Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Vizualizácia experimentálnych údajov v zdieľanej rozšírenej realite a jej používateľské rozhranie

Bakalárska práca

2021 **Martin Fekete**

Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Vizualizácia experimentálnych údajov v zdieľanej rozšírenej realite a jej používateľské rozhranie

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	9.2.1. Informatika
Školiace pracovisko:	Katedra počítačov a informatiky (KPI)
Školiteľ:	Ing. Štefan Korečko, PhD.
Konzultant:	RNDr. Martin Vaľa, PhD.

Košice 2021 **Martin Fekete**

Abstrakt v SJ

Táto bakalárska práca skúma možnosti vizualizácie dát vo virtuálnej realite, pričom hlavným cieľom je využiť už existujúcu vizualizáciu knižnicou JSROOT. V úvode práce je analýza knižnice JSROOT, ktorá je základným pilierom tejto práce, doplnená o analýzu softvérového rámca pre virtuálnu realitu na webe. Analýza pozostáva aj z prevedených experimentov, ktorých výsledky sa neskôr uplatnia, aj pri implementácií. V ďalšej časti práce dochádza k návrhu riešenia, kde sa zohľadňujú poznatky z analýzy a vykonaných experimentov a formuje sa tak ucelené riešenie, ktorého úlohou je zlúčiť rôzne technológie a zabezpečiť komunikáciu medzi rozdielnymi vrstvami architektúry. Okrem samotného riešenia vizualizácie dát bude predmetom aj používateľské rozhranie. Z hlavnej časti sa táto práca dá klasifikovať ako experiment, preto dosiahnuté výsledky budú dôkladne otestované a vyhodnotené. V záverečnej časti sa k vyhodnoteniu určia aj možnosti pre rozširovanie projektu v budúcnosti a ďalšie smerovanie.

Kľúčové slová v SJ

histogram, A-Frame, JSROOT, React, rozšírená realita, virtuálna realita

Abstrakt v AJ

This bachelor thesis examines the possibility of visualization of data in a virtual reality, while the main goal is to use the already existing Visualization of JSROOT library. In the beginning of the thesis is a library of the JSROOT library, which is a fundamental pillar of this work, supplemented with an analysis of the software frame for virtual reality on the web. The analysis consists of converted experiments whose results will be applied later, even when implementing. In the next part of the thesis there is a proposal of a solution where knowledge of analysis and experiments are taken into account and it is formed a comprehensive solution whose task is to merge various technologies and ensure communication between different layers of architecture. In addition to the data visualization solution itself, the user interface will also be subject. From the main section, this work can be classified as an experiment, so the results achieved will be thoroughly tested and evaluated. In the final section, the options for the expansion of the project in the future and other routing are also designed to evaluate.

Kľúčové slová v AJ

histogram, A-Frame, JSROOT, React, extended reality, virtual reality

Bibliografická citácia

FEKETE, Martin. *Vizualizácia experimentálnych údajov v zdieľanej rozšírenej realite a jej používateľské rozhranie* . Košice: Technická univerzita v Košiciach, Fakulta elektrotechniky a informatiky, 2021. 77s. Vedúci práce: Ing. Štefan Korečko, PhD.

62508

TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Katedra počítačov a informatiky

ZADANIE BAKALÁRSKEJ PRÁCE

Študijný odbor: **Informatika**
Študijný program: **Informatika**

Názov práce:

Vizualizácia experimentálnych údajov v zdieľanej rozšírenej realite a jej používateľské rozhranie

Experimental Data Visualization in Shared Extended Reality and its User Interface

Študent: **Martin Fekete**
Školiteľ: **Ing. Štefan Korečko, PhD.**
Školiace pracovisko: **Katedra počítačov a informatiky**
Konzultant práce: **RNDr. Martin Vaľa, PhD.**
Pracovisko konzultanta:

Pokyny na vypracovanie bakalárskej práce:

1. Analyzovať softvérovú knižnicu JSROOT a webový rámec A-Frame z hľadiska ich použitia pre vizualizáciu experimentálnych údajov v rozšírenej realite.
2. Navrhnúť webový softvérový komponent pre vizualizáciu experimentálnych údajov v rozšírenej realite a to vo forme vybraných typov grafov a vrátane používateľského rozhrania.
3. Softvérový komponent implementovať s použitím A-Frame.
4. Implementované riešenie overiť z hľadiska výkonu a spokojnosti potencionálnych používateľov.
5. Vypracovať dokumentáciu podľa pokynov vedúceho práce.

Jazyk, v ktorom sa práca vypracuje: slovenský
Termín pre odovzdanie práce: 28.05.2021
Dátum zadania bakalárskej práce: 30.10.2020

prof. Ing. Liberios Vokorokos, PhD.
dekan fakulty

Čestné vyhlásenie

Vyhlasujem, že som záverečnú prácu vypracoval(a) samostatne s použitím uvedenej odbornej literatúry.

Košice, 13.5.2021 .....

Vlastnoručný podpis

Poďakovanie

Na tomto mieste by som veľmi rád poďakoval svojmu vedúcemu práce za jeho čas, trpezlivosť a odborné vedenie počas riešenia mojej záverečnej práce a konzultantovi za jeho rady, nápady a úsilie, bez ktorého by sme len ťažko dosiahli želané výsledky.

Rovnako by som sa rád poďakoval svojim rodičom a priateľom za ich neustálu podporu počas celého môjho štúdia.

V neposlednom rade by som sa rád poďakoval pánom *Donaldovi E. Knuthovi* a *Leslie Lamportovi* za typografický systém LaTeX, ktorý mi ušetril veľmi veľa času pri písaní práce.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Vo fyzikálnych experimentoch na urýchľovačoch častíc sa pracuje s obrovským množstvom dát. Okrem samotného experimentovania, ukladania týchto dát na disky obrovských kapacít nesmieme zabúdať aj na dôležitosť prezentácie dát. Pod pojmom prezentácia dát nám bežne napadnú každodenne využívané metódy ako zobrazovanie dát v tabuľkách či grafoch. Na týchto metódach prezentovania dát nie je nič zlé, sú jednoduché, prehľadné, ľahko sa s nimi pracuje a dajú sa využívať na zobrazovanie rôznych typov dát.

Avšak pri experimentoch, v ktorých je potrebné pracovať s obrovským množstvom dát je často potrebné, aby dáta boli dobre organizované a aby sa k nim používateľ vedel dostať systematicky. Na vizualizáciu údajov vieme využiť histogramy, ktorých body nazývané biny, určujú frekvenciu výskytu hodnôt na danom intervale. Na základe frekvencie, ktorá je reprezentovaná zovňajškom binu, vieme zistiť všetky potrebné informácie, na základe ktorých potom vieme zobraziť histogramy s konkrétnejšími dátami pre zvolený interval.

Tento princíp je efektívnejší a prehľadnejší, ale o to náročnejší na implementáciu. Dnes svetu vládnu rôzne aplikácie založené na 2D zobrazení, kde si môžeme zobrazovať neustále viac dát. Po kliknutí na bin sa nám zobrazia ďalšie histogramy s podrobnejšími údajmi. Predstavme si však situáciu, kedy máme zobrazený histogram s veľkým počtom binov a po kliknutí na ktorýkoľvek z nich sa nám zobrazí ďalší pomerne rozsiahly histogram alebo viac histogramov a plocha na zobrazenie nie je dostatočne veľká na to, aby boli v rovnakom čase zobrazené všetky potrebné dáta.

V súčasnosti však čoraz častejšie počujeme pojem virtuálna realita VR alebo všeobecnejšie rozšírená realita AR, ktorá postupne nahrádza 2D zobrazenie a mnoho projektov je už integrovaných do virtuálneho prostredia. 3D prostredie virtuálnej reality poskytuje dostatočný priestor pre zobrazenie dát a zároveň poskytuje aj zážitok pre nás ako používateľov. Pri použití vhodných zariadení by sme mohli stáť a prechádzať sa medzi binmi v histograme a po kliknutí na bin následne zobraziť ďalšie príslušné histogramy, ktoré by sa nám mohli zobraziť

po stranách a takisto by sme vedeli manipulovať s histogramami použitím špeciálnych ovládačov, a to prirodzenými pohybmi, ktorými v každodennom živote manipulujeme s vecami. Zaujímavé by bolo aj využitie zdieľanej virtuálnej reality, ktorá by nám umožnila vytvoriť virtuálne zdieľané 3D prostredie, v ktorom by sa mohli ľudia podieľajúci sa na experimentoch stretávať vo virtuálnych miestnostiach v reálnom čase, bezohľadu na to kde vo svete sa nachádzajú a zobrazovať si údaje, ktoré by boli viditeľné pre všetkých v miestnosti.

Našou úlohou je preskúmať možnosti zobrazenia týchto dát vo virtuálnej realite a následne vytvoriť prototypovú implementáciu, ktorá nám poslúži pre prehľadnejšie, efektívnejšie zobrazovanie dát a poskytne nám aj vizuálny zážitok pri tejto prezentácií dát.

Formulácia úlohy

Cieľom tejto práce je preskúmať možnosti zobrazenia dát z fyzikálnych experimentov na urýchľovačoch častíc v Dubne respektíve v CERN do zdieľanej virtuálnej reality a následne realizovať prototypovú implementáciu vizualizácie dát. Ako platforma pre samotnú vizualizáciu by mal byť použitý systém LIRKIS G-CVE, ktorý je vyvíjaný použitím knižníc A-Frame a Networked Aframe na FEI KPI.

Prvoradou úlohou bude preskúmať vizualizáciu údajov, ktorú poskytuje knižnica JSROOT a pokúsiť sa ju integrovať do virtuálnej reality na webe. Ďalšou úlohou bude vymyslieť efektívny systém, ktorý nám umožní tieto dáta zobrazovať interaktívne a v primeranom množstve tak, aby v istom čase bolo zobrazené len určité a potrebné množstvo dát a aby sa vhodným ovládaním tieto dáta obmieňali a hlavne, aby sme zabezpečili dobrý výkon samotnej aplikácie. Medzi úlohy patrí aj návrh efektívneho a jednoduchého používateľského rozhrania.

Nakoniec by sme mali všetky vytvorené funkcie zovšeobecniť a vytvoriť knižnicu alebo softvérový rámec, aby bolo možné využívať tieto funkcie aj na vizualizáciu iných dát s podobným charakterom poprípade preskúmať ešte iné možnosti zapuzdrenia dát do komponentu pre jednoduché použitie v budúcnosti.

