
8  const controllerLeftEl = this.data.controllerLeft;
9  if (controllerLeftEl){
10   const left = dat.GUIVR.addInputObject(
       controllerLeftEl.object3D);
11   scene.add(left);
12   bindInput(controllerLeftEl, left);
13 }
```

After the datgui component, we'll create

6.20 Hand controller components

Figure 6 – 5 SettingsPanel component

```
1  import React from 'react'
2  import { Entity } from 'aframe-react'
3
4  const RightController = (props) => {
5    return (
6      <Entity
7        id="rightController"
8        hand-controls="right"
9        sphere-collider={{ objects: '.interactive', radius: 0.05
             }}
10       if-no-vr-headset={{ visible: false }}
11       super-hands
12       static-body={{ shape: 'sphere', sphereRadius: 0.05 }}
13       {...props}
14     />
15   )
16 }
17
```

```
18 export default RightController
```

```
1  import React from 'react'
2  import { Entity } from 'aframe-react'
3
4  const LeftController = (props) => {
5    return (
6      <Entity
7        id="leftController"
8        hand-controls="left"
9        sphere-collider={{ objects: '.interactive', radius: 0.05
             }}
10       if-no-vr-headset={{ visible: false }}
11       teleport-controls
12       super-hands
13       static-body={{ shape: 'sphere', sphereRadius: 0.05 }}
14       {...props}
15     />
16   )
17 }
18
19 export default LeftController
```

6.21 Adding components into A-Frame scene