1 Analytická časť

V prvom rade potrebujeme analyzovať dáta, ktoré sú prezentované ako výsledok fyzikálnych experimentov na urýchľovačoch častíc v CERN-e. Počas tejto časti analýzy sa budeme opierať o odborné informácie ale taktiež budeme aj konzultovať s RNDr. Martinom Vaľom, PhD z Univerzity Pavla Jozefa Šafárika v Košiciach, ktorý sa podieľa na týchto experimentoch. Počas práce sa budú využívať jeho cenné poznatky, rady a skúsenosti, aby sa vytvorila vizualizácia poskytujúca, čo najviac potrebných funkcionalít a aby samotná vizualizácia poskytovala, čo najlepší používateľský zážitok.

1.1 Analýza existujúcich riešení

Vzhľadom na to, že máme za úlohu integrovať dáta, ktoré už sú zobrazované na webe a sú súčasťou rôznych aplikácií, sme sa pokúsili nájsť nejaké riešenia, ktoré zahŕňajú aj samotnú virtuálnu realitu. Vyskúšali sme zadávať do prehliadača výrazy spájajúce JSROOT [1], A-Frame [2] a podobné súvislosti. Skúmali sme aj všeobecnejšie riešenia ako len zobrazovanie dát vo virtuálnej realite bez použitia práve A-Frame. Konkrétne sme nenašli žiadne riešenia, ktoré by sme vedeli nejakým spôsobom použiť pri riešení nášho problému týkajúceho sa zobrazovania dát, ktoré sú už zobrazené pomocou funkcií knižnice JSROOT. Po komunikácii s fyzikom z UPJŠ sme zistili, že samotné riešenie tohto problému má charakter experimentu. Riešenie nie je reakciou na iné riešenia konkurencie.

Z toho dôvodu sme sa rozhodli rozdeliť samotnú analýzu na dve časti. V prvej sa zameráme na preštudovanie jednotlivých oblastí, a to analýzu zobrazovaných dát, knižnice JSROOT, samotného webového rámca pre virtuálnu realitu a ďalšie webové technológie, ktoré nám pomôžu vytvoriť webové používateľské rozhranie. Následne v druhej časti analýzy sa pokúsime na základe zistených informácií z prvej časti analýzy nájsť súvislosti, ktoré nám umožnia tieto histogramy zobraziť v tomto webovom softvérovom rámci. V tejto časti sa pokúsime aj otestovať rôzne varianty riešení a následne porovnať a vybrať riešenie, ktoré bude najefektívnej-

šie z hľadiska počtu riadkov kódu a aj výkonu samotnej aplikácie. Tieto poznatky potom neskôr použijeme pre návrh riešenia.

1.2 Analýza vizualizácie dát z fyzikálnych experimentov v CERN

Predtým ako sa pustíme do samotnej analýzy prezentovaných dát v CERN-e si stručne predstavíme samotný CERN. Ako je uvedé priamo na hlavnej stránke CERN-u [3] hovoríme o laboratoriu v ktorom fyzici a inžinieri používajú najkomplexnejšie vedecké prístroje na svete slúžiace na štúdium základných zložiek hmoty. Subatomárne častice sa zrážajú blízko seba v rýchlostiach, ktoré dosahujú svetelné lúče. Taktiež je na stránke CERN-u [3] objasnený pojem LHC ktorý definuje najväčší a najvýkonnejší urýchľovač častíc na svete. Celkovo sa v CERN-e pracuje na 7 experimentoch na urýchľovači častíc LHC a to na experimentoch ATLAS, CMS, ALICE, LHCb, TOTEM, LHCf, MoEDA [4]. Tieto experimenty na urýchľovačoch častíc sú charakteristické generovaním obrovského množstva dát, ktoré pribúdajúcimi rokmi stále narastá.

1.2.1 Charakteristika dát

V týchto experimentoch sa neustále zvyšuje množstvo spracovaných dát. Obrovské množstvo dát spracovávajú výpočtové jednotky zložené z viacerých počítačov nazývané klastre, keďže sa jedná o výpočty pre ktoré nepostačuje jeden počítač. Pre správu a monitorovanie klastrov sú používané dávkovacie systémy a monitorovacie aplikácie. Táto problematika je podrobnejšie opísaná v bakalárskej práci Webové rozhranie pre systém SALSA [5] v ktorej je riešený problém zjednotenia monitorovacej aplikácie a dávkovacieho systému do webovej aplikácie.

Zdroj [6] uvádza, že prevažná väčšina dát je určená len na čítanie. Samotné dáta sú zapisované počas experimentov a ukladané pri vysokých rýchlostiach. Dáta sa potom už nemodifikujú. Údaje sú uchovávané na viacerých úložiskách po celom svete. Pre vizualizáciu údajov sú následne vytvárané dátové analýzy, ktoré sú zobrazované formou histogramov a grafov. Ako je uvedené v publikácii [7] tak histogramy sú dôležité pre zobrazovanie a sumarizáciu dát. Stĺpce histogramu sú známe ako biny a využívajú sa najmä v oblastiach, kde sa pracuje s obrovským množstvom dát.

Na stránke [8] je možnosť si zobraziť vzorové histogramy. Preto sme sa rozhodli preskúmať tieto zobrazenia. Na stránke sú jednotlivé grafy, histogramy, pro-

jekcie rozdelené do kategórii a každá kategória obsahuje príklad. Po preskúmaní príkladov zobrazovania údajov sme zistili, že je tam podstatne viac nástrojov na prezentáciu dát, a to 2D histogramy, 3D histogramy, projekcie.

Z toho dôvodu by bolo podrobné analyzovanie a študovanie každého nástroja na zobrazenie údajov podrobne časovo náročné. Preto sme sa rozhodli zistiť na aké zobrazenia sa máme zamerať v tejto práci. Výber prioritných zobrazení sme konzultovali s RNDr. Martinom Vaľom, PhD. Pri konzultácii sme zistili, že nie každá forma zobrazenia dát vo virtuálnej realite by mohla byť prínosom špeciálne v prípade zobrazenia 2D histogramu vo virtuálnej realite. Mohol by nastať problém pri samotnej interakcií vedcov s týmto histogramom vo virtuálnej realite. Výsledok konzultácie je obsiahnutý v nasledujúcich bodoch.

- Zobrazenie histogramu s označením TH3 vo virtuálnej realite, ktoré bude hlavným cieľom tejto práce
- Zobrazenie histogramu s označením TH2 vo virtuálnej realite, ktoré bude taktiež patriť medzi ciele tejto práce
- Zobrazenie 3D projekcií vo virtuálnej realite, ktoré sa bude tiež riešiť v tejto práci
- Cieľom bude aj preskúmanie možností vizualizácie 2D histogramov s označením TH1. 2D

V ďalších častiach sa preto budeme sústrediť špeciálne na vizualizáciu histogramov s označeniami TH1, TH2, TH3 a samotných projekcií.

1.2.2 Softvérový rámec ROOT

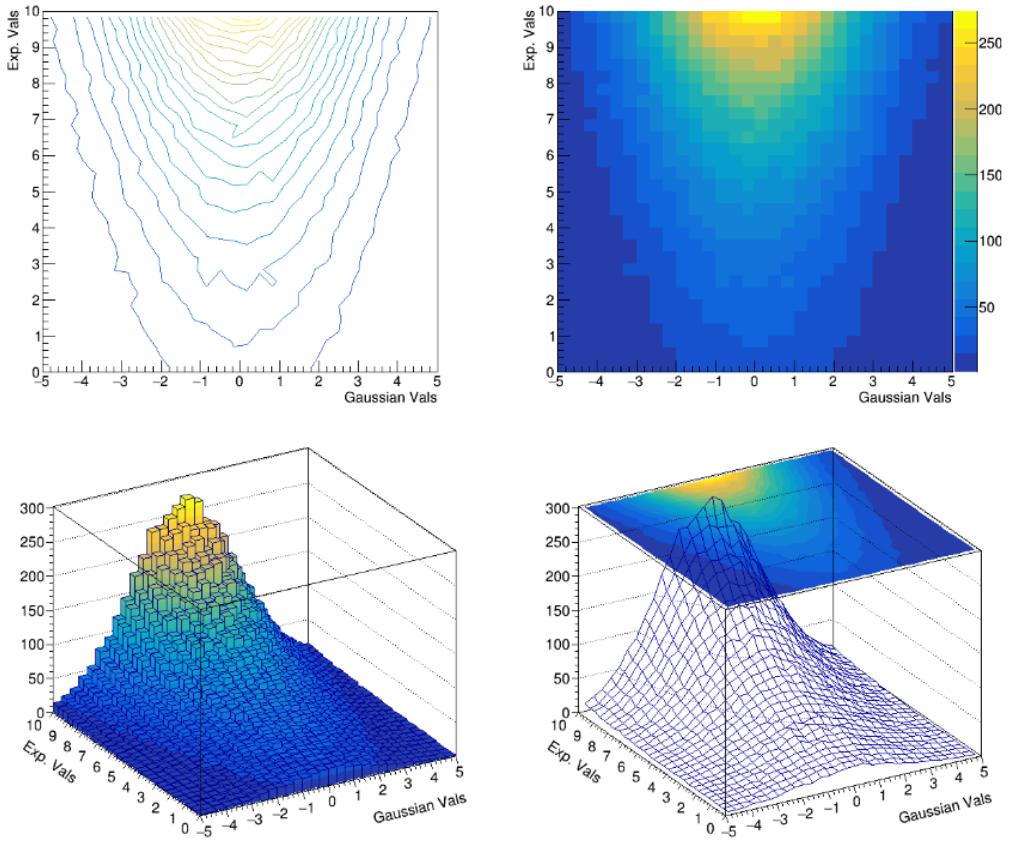
V tejto časti analýzy budeme čerpať informácie prevažne z dokumentácie softvérového rámca ROOT [9]. Ako je uvedené hneď v úvode dokumentácie, ROOT je objektovo-orientovaný softvérový rámec, ktorý bol vyvinutý na účely monitorovania, uchovávania a analýzy dát. Ako je ďalej spomenuté, je využívaný v laboratóriách jadrovej fyziky a fyziky vysokých energií na celom svete. Uplatnenie nachádza aj v oblastiach medicíny a financií. Samotný ROOT je naprogramovaný v jazyku C++ a bol vytvorený v kontexte experimentu, ktorý generoval enormné množstvo dát v CERNe. V ROOTe sú histogramy rozdelené do tried. Jednotlivé triedy histogramov sú znázornené v tabuľke 1.1.

Tabuľka 1.1: Triedy histogramov
(Zdroj: [9])

Triedy histogramov			
Trieda	Rozmer	Dátový typ na kanál	Maximálny obsah binu
TH1C	1D	byte	255
TH1S	1D	short	65 535
TH1I	1D	integer	2147483647
TH1F	1D	float	7 číslic
TH1D	1D	double	14 číslic
TH2C	2D	byte	255
TH2S	2D	short	65 535
TH2I	2D	integer	2147483647
TH2F	2D	float	7 číslic
TH2D	2D	double	14 číslic
TH3C	3D	byte	255
TH3S	3D	short	65 535
TH3I	3D	integer	2147483647
TH3F	3D	float	7 číslic
TH3D	3D	double	14 číslic

Biny histogramu z triedy TH1 sú zobrazované ako 2D objekty, kde pozíciu binu určuje hodnota na osi x a obsah je určený výškou, teda hodnotou na osi y. Pre TH2 je charakteristické, že histogramy sú zobrazené ako 3D objekty. Poloha binov je určená súradnicami na osiach x a y. Tretí rozmer v tomto prípade je výška binov. Histogramy triedy TH3 sú taktiež zobrazované ako 3D objekty s rozdielom, že hodnota obsahu binu nie je prezentovaná výškami binov, hladiny obsahu binu majú priradenú farbu, ktorou sa dá vizuálne zistiť rozsah obsahu pre bin. Pozícia binu v histograme je určená súradnicami na osiach x, y, z.

V prípade histogramov existuje viacero spôsobov zobrazenia. Napriek tomu, že histogram triedy TH2 sa dá zobraziť ako 3D tak existuje aj 2D zobrazenie.



Obr. 1.1: Porovnanie zobrazení TH2 histogramu
(Zdroj: [10])

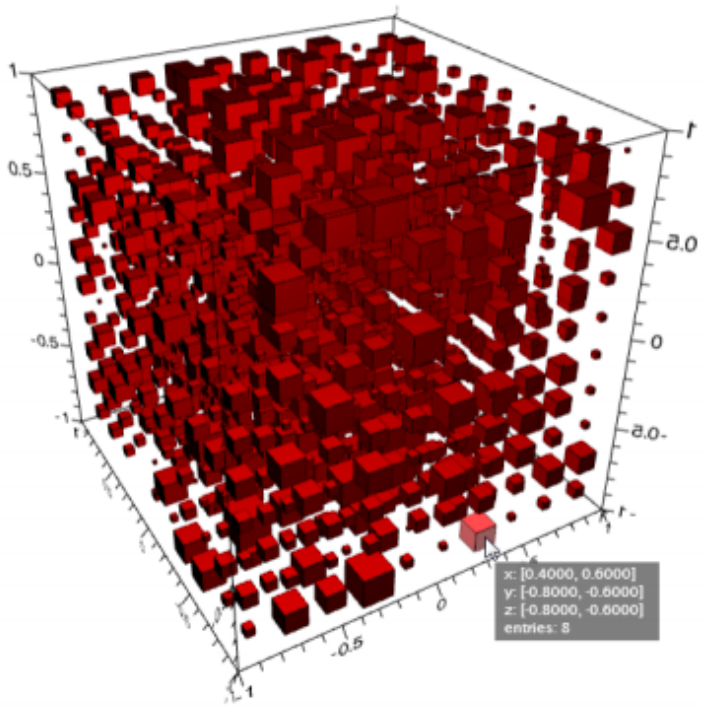
Na obrázku 1.1 je porovnanie zobrazení histogramu z kategórie TH2, ktorý je zobrazený ako trojrozmerný objekt a aj ako dvojrozmerná plocha, kde jednotlivé rozsahy obsahov binov sú charakterizované farbami. Vo všetkých štyroch prípadoch ide o rovnaký histogram. V prvom riadku sú zobrazené 2D verzie histogramu. Obrázok vľavo definuje oblasti binov s rovnakým obsahom a zobrazuje ich ohraničenia, zobrazenie na obrázku vpravo definuje biny a ich obsahy znázornené farbou prislúchajúcou danej hodnote obsahu. V druhom riadku vidíme vľavo histogram vykreslený v 3D priestore. Obsahy binov sú znázornené farbou a rovnako aj výškou stĺpcov. Na poslednom obrázku je znázornený vzťah zobrazení na predchádzajúcich obrázkoch.

1.2.3 Knižnica JSROOT

Softvérový rámec ROOT nám definuje funkcionalitu, ktorá tvorí základ pre vytváranie dátových analýz. Pre využitie na webe je preto tento softvérový rámec bez prípadných rozšírení, ktoré umožnia využívať objekty a funkcie ROOT-u na webe.

Knižnica JSROOT rozširuje funkcionalitu samotného softvérového rámca ROOT. ROOT obsahuje základnú funkcionalitu, ktorá je rozšírená pre využite funkcií na webe. JSROOT poskytuje možnosť čítať dáta z binárnych ROOT súborov a zobrazuje tieto dáta na webe.

Ako uvádza zdroj [11], JSROOT je JavaScript-ová verzia ROOT-u. Knižnica je používaná pre renderovanie grafických objektov histogramov a grafov. Samotné zobrazovanie 3D objektov používa knižnice *D3.js* [12] a *Three.js* [13]. Knižnica *D3.js* je používaná na renderovanie 1D a 2D histogramov a grafov. *Three.js* je používaná pri 2D a 3D zobrazeniach histogramov a grafov. Na obrázku 1.2 je príklad 3D objektu histogramu triedy TH3.



Obr. 1.2: TH3 histogram zobrazený pomocou *Three.js* a *webGL*
(Zdroj: [11])

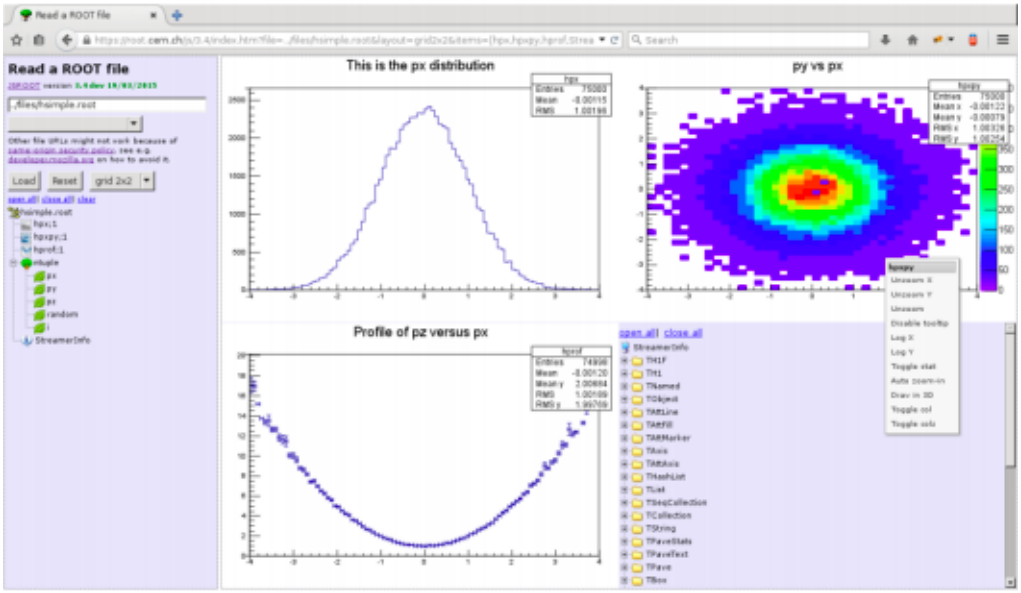
V publikácii [14] je vysvetlené, že knižnica JSROOT je rozdelená do modulov, ktoré sú na sebe nezávislé. Moduly knižnice tvoria:

- **io** modul určený na čítanie binárnych ROOT súborov
- **2d** modul implementuje hlavnú 2D grafiku, založenú na knižnici *d3.js*
- **Jq2d** modul implementuje pokročilú 2D grafiku pomocou jQuery [15] a jQuery-ui.js

- **3d** modul implementujúci funkcie pre zobrazovanie 3D grafiky pre TH2 a TH3 pomocou knižnice *Three.js*
- **gui** generické používateľské rozhranie

Pre možnosť využitia tejto knižnice v klientských aplikáciách je potrebné načítať skript *JSRootCore* na stránke pre správnu špecifikáciu týchto modulov a následné použitie funkcií na webovej stránke. Pre použitie objektov získaných z ROOT súborov na webe je potrebné tieto dáta prezentované v objektoch načítať pred samotným použitím na webe. V JSROOT-e existuje *File* API, ktoré definuje funkcie na otvorenie súborov a následne prečítanie obsahu súboru a to konkrétne metódou *OpenFile*. Funkcia je asynchrónna a príjíma parameter definujúci cestu k binárnemu ROOT súboru a callback funkciu, ktorá sa vykoná po vykonaní vstupno-výstupnej operácie. Ako parameter v callback funkcií je inštancia samotného súboru, z ktorého môžeme asynchrónne prečítať dáta a pracovať s nimi. Pre čítanie dát z inštancie *File* existuje funkcia *ReadObject*. Pre prístup k objektom zo vzdialených serverov knižnica JSROOT obsahuje triedu *XMLHttpRequest*. Na tento účel sa používa funkcia *NewHttpRequest* na vyžiadanie objektu s dátami zo vzdialeného servera. Server zabezpečí prečítanie ROOT súboru a vytvorenie objektu, ktorý poskytne klientskej aplikácií vo forme JSON. Na webe sa spracuje tento objekt v callback funkcii, ktorá obsahuje tento objekt ako parameter a zavolá sa až po prijatí dát zo servera [1].