```
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>
```

6.22 Interactions

6.22.1 Grabbing

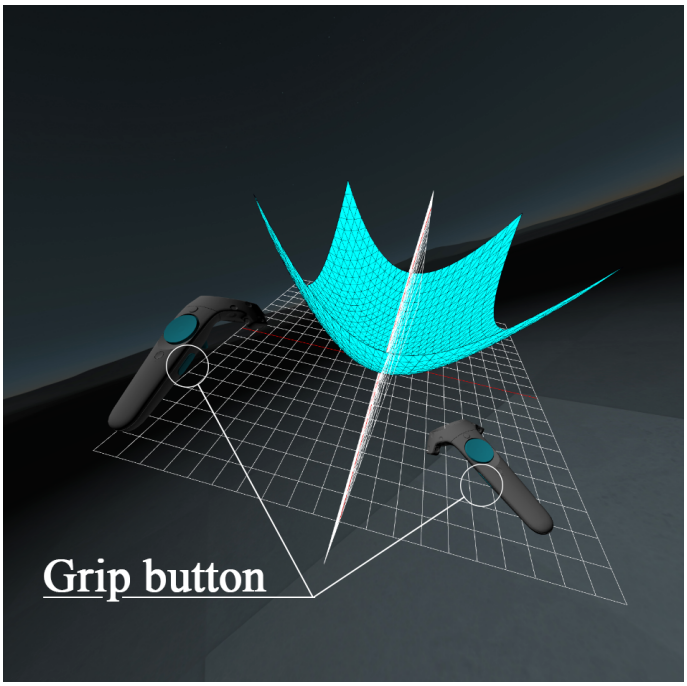

Figure 6 – 6 Grabbing functionality

6.22.2 Scaling

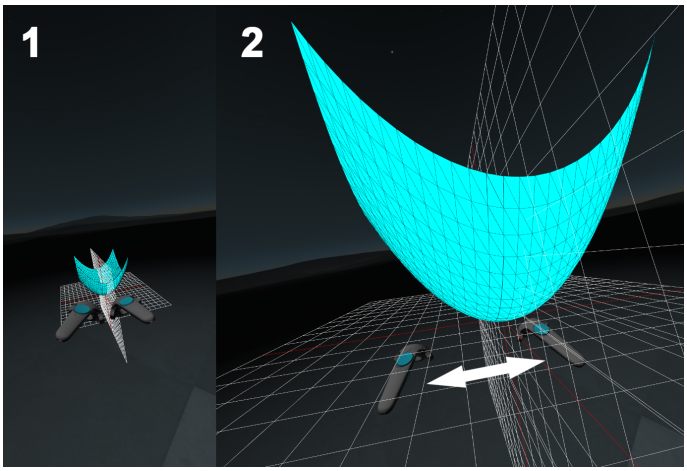

Figure 6 – 7 Scaling functionality

6.23 Parsing equations with Math.js

6.24 AttentionBox component

6.25 Building and deploying the application to web hosting

7 Conclusion

Táto časť záverečnej práce je povinná. Autor uvedie zhodnotenie riešenia. Uvedie výhody, nevýhody riešenia, použitie výsledkov, ďalšie možnosti a pod., prípadne načrtne iný spôsob riešenia úloh, resp. uvedie, prečo postupoval uvedeným spôsobom.

References

BECK, K. 2003. *Test-driven Development: By Example.* USA: Addison-Wesley Professional, 2003. 220 s. ISBN 9780321146533

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]. Available on internet: https://docs.npmjs.com/all

A-Frame Introduction [online]. Available on internet: https://aframe.io/docs/0.5.0/introduction/

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]. Available on*

internet: https://facebook.github.io/flux/docs/in-depth-overview.html

Redux [online]. Available on internet: https://aframe.io/docs/0.5.0/introduction/

Appendices

Appendix A Prílohy

Appendix B Bibliografické odkazy

Appendix C Vytvorenie zoznamu skratiek a symbolov

Appendix D

Appendix A

Prílohy (Appendices)

Táto časť záverečnej práce je povinná a obsahuje zoznam všetkých príloh vrátane elektronických nosičov. Názvy príloh v zozname musia byť zhodné s názvami uvedenými na príslušných prílohách. Tlačené prílohy majú na prvej strane identifikačné údaje - informácie zhodné s titulnou stranou záverečnej práce doplnené o názov príslušnej prílohy. Identifikačné údaje sú aj na priložených diskoch alebo disketách. Ak je médií viac, sú označené aj číselne v tvare I/N, kde I je poradové číslo a N je celkový počet daných médií. Zoznam príloh má nasledujúci tvar:

Appendix A CD médium - záverečná práca v elektronickej podobe, prílohy v elektronickej podobe.

Appendix B Používateľská príručka

Appendix C Systémová príručka

Prílohová časť je samostatnou časťou kvalifikačnej práce. Každá príloha začína na novej strane a je označená samostatným písmenom (Appendix A, Appendix B, ...). Číslovanie strán príloh nadväzuje na číslovanie strán v hlavnom texte. Pri každej prílohe sa má uviesť prameň, z ktorého sme príslušný materiál získali.

Appendix B

Bibliografické odkazy

Táto časť záverečnej práce je povinná. V zozname použitej literatúry sa uvádzajú odkazy podľa normy STN ISO 690-2 (01 0197) (Informácie a dokumentácia. Bibliografické citácie. Časť 2: Elektronické dokumenty alebo ich časti, dátum vydania 1. 12. 2001, ICS: 01.140.20). Odkazy sa môžu týkať knižných, časopiseckých a iných zdrojov informácií (zborníky z konferencií, patentové dokumenty, normy, odporúčania, kvalifikačné práce, osobná korešpondencia a rukopisy, odkazy cez sprostredkujúci zdroj, elektronické publikácie), ktoré boli v záverečnej práci použité.

Forma citácií sa zabezpečuje niektorou z metód, opísaných v norme STN ISO 690, 1998, s. 21. Podrobnejšie informácie nájdete na stránke http://www.tuke.sk v záložke Výsledky práce/Prehľad normy pre publikovanie STN ISO 690 a STN ISO 690-2.

Existujú dva hlavné spôsoby citovania v texte.

- Citovanie podľa mena a dátumu.
- Citovanie podľa odkazového čísla.

Preferovanou metódou citovania v texte vysokoškolskej a kvalifikačnej práce je podľa normy ISO 7144 citovanie podľa mena a dátumu (??). V tomto prípade sa zoznam použitej literatúry upraví tak, že za meno sa pridá rok vydania. Na uľahčenie vyhľadávania citácií sa zoznam vytvára v abecednom poradí autorov.

Príklad: ...podľa (?) je táto metóda dostatočne rozpracovaná na to, aby mohla byť všeobecne používaná v ...

Druhý spôsob uvedenia odkazu na použitú literatúru je uvedenie len čísla tohto zdroja v hranatých zátvorkách bez mena autora (autorov) najčastejšie na konci príslušnej vety alebo odstavca.

Príklad: ...podľa [13] je táto metóda dostatočne rozpracovaná na to, aby mohla byť všeobecne používaná v ...ako je uvedené v [14].

Citácie sú spojené s bibliografickým odkazom poradovým číslom v tvare indexu alebo čísla v hranatých zátvorkách. Odkazy v zozname na konci práce budú usporiadané podľa týchto poradových čísel. Viacero citácií toho istého diela bude mať rovnaké číslo. Odporúča sa usporiadať jednotlivé položky v poradí citovania alebo podľa abecedy.

Rôzne spôsoby odkazov je možné dosiahnuť zmenou voľby v balíku natbib:

```
% Citovanie podla mena autora a roku
\usepackage[]{natbib}\citestyle{chicago}
% Možnosť rôznych štýlov citácií. Príklady sú uvedené
% v preambule súboru natbib.sty.
% Napr. štýly chicago, egs, pass, anngeo, nlinproc produkujú
% odkaz v tvare (Jones, 1961; Baker, 1952). V prípade, keď
% neuvedieme štýl citácie (vynecháme \citestyle{}) v "options"
% balíka natbib zapíšeme voľbu "colon".
```

Keď zapneme voľbu numbers, prepneme sa do režimu citovania podľa odkazového čísla.

```
% Metoda ciselnych citacii
\usepackage[numbers]{natbib}
```

Pri zápise odkazov sa používajú nasledujúce pravidlá:

V odkaze na knižnú publikáciu (pozri príklad zoznamov na konci tejto časti):

- Uvádzame jedno, dve alebo tri prvé mená oddelené pomlčkou, ostatné vynecháme a namiesto nich napíšeme skratku et al. alebo a i.
- Podnázov sa môže zapísať vtedy, ak to uľahčí identifikáciu dokumentu. Od názvu sa oddeľuje dvojbodkou a medzerou.
- Dlhý názov sa môže skrátiť v prípade, ak sa tým nestratí podstatná informácia. Nikdy sa neskracuje začiatok názvu. Všetky vynechávky treba označiť znamienkami vypustenia „..."

Pri využívaní informácií z elektronických dokumentov treba dodržiavať tieto zásady:

- uprednostňujeme autorizované súbory solídnych služieb a systémov,
- zaznamenáme dostatok informácií o súbore tak, aby ho bolo opäť možné vyhľadať,
- urobíme si kópiu použitého prameňa v elektronickej alebo papierovej forme,
- za verifikovateľnosť informácií zodpovedá autor, ktorý sa na ne odvoláva.

Pre zápis elektronických dokumentov platia tie isté pravidlá, ako pre zápis „klasických". Navyše treba uviesť tieto údaje:

- druh nosiča [online], [CD-ROM], [disketa], [magnetická páska]
- dátum citovania (len pre online dokumenty)
- dostupnosť (len pre online dokumenty)

Poradie prvkov odkazu je nasledovné: Autor. Názov. In Názov primárneho

zdroja: Podnázov. [Druh nosiča]. Editor. Vydanie alebo verzia. Miesto vydania : Vydavateľ, dátum vydania. [Dátum citovania]. Poznámky. Dostupnosť. ISBN alebo ISSN.