Na samotné zobrazenie dát z ROOT súborov vieme využiť aj rozhranie na stránke Cernu [8], v ktorom si vieme otvoriť binárny súbor a zobraziť jeho obsah. Pre zobrazenie týchto dát je nutné navštíviť stránku a len na tejto stránke vieme zobraziť tieto dáta priamo zo súboru. Tieto dáta si vieme jednoducho načítať a zobraziť, ale nevieme ich využiť priamo v klientskych aplikáciách ako to potrebujeme my pre riešenie nášho problému. Tento variant je však efektívny pre rýchle zobrazenie dátových analýz a obsahuje jednoduché rozhranie pre interakciu. Samotné rozhranie je zobrazené na obrázku 1.3

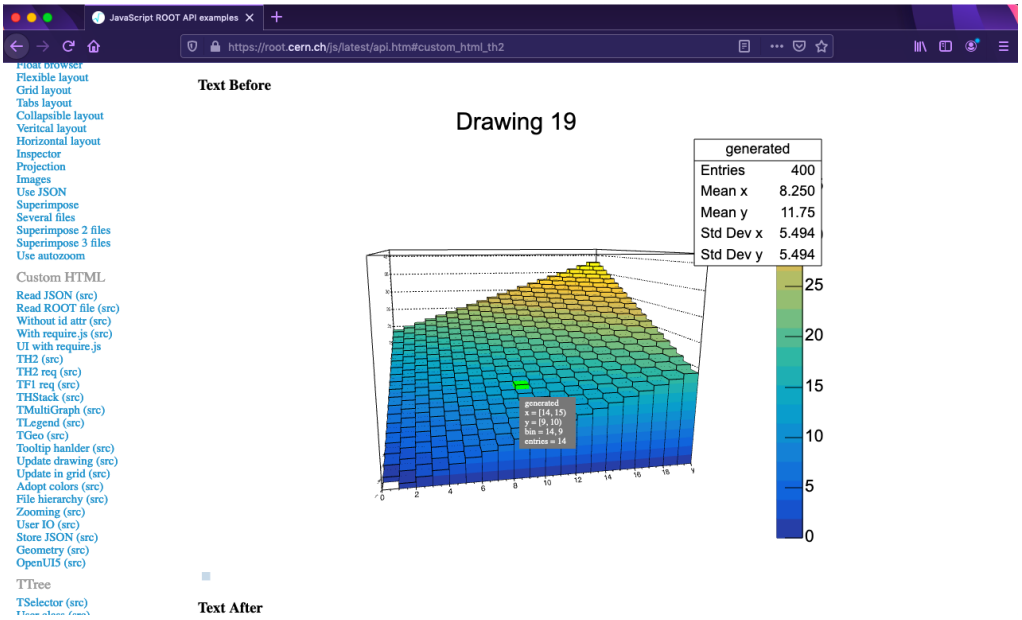


Obr. 1.3: Zobrazenie dát pomocou používateľského rozhrania na stránke (Zdroj: [8])

Rozhranie ponúka na ľavej strane vstup pre zadanie textu, ktorý nám umožňuje vyhľadávať súbory a následne ich zobrazí v tejto sekcii. Zobrazená je aj súborová štruktúra, v ktorej sa vie používateľ orientovať a vyhľadávať súbory. Po kliknutí na súbor sa načítajú príslušné dáta a zobrazia sa na pravej strane obrazovky. Rozhranie obsahuje nespočetné množstvo možností, kde si vieme nastaviť atribúty zobrazenia a vieme aj zobrazovať konkrétne údaje podľa potreby.

1.2.4 Experimentálne overenie použiteľnosti knižnice JSROOT

Knižnica JSROOT obsahuje funkcie, ktoré sa využívajú na zobrazovanie histogramov. Pre zobrazenie 3D histogramov využíva knižnicu *Three.js*. Na *Three.js* sú založené webové softvérové rámce pre virtuálnu realitu a preto sa v tejto časti zameriame na získanie objektu histogramu, ktorý by sme mohli neskôr ďalej skúmať a integrovať ho do prostredia virtuálnej reality. Najskôr sme sa rozhodli zobraziť si príklad 3D histogramu, ako je na obrázku 1.4 a preskúmať tento histogram.



Obr. 1.4: TH2 histogram
(Zdroj: [16])

Histogram je interaktívny. Po nastavení kurzora na bin histogramu sa bin zvýrazní a zobrazuje sa kontextové okno s údajmi o súradniciach binu a obsahom. Obsah predstavuje premenná s názvom obsah v kontextovom okne a je aj kľúčovým údajom, ktorý definuje farbu samotného binu a aj jeho výšku v tomto type histogramu. Používateľ si vie jednoducho polohovať tento histogram otáčaním po všetkých osiach a aj priblížiť jednotlivé biny podľa toho, ako to potrebuje.

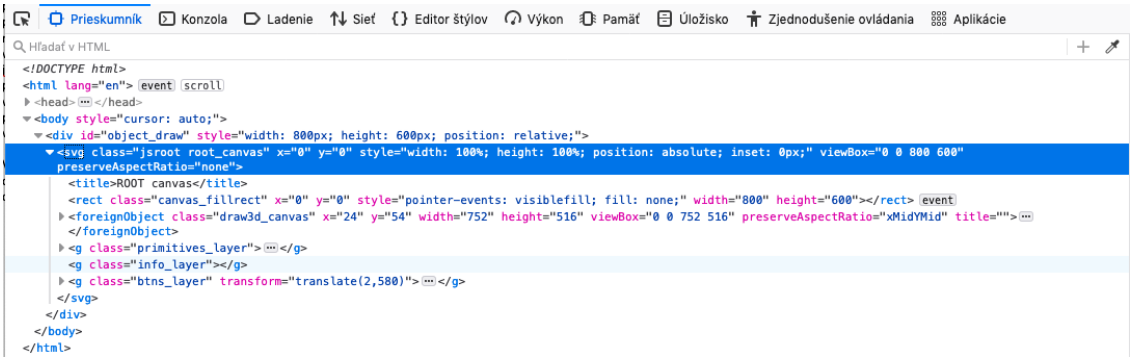
Na stránke s príkladmi histogramov je možné zobraziť zdrojový kód, ktorý sa stará o vygenerovanie histogramu a o jeho vykreslenie na webovú stránku. Pri zobrazení zdrojového kódu pre príklad th2 histogramu na obrázku 1.4 môžme vidieť funkciu 1.1, ktorá volá funkcie volané na objekte JSROOT, ako napríklad *CreateHistogram*, ktorá vytvorí objekt histogramu. Následne sa volajú metódy na objekte tohto histogramu, ako napríklad metóda *getBin*, ktorá vracia referenciu na bin histogramu podľa súradníc binu a funkcia *setBinContent*, ktorá nastaví hodnotu obsahu binu. Keďže sa jedná o príklad tak obsah je vygenerovaný len ilustračne v reálnom svete sa používajú dáta, ktoré sú spracované a uchovávané na úložiskách. Na vykreslenie histogramov je možné použiť metódy *draw* a *redraw*. Rozdiel v ich využití spočíva vo výslednom SVG elemente, kde v prípade použitia funkcie *draw* sa vytvorí nový element a histogram sa vykreslí odznova. Tento prístup je výhodný hlavne pri zobrazení dát, ktoré nie sú citlivé na zmeny. V opačnom prípade funkcia *redraw* zabezpečí len aktualizáciu už vytvoreného elementu do podoby akú nadobúda dátový objekt. Pre vykreslenie príkladu je použitá funkcia *redraw*, ktorá prijíma ako parametre údaje, ako sú, identifikátor elementu v

HTML dokumente, do ktorého je vykreslený histogram, následne samotný objekt histogramu, ktorý je vykresľovaný a ako posledný parameter zobrazujúci typ zobrazenia. Samotná funkcia pristupuje k elementu v DOM na základe identifikátora a neposkytuje návratovú hodnotu, s ktorou by sme vedeli pracovať v našej implementácií.

Zdrojový kód 1.1: funkcia, ktorá vytvára a vykresľuje histogram [17]

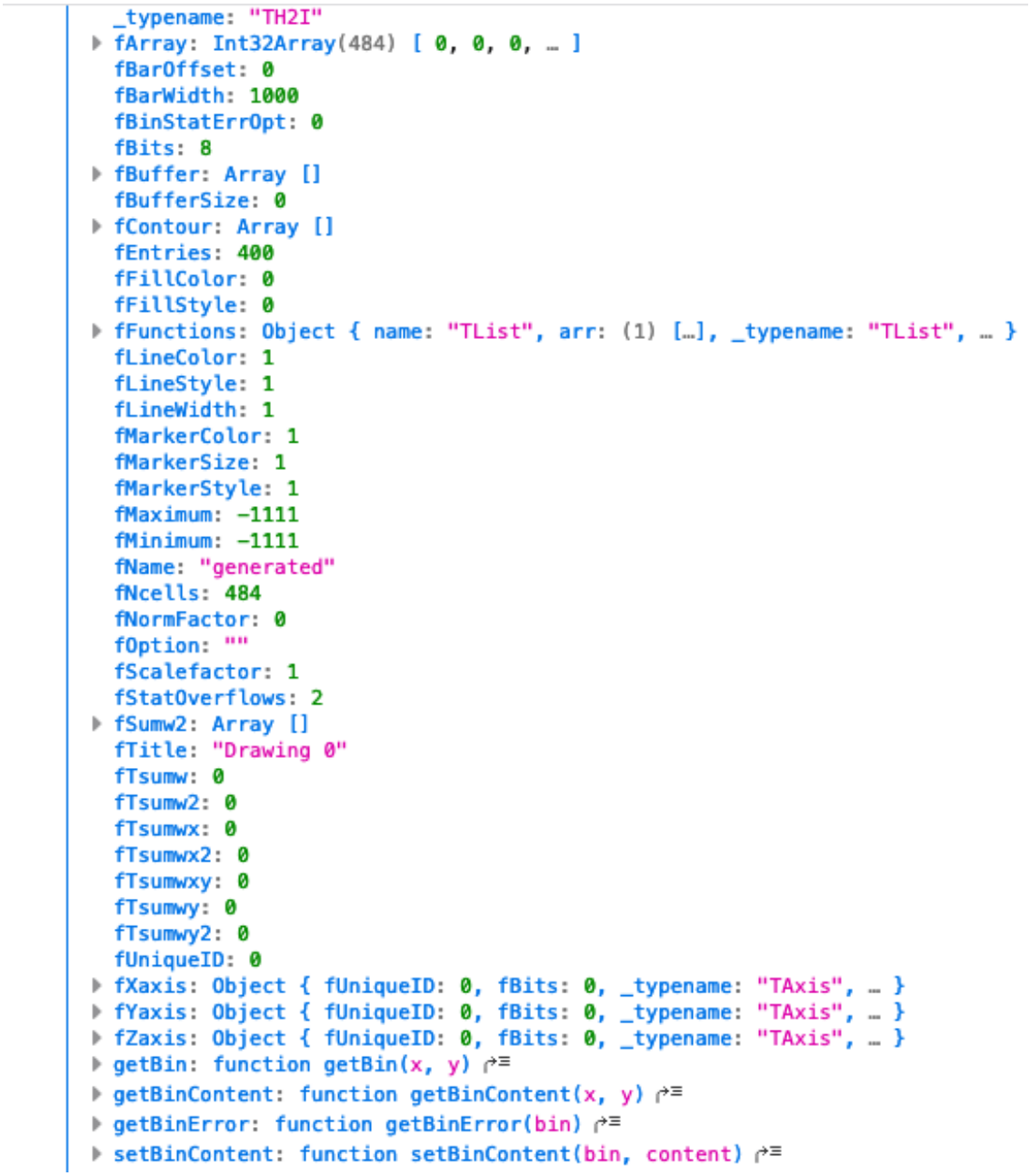
```
function updateGUI() {
   var histo = JSROOT.CreateHistogram("TH2I", 20, 20);
   for (var iy=0;iy<20;iy++)
      for (var ix=0;ix<20;ix++) {
         var bin = histo.getBin(ix+1, iy+1), val = 0;
         switch (cnt % 4) {
            case 1: val = ix + 19 - iy; break;
            case 2: val = 38 - ix - iy; break;
            case 3: val = 19 - ix + iy; break;
            default: val = ix + iy; break;
         }
         histo.setBinContent(bin, val);
      }
   histo.fName = "generated";
   histo.fTitle = "Drawing " + cnt++;

   JSROOT.redraw('object_draw', histo, "colz");
}
```

Po analýze zobrazeného histogramu a jeho zdrojového kódu, sme sa rozhodli použiť samotný zdrojový kód [17] a vytvoriť si jednoduchú testovaciu stránku, ktorá zobrazí tento th2 histogram, aby sme preskúmali aj HTML dokument a zistili formát objektov, ktoré budeme potrebovať neskôr integrovať do virtuálnej reality. Po otvorení HTML dokumentu vo webovom prehliadači a použitím nástrojov pre vývojárov sme si zobrazili štruktúru HTML dokumentu, zamerali sme sa hlavne na element s identifikátorom `"object_draw"` do ktorého funkcia vykresľuje objekt histogramu. Na obrázku 1.5 je znázornená štruktúra elementov, ktoré vygenerovala funkcia *redraw*. Funkcia *redraw* vytvorí SVG element.



Obr. 1.5: Element s vykresleným histogramom v DOM

Na obrázku 1.6 je zobrazený objekt histogramu vytvorený funkciou *CreateHistogram*. Objekt obsahuje atribúty, ktoré definujú rôzne vlastnosti histogramu, avšak nás zaujímajú atribúty, ktoré by sme vedeli využiť pri integrácií histogramu do virtuálnej reality. Samotný objekt obsahuje pole s binmi rovnako ako aj funkcie *getBin* a *getBinContent*, ktorými dokážeme získať informácie o presnej polohe a obsahu binu. Nenašli sme žiadny atribút, ktorý by obsahoval 3D objekty alebo geometriu pre zobrazenie pomocou nejakej knižnice, alebo softvérového rámca zobrazujúceho 3D scénu.

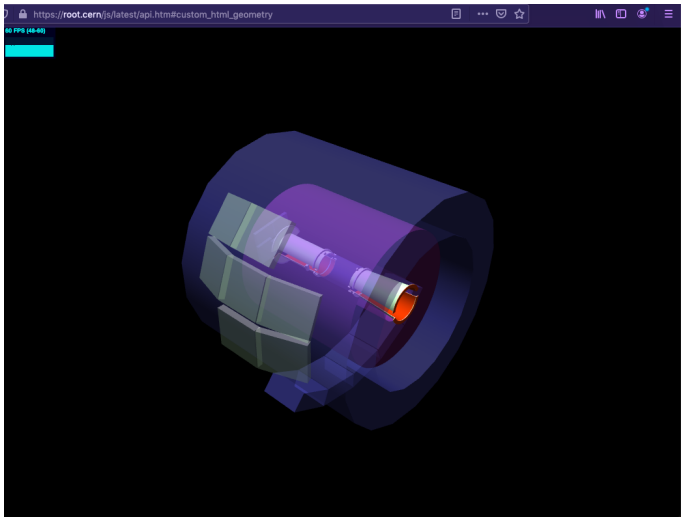


Obr. 1.6: Histo objekt zobrazený v konzole prehliadača

Keďže je knižnica JSROOT globálne využívaná viacerými výskumnými inštitúciami pracujúcimi s obrovským množstvom dát je pre nás nemožné, aby sme mohli upravovať funkcie tejto knižnice. Funkcia *draw* alebo *redraw*, ktorá sa stará o vytvorenie finálneho SVG elementu v HTML DOM je preto nepostačujúca. Funkcie neposkytujú žiadnu návratovú hodnotu a starajú sa o vytvorenie SVG v HTML DOM. Funkcia vo svojom tele vytvára 3D objekt histogramu, ktorý následne zobrazí pomocou *WebGL.js* a *Three.js* v scéne. Samotný SVG element v HTML štruktúre môžeme považovať za scénu, v ktorej sú zobrazené všetky objekty, v tomto prípade histogram. Výsledný SVG element je znázornený na obrázku 1.2. Z tohto

experimentu sme zistili, že funkcie *draw* a *redraw* ako definuje aj ich názov len vykreslia scénu aj s objektom, ktorý sa má zobraziť a nevieme získať tento 3D objekt, s ktorým by sme vedeli pracovať v javascripte. Rovnako predpokladáme, že samotná virtuálna realita už bude obsahovať scénu a preto ak chceme vyriešiť tento problém budeme potrebovať získať len samotný 3D objekt histogramu, ktorý v tomto experimente bol zakomponovaný v scéne. Vieme získať objekt histogramu, ktorý obsahuje údaje o samotnom histograme a všetkých binoch. Pomocou tohto objektu by sa však mohli dať vygenerovať všetky biny manuálne ako objekty vo virtuálnej realite s príslušnou pozíciou v scéne. Toto riešenie považujeme ako posledné možné riešenie, keďže sa jedná o vytvorenie celého histogramu a jeho geometrie. Ako optimálnejšie riešenia sú riešenia v prípade získania už hotovej geometrie histogramu, ktoré sa budeme snažiť integrovať do virtuálnej reality.

JSROOT obsahuje *TGeo* API, ktoré sa využíva taktiež na vytváranie objektov obsahujúcich geometriu. Ako uvádza dokumentácia [1] veľakrát potrebujeme pracovať so samotným objektom a upravovať alebo modifikovať ho v neskorších fázach, využitím JavaScriptu. Z toho dôvodu nám nepostačuje funkcia *draw*, ktorá nám vytvára už priamo element, ktorý je vygenerovaný v HTML DOM a tento element už rovno definuje vzhľad celej scény aj so zobrazenými všetkými objektami. V prípade *TGeo* sa používa funkcia build na objekte GEO, ktorá na základe objektu a jeho nastavení, ktoré funkcia prijíma, ako parametre vráti *Three.js* 3D model. Môžeme ho chápať, ako model existujúci v scéne a je možné s ním pracovať separátne, čo v prípade SVG elementu vykresleného funkciou draw nie je možné. Ako ďalej uvádza dokumentácia, aj na tento 3D model je možné zavolať funkciu *draw* a vykresliť ho na scénu ako, aj v prípade objektu definujúceho histogram. Funkcia build prijíma parametre, ako napríklad parameter *obj*, na základe údajov v *obj* sa generuje*Three.js* 3D model a parameter *opt*, ktorý definuje nastavenia pre samotný objekt. Na obrázku je zobrazená scéna, na ktorej je zobrazený 3D model urýchľovača častíc. Urýchľovač častíc je vytvorený funkciou *build*. V tomto príklade [18] samotné vykreslenie scény nie je realizované funkciou *draw* alebo *redraw*, ale renderovaním scény použitím *Three.js* knižnice. Tento princíp však budeme rozoberať podrobne pri analýze A-Frame v nasledujúcej kapitole.



Obr. 1.7: *Three.js* model vytvorený funkciou *build*

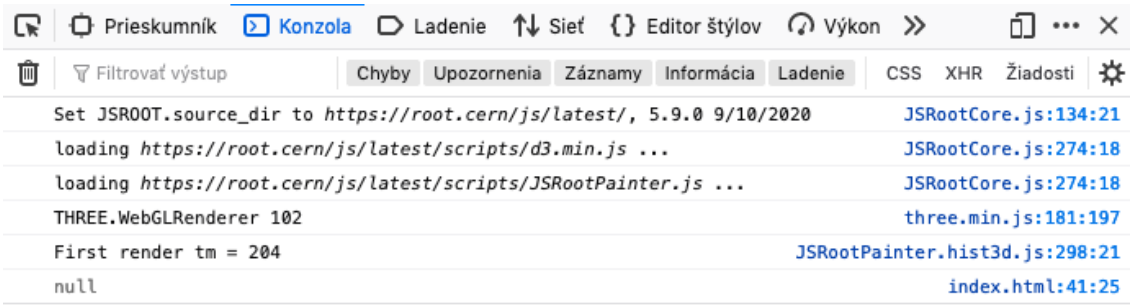
Pri skúmaní TGeo API sme nenašli žiadne príklady zobrazovania interaktívnych histogramov. Vzhľadom na túto skutočnosť sme sa rozhodli vykonať ďalší experiment, ktorého cieľom bude overiť vytvorenie separátneho 3D *Three.js* modelu pomocou JSROOT TGeo. Nebudeme využívať API na získanie objektu zo servera. Pre potreby nášho experimentu máme vygenerovaný názorný TH2I histogram, ktorý sme použili v predchádzajúcom experimente 1.1. Tento objekt histogramu sa pokúsime poskytnúť ako parameter obj a následne zavolať funkciu *build* na objekte GEO. Pre overenie budeme sledovať konzolový výpis vytvoreného 3D modelu. V našom kóde sme doplnili do URL adresy pre načítanie skriptu *jsroot.core* parameter *geom*, ktorý nám zabezpečí načítanie potrebnej funkcionality pre používanie objektu GEO a rozšírili kód o fragment 1.2, ktorý vytvorí 3D model a následne ho vypíše do konzoly. Aby sme zabezpečili korektné vykonanie fragmentu kódu až po načítaní potrebných skriptov, použili sme funkciu *setTimeout* a nastavili oneskorenie vykonávania na 2 sekundy. Funkcie sa tak vykonajú v momente kedy už budú načítané všetky potrebné skripty.

Zdrojový kód 1.2: Rozšírenie funkcie 1.1 pre vytvorenie 3D objektu z TH2I histogramu

```
// function demonstrate creation of three.js model
setTimeout(() => {
      let opt = { numfaces: 100000, numnodes: 1000 };
      let obj3d = JSROOT.GEO.build(histo, opt);
      console.log(obj3d)
}, 2000)
```

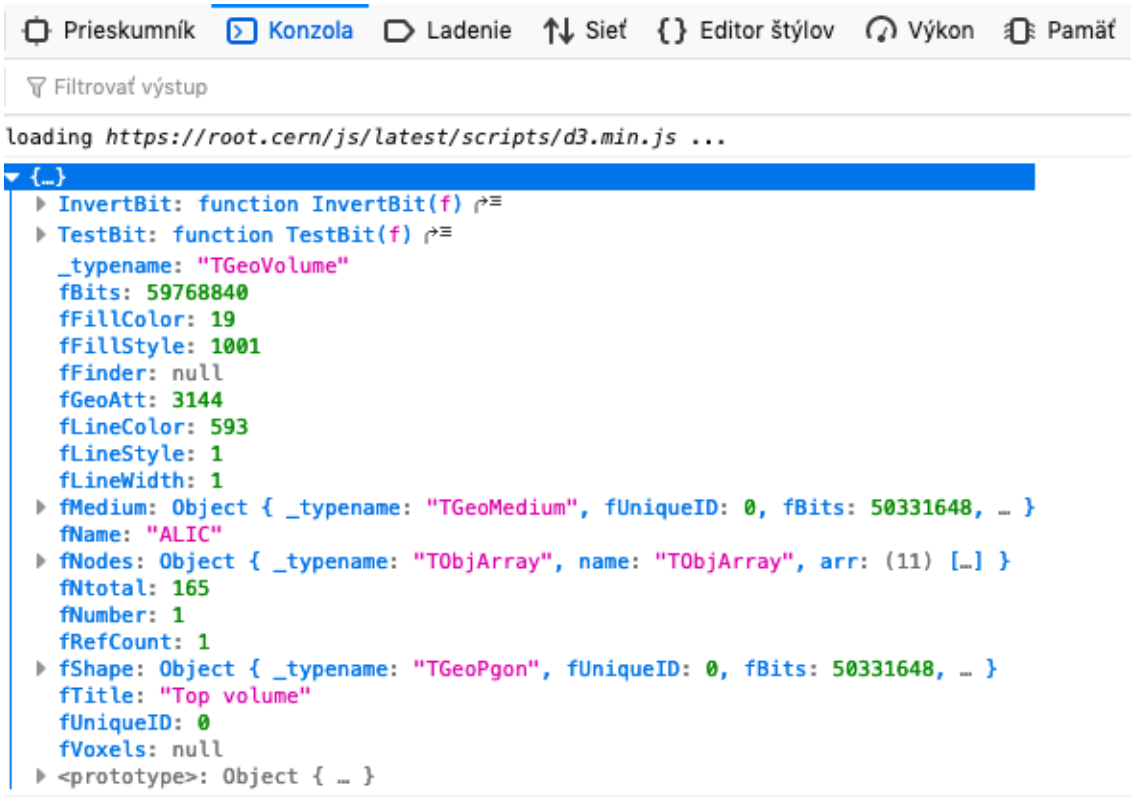
Výpis v konzole zobrazený aj na obrázku 1.8 nám odhalil hodnotu **null** v premen-

nej *obj3d*, ktorá mala obsahovať inštanciu 3D objektu histogramu. Z toho vyplýva, že samotná funkcia nevytvorila objekt, ktorý sme očakávali a ktorý by sme mohli neskôr skúšať integrovať do virtuálnej reality.



Obr. 1.8: Konzolový výpis po spustení našej testovacej stránky

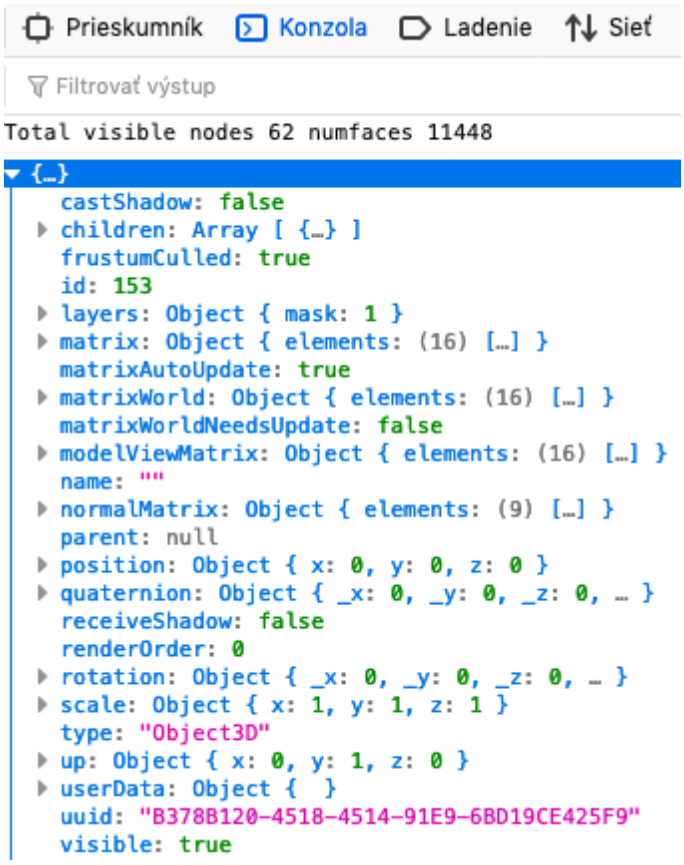
Pre lepšie pochopenie príčiny sme si skúsili zobraziť v konzole objekty, ktoré poskytujeme ako parametre obj do funkcie build a následne ich porovnať a analyzovať. Na tento test sme si opäť upravili zdrojový kód našej testovacej stránky, aby po spustení vypísal objekt histogramu do konzoly. Rovnako sme použili zdrojový kód príkladu stránky so zobrazeným urýchľovačom častíc [18], ktorý sme upravili, aby taktiež vypísal objekt do konzoly. Oba tieto vypísané objekty v konzole sme porovnali a snažili sme sa zistiť aké možnosti máme pri úprave objektu histogramu, aby sme získali jeho geometriu pre ďalšie použitie.



Obr. 1.9: Konzolový výpis objektu, ktorý bol poskytnutý ako parameter *obj* do funkcie *build*. Objekt obsahuje popis geometrie urýchľovača častíc ALICE

Pri porovnaní objektu histogramu na predchádzajúcom obrázku 1.6 a objektu urýchľovača častíc je zrejme, že objekt urýchľovača obsahuje atribúty, ktoré definujú samotný estetický výzor objektu. Urýchľovač obsahuje atribúty ako farby výplne, štýl výplne, geometriu, farby čiar, šírku čiar, štýl čiar, samotný tvar objektu a polia, ktoré obsahujú ďalšie objekty a tie tvoria výsledný objekt a ešte mnoho ďalších atribútov. Atribúty ako tvar objektu, pole s ďalšími objektami a ďalšie. Neobsahuje objekt histogramu, ktorý je zobrazený na obr. 1.9. Objekt histogramu obsahuje údaje o binoch ako súradnice, obsah, názov a mnohé ďalšie, avšak neobsahuje geometrické údaje ako napríklad tvar objektu na základe, ktorých sa dá vytvoriť *Three.js* 3D model histogramu. Môžme tak skonštatovať, že objekty na obrázkoch 1.6 a 1.9 nie sú identické a funkciou *build* nezískame 3D model histogramu.

Jedným z našich cieľov je aj zobrazenie 3D projekcií vo virtuálnej realite. Keďže sme sa dozvedeli, že samotná funkcia *build* volaná na objekte GEO vytvára na základe údajov o objektoch, ktoré sú uložené v binárnych súboroch na serveroch CERN-u 3D objekty použiteľné v *Three.js* scéne a preto sme si zobrazili aj samotný objekt urýchľovača 1.10, ktorý nám vytvorila funkcia *build*.



Obr. 1.10: Konzolový výpis *Three.js* objektu typu *Object3D*, ktorý vytvorila funkcia *build*

Projekcie môžeme považovať za 3D objekty, ktoré neobsahujú žiadnu interakciu. Ich primárny účel bude hlavne zobrazenie objektu vo virtuálnej realite v prípade prezentácie a vizualizácie urýchľovačov a detektorov, ktoré používa CERN vo svojich výskumoch.

1.2.5 Zhrnutie výsledkov analýzy knižnice JSROOT

Pri analýze knižnice JSROOT a experimentoch, ktorými sme sa snažili otestovať potenciálne možnosti zobrazení dát a funkcií tejto knižnice sme sa rozhodli zhrnúť všetky fakty a poznatky, ktoré sme získali počas experimentovania a skúmania. Nasledujúce body obsahujú kľúčové poznatky, získané v analýze JSROOT-u.

- Načítanie knižnice JSROOT vytvorí globálny objekt JSROOT, na ktorom následne vieme volať funkcie. Dôležitý fakt je rozdelenie JSROOT na moduly pre samotnú rozsiahlosť knižnice. Je dôležité zadefinovať potrebné URL parametre, aby sa načítali všetky potrebné skripty.
- Na načítanie objektu, s ktorým budeme neskôr pracovať vieme použiť File

API a konkrétne funkciu na otvorenie binárneho súboru *OpenFile* a na inštancií *File* po otvorení súboru funkciou *ReadObject*.

- Pre prístup k objektu zo vzdialeného servera, vieme použiť funkciu *NewHttpRequest* a získať dáta vo forme JSON stringu.
- Pre vytváranie histogramov tried TH máme k dispozícií JSROOT funkciu *CreateHistogram*.
- Funkcie *draw* a *redraw* vykresľujú *Three.js* scénu s objektami do html dokumentu. Zo samotných funkcií však nevieme získať objekt s geometriou, s ktorým by sme vedeli pracovať v nasledujúcej fáze.
- Máme k dispozícií objekt histogramu, ktorý obsahuje informácie o binoch.
- JSROOT obsahuje aj TGeo API, ktoré definuje objekt GEO a funkcia *build* volaná na tomto objekte vytvorí 3D *Three.js* model typu *Object3D*. Samotný objekt, z ktorého sa vytvára 3D objekt musí obsahovať popis tvaru a geometriu.
- Z objektu histogramu sme nedokázali získať *Object3D* funkciou *build* z dôvodu chýbajúcich atribútov dôležitých pre vytvorenie objektu. V nasledujúcich fázach budeme musieť vytvoriť geometriu histogramu na základe údajov o binoch.
- Pre vizualizáciu projekcií máme k dispozicií Object3D model, ktorý má geometriu a budeme sa snažiť integrovať hotový objekt do virtuálnej reality.

1.3 Integrácia údajov do virtuálnej reality

Pre samotné riešenie tohto problému potrebujeme najskôr pochopiť rozdiel medzi prezentáciou údajov v dvojrozmernom priestore a zobrazením v trojrozmernom priestore, aby sme sa vedeli vo fáze analýzy zamerať na dôležité technológie, ktoré nám pomôžu využiť práve možnosti, ktoré nám virtuálna realita ponúka naviac oproti samotným možnostiam, ktoré sú ponúkané aplikáciami založenými na dvojrozmernom zobrazení. V článku [19], ktorý sa venuje problematike zobrazovania obrovského množstva dát vo virtuálnej realite je zdôraznené, že hlavná výhoda používania virtuálnej reality na zobrazovanie dát je prezentácia 3D dát a umiestnenie používateľa v 3D simulácií, ktorá umožňuje rýchlo a efektívne spracovávať informácie a uchovávať ich dlhšie. V prípade zobrazenia dát v 3D pries-

tore môžeme tiež jednoducho chápať rôzne dimenzie údajov pomocou farieb, tvarov, priehľadností, animácií a ďalších.

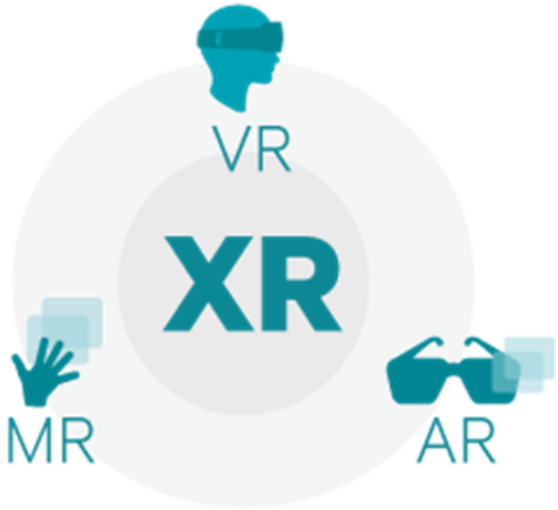
Podľa Steva Brysona [20] rozhrania virtuálnej reality umožňujú rýchle a intuitívne skúmanie údajových zväzkov a taktiež umožňujú aj preskúmanie javov na rôznych miestach zväzku. Užívateľ získava kontrolu nad vizualizačným prostredím prostredníctvom rozhrania, ktoré je integrované v samotnom prostredí, v ktorom je aj samotný užívateľ.

V našom prípade sa budeme snažiť využiť, čo najefektívnejšie vlastnosť virtuálnej reality, ktorá nám umožní umiestniť samotného používateľa priamo do prostredia, v ktorom budú zobrazené aj samotné údaje. Následne sa zameriame na vytvorenie jednoduchého a prirodzeného rozhrania pre interakciu užívateľa s dátami v prostredí tak, aby sa používanie podobalo, čo najviac interakciám v reálnom svete.

1.3.1 Rozšírená realita

Skôr než začneme skúmať samotné technológie využívajúce virtuálnu realitu si objasníme pojmy, ako virtuálna realita VR, rozšírená realita AR a zmiešaná realita MR. V publikácií [21] je práve VR a AR zaradzovaná do rovnakej kategórie napriek tomu, že samotné technológie sú odlišné. Pri snahe objasniť, čo najzrozumiteľnejšie tieto pojmy sme narazili na viaceré definície. Wilson [22] zadefinoval AR ako formu VR, kým Drascic a Milgram [23] definujú AR a VR ako protiklady, kde AR definuje prostredie reálneho sveta voči samotnej virtuálnej realite, ktorá bola popísaná ako dokonalý protiklad AR, teda abstraktné prostredie, ktoré nemá nič spoločné s reálnym svetom. Presnejšie definície AR a VR sme našli v knihe [24] v ktorej je VR definovaná ako simulácia reálneho sveta tvorená 3D priestorom, časom a pohlcujúcim rozhraním pre užívateľa. AR je nová forma interakcie používateľa, ktorá zobrazuje informácie priamo v prostredí reálneho sveta. Ako prostredie sa sníma reálne prostredie a nenahrádza sa abstraktným prostredím ako v prípade VR. V AR človek zapája všetky svoje zmysly, čo v prípade VR nie je úplne možné, keďže človek stráca kontakt s reálnym svetom. XR a AR je častokrát prekladaná ako zmiešaná realita, čo môže spôsobovať zmätky, a preto v tejto práci bude pojem zmiešaná realita predstavovať XR.

Zmiešaná realita MR zlučuje fyzickú realitu s virtuálnou realitou s rozdielom, že medzi týmito realitami existuje interakcia v reálnom čase. Fyzické objekty v reálnej scéne môžu interagovať s abstraktnými a pri manipulácii používateľa s fyzickým objektom to má vplyv aj na abstraktné objekty, ktoré sú v interakcií s týmto objektom [25].



Obr. 1.11: Súvislosti pojmov
(Zdroj: [25])

Spomínané typy realít však môžme zaradiť do všeobecnejšej množiny, ktorá sa nazýva rozšírená realita XR ako je zobrazené na obrázku 1.11.

1.3.2 Webový rámec pre virtuálnu realitu A-Frame

Pre vyriešenie nášho problému potrebujeme zvoliť technológiu, ktorá nám umožní zobraziť dáta vo VR na webe. Pre jednoduchú použiteľnosť a podporu pre rozličné zariadenia sme sa rozhodli pre softvérový rámec A-Frame.

Keďže sa v tejto časti analýzy budeme venovať tomuto softvérovému rámcu ,budeme preto čerpať informácie hlavne z dokumentácie [2]. Ako je definované v úvode dokumentácie, A-Frame je webový softvérový rámec pre vytváranie VR zážitkov. A-Frame rozširuje HTML, čo umožňuje jednoduchú prácu s týmto softvérovým rámcom.

Aj napriek podobnosti s HTML, A-Frame nie je len značkovací jazyk, ale obsahuje jadro založené na Entity-Component-System, ktorá poskytuje deklaratívnu a rozšíriteľnú štruktúru knižnice *Three.js* [26]. Samotný A-Frame má jednu z najväčších VR komunít a obsahuje rozšírenia a podporu pre VR headsety ako Vive, Rift, Windows Mixed Reality, Oculus go, Daydream, GearVR a rovnako aj pre rozšírenú realitu AR.

1.3.3 Entity-Component-Systém architektúra A-Frame

ECS architektúru môžeme definovať ako návrhový vzor, ktorý uprednostňuje princípy kompozície pred dedením a hierarchiou. Medzi benefity takéhoto prístupu patrí hlavne rozdelenie funkcionalít do menších znovu použiteľných celkov známych ako komponenty. Taktiež redukuje komplexnosti spôsobované dlhým zreťazením dedičností a prispieva k lepšej flexibilite pri definovaní objektov spájaním viacerých znovu použiteľných častí. Taktiež dovoľuje rozširovanie nových funkcionalít a ich zdieľanie ako komunitné komponenty. Samotná ECS architektúra zahŕňa:

- **entity** - entita je základný element, bez ktorého by nebolo možné renderovať objekty. Element <*a-entity*> reprezentuje kontajner, ktorý obsahuje objekty ku ktorým vieme pristupovať, meniť ich atribúty a pridávať funkcionalitu pomocou komponentov.
- **komponenty** - komponenty sú znovu použiteľné moduly alebo dátové kontajnery umožňujúce pridávať entitám rôzne dáta a funkcionalitu. Základná charakteristika komponentov v A-Frame je ich univerzálnosť a znovu použiteľnosť pre rôzne entity. Komponenty vieme definovať a vytvárať API volaním *AFRAME.registerComponent* (*name*, *definition*)
- **systém** - systém môžeme použiť na oddelenie logiky od dát. Systém sa zvyčajne zvykne starať o logiku ,zatiaľ kým komponenty plnia funkciu dátových kontajnerov. Skrátene by sme mohli definovať systém, ktorý poskytuje manažment a servis pre triedy komponentov. Systémy sú pridávané do scény. Systémy sú vkladané do elementu <a-scene> ako atribúty a vytvárané API volaním *AFRAME.registerSystem* (*name*, *definition*)

```HTML
<a-entity geometry="primitive: sphere; radius: 1.5"
          light="type: point; color: white; intensity: 2"
          material="color: white; shader: flat; src: glow.jpg"
          position="0 0 -5"></a-entity>
```

Obr. 1.12: Na obrázku je zobrazený príklad definície A-Frame entity a komponentov
(Zdroj: [2])

Na obrázku 1.12 je zobrazený príklad definície A-Frame entity, ktorá obsahuje formou HTML atribútov definované komponenty, ktoré pridávajú objektom špe-

cifickú funkcionalitu. Oranžová farba na tomto obrázku symbolizuje názvy komponentov, zelená definíciu a atribúty týchto komponentov. Ako príklad vieme uviesť komponent **geometry**, ktorý definuje geometriu 3D objektu v scéne alebo komponenty ako **material** a **position**, ktoré sa starajú o vizuálne vlastnosti a pozíciu objektu v scéne.

Všetky tieto komponenty vieme využívať neobmedzene a úpravou parametrov vieme meniť tieto vlastnosti 3D objektov aj v iných entitách. Vieme ich používať nezávisle.

1.3.4 Vytváranie vlastných komponentov v A-Frame

Softvérový rámec obsahuje už zaregistrované komponenty ako komponenty definujúce funkcionality ako nastavenie tvaru 3D objektu, pozície, veľkosti, materiálu a mnohé ďalšie, ktoré sú pripravené priamo na použitie. Častokrát potrebujeme objektom pridať funkcionality alebo dáta, ktoré sú zriedkavé a nie sú implementované v jadre softvérového rámca. Pre tento účel vieme samotné komponenty vytvárať a rozširovať tak vlastnosti a funkcionality objektov v scéne.

A-Frame komponent môžeme definovať ako JavaScript objekt zložený z atribútov a metód. V samotných komponentoch máme možnosť používať *javascript* a rovnako aj vieme pristupovať priamo k 3D objektom na úrovni *Three.js* a webAPIs.

```js
AFRAME.registerComponent('foo', {
  schema: {
    bar: {type: 'number'},
    baz: {type: 'string'}
  },

  init: function () {
    // Do something when component first attached.
  },

  update: function () {
    // Do something when component's data is updated.
  },

  remove: function () {
    // Do something the component or its entity is detached.
  },

  tick: function (time, timeDelta) {
    // Do something on every scene tick or frame.
  }
});
```

Obr. 1.13: Príklad vytvárania vlastného komponentu v A-Frame
(Zdroj: [2])

Štruktúra komponentu 1.13 pozostáva zo schémy, v ktorej sú atribúty komponentu a funkcií, ktoré sú vykonávané v rôznych fázach životného cyklu komponentu.

Životný cyklus komponentu pozostáva z funkcií:

- **init**() funkcia sa zavolá po inicializácií objektu.
- **update**() funkcia sa zavolá pri zmene vlastností objektu ale aj pri inicializácií komponentu.
- **remove**() funkcia sa zavolá pri odstránení objektu zo scény.
- **tick**(**time, timeDelta**) funkcia sa zavolá pri každej zmene alebo pri tick volaní scény.
- **tock**(**time, timeDelta**) funkcia sa zavolá až po vykreslení scény.
- **play**() funkcia sa okrem inicializácie komponentu volá hlavne keď scéna alebo entita spustí pridanie nejakého správania.

- **pause**() funkcia sa volá pri odobratí komponentu zo scény ale aj v prípadoch, keď scéna alebo entita pozastaví správanie.
- **updateSchema**() metóda sa volá pri zmene vlastnosti komponentu. Tieto vlastnosti sú definované v schéme komponentu a slúži na dynamickú zmenu týchto vlastností.

Na obrázku 1.14 je ukážka vloženia vytvoreného komponentu *foo* do entity spolu aj s parametrami, ktoré budú uložené v schéme komponentu.

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

Obr. 1.14: Vloženie komponentu do entity
(Zdroj: [2])

V niektorých prípadoch je potrebné priradiť entite viac inštancií jedného komponentu. Možnosť definovania viacerých inštancií pri definovaní komponentu v A-Frame je predvolene vypnutá a preto je potrebné toto nastavenie upraviť pri registrácií komponentu nastavením premennej *multiple* na hodnotu **True**. Pri pridávaní inštancií jedného komponentu sú tieto inštancie odlíšené pomocou identifikátora ID. V metódach komponentu vieme rozlíšiť komponenty prístupom k premennej *this.id*.

```HTML
<a-scene>
  <a-entity
    sound="src: url(sound.mp3)"
    sound__beep="src: url(beep.mp3)"
    sound__boop="src: url(boop.mp3)"
  ></a-entity>
</a-scene>
```

Obr. 1.15: Názorná ukážka definície viacerých inštancií komponentu, v tomto prípade ide o komponent sound. Názov komponentu je od ID oddelený dvomi podtržníkmi
(Zdroj: [2])

Pri práci v A-Frame je podľa dokumentácie [2] odporúčaný prístup písania kódu práve do týchto komponentov, neodporúča sa vytvárať moduly alebo súbory a vkladať ich do elementov **script** v HTML ako je to bežne aplikované a používané pri vyvoji 2D webových aplikácií. Najideálnejší prístup je si premyslieť funkcionalitu a vhodné ju rozdeliť do viacerých komponentov.

1.3.5 Vytváranie vlastných primitív v A-Frame

V A-Frame sa používajú elementy nazývané entity. Entita v A-Frame definuje všeobecný element ku ktorému je možné pripájať komponenty, ktoré pridávajú entitám rozličné vlastnosti a funkcionalitu. A-Frame však obsahuje aj špeciálne entity, ktoré už obsahujú komponenty, a tak tie entity majú už predvolenú funkcionalitu a vlastnosti. Táto entita je známa ako primitívum. Ako primitívum v A-Frame môžeme považovať napríklad box, circle, cursor, plane, camera a mnoho ďalších. Pre vytvorenie primitíva je potrebné zadefinovať sémantické meno (napríklad <a-circle>), množinu komponentov aj s potrebnými hodnotami a správne namapovať dáta komponentov na HTML atribúty. Primitíva sa definujú hlavne pri častom využívaní entity so špecifickými vlastnosťami za účelom pohodlnejšieho používania entít v HTML DOM.

Primitívum vieme zaregistrovať API volaním *registerPrimitive*. Ako parameter je potrebné nastaviť názov primitíva a samotný názov musí obsahovať pomlčku. Parameter **definitions** reprezentuje *JavaScript-tový* objekt, ktorý definuje dôležité vlastnosti primitíva **defaultComponents** na definovanie množiny komponentov a **mappings** na vhodné mapovanie dát komponentov na HTML atribúty.

```js
AFRAME.registerPrimitive('a-ocean', {
  // Attaches the `ocean` component by default.
  // Defaults the ocean to be parallel to the ground.
  defaultComponents: {
    ocean: {},
    rotation: {x: -90, y: 0, z: 0}
  },

  // Maps HTML attributes to the `ocean` component's properties.
  mappings: {
    width: 'ocean.width',
    depth: 'ocean.depth',
    density: 'ocean.density',
    color: 'ocean.color',
    opacity: 'ocean.opacity'
  }
});
```

Obr. 1.16: Príklad vytvárania vlastného primitíva v A-Frame
(Zdroj: [2])

1.3.6 Interaktivita v A-Frame a ovládače

V našom prípade okrem samotného zobrazenia údajov potrebujeme zabezpečiť aj samotnú interaktivitu zobrazeného histogramu. A-Frame umožňuje interakcie s objektami viacerými spôsobmi. Na úrovni JavaScriptu je vyriešená interaktivita funkciou *addEventListener*, ktorá sa pripína na objekty v scéne. Pridávanie listenerov je možné realizovať získaním referencie na objekt v HTML DOM napríklad funkciami *querySelector* a *getElementById*, ako sú aj využívané vo vývoji webových 2D aplikácií. V A-Frame však máme aj iný spôsob definovania listenerov, a to priamo v implementácií komponentu. Ako bolo spomenuté v predošlej podkapitole tak v komponente vieme pristupovať priamo k objektu v scéne a rovnako vieme aj pripnúť na tento objekt listener na udalosti.

Hovoríme o virtuálnej realite a výsledné stránky budú zobrazované nielen na počítačoch, ale najmä použitím headsetov a iného hardvéru umožňujúceho zobrazenie virtuálneho prostredia. Tomu prislúcha široká škála ovládačov a udalostí, ktorými by sme mohli vykonávať interakcie s objektami. Bežné udalosti, ktoré poskytuje webový prehliadač určite nebudú stačiť. A-Frame má podporu pre tieto zariadenia a ovládače, ktoré umožnia rozpoznávať oveľa väčšie spektrum udalostí.

A-Frame obsahuje pomerne veľa ovládačov pre interakciu na rozličných zariadeniach. My sa zameriame na ovládače umožňujúce interakcie na stolných počítačoch a headsetoch Oculus.

- **wasd controls** - ovládanie pohybu pomocou kláves wasd.
- **look-controls** - v prostredí virtuálnej reality pohybovaním sa.
- **oculus-go-controls** - ovládač pre zariadenie Oculus-go.
- **oculus-touch-controls** - ovládač pre zariadenie Oculus.

Spomenuté A-Frame komponenty po pridaní do entít umožnia následne rozpoznávanie a zachytávanie udalostí, ktoré prislúchajú špecifickým zariadeniam a ovládačom.

1.3.7 Možnosti vývoja s Three.js v A-Frame

A-Frame je softvérový rámec založený na knižnici *Three.js*. V A-Frame je jednotlivá entita zložená z množiny objektov. Poskytuje možnosť pristupovať plnohodnotne k scéne a objektom na úrovni knižnice *Three.js*. Tento princíp je efektívny

a prospešný z hľadiska možnosti využitia objektov vytvorených pomocou funkcií knižnice *Three.js* a ich vkladania do scény respektíve do entít v A-Frame bez akejkoľvek potreby úpravy objektov alebo konverzie. Scéna vytvorená v A-Frame bude rozširovať scénu vytvorenú knižnicou *Three.js*. Na obrázku 1.17 je zobrazený súvis jednotlivých prístupov pri tvorení scény pomocou A-Frame a *Three.js*.

```
<a-scene>
  <a-box>
    <a-sphere></a-sphere>
    <a-light></a-light>
  </a-box>
</a-scene>
```

```
THREE.Scene
  THREE.Mesh
    THREE.Mesh
    THREE.Light
```

Obr. 1.17: Príklad na ľavej strane demonštruje prístup vytvorenia scény použitím A-Frame. Na pravej strane je zobrazený prístup pomocou knižnice Three.js. V oboch prípadoch je výsledná scéna rovnaká.
(Zdroj: [2])

Výhoda tohto prístupu je hlavne možnosť pristupovať k objektom v entitách a modifikácia ich atribútov priamo pomocou funkcií Three.js rovnako aj možnosť vloženia objektov so zložitejšou geometriou bez interaktivity bez potreby hierarchického vytvárania objektu z podobjektov v A-Frame.

Z A-Frame entity je možné získať Three.js objekt pomocou funkcie *getObject3D* a rovnako aj vložiť objekt do entity funkciou *setObject3D*. Tento princíp je zobrazený na obrázkoch 1.18 a 1.19.

```
JS
entityEl.getObject3D('light');
```

Obr. 1.18: Demonštrácia získania Three.js objektu z A-Frame entity.
(Zdroj: [2])

Na takomto objekte je možné používať funkcie knižnice Three.js obsiahnuté v dokumentácii [26].

```js
AFRAME.registerComponent('pointlight', {
  init: function () {
    this.el.setObject3D('light', new THREE.PointLight());
  }
});
// <a-entity light></a-entity>
```

Obr. 1.19: Demonštrácia vloženia Three.js objektu do A-Frame entity. V tomto prípade tento objekt reprezentuje svetlo v scéne a je nastavený ako atribút light. Entita je množina objektov zložená z viacerých Three.js objektov.
(Zdroj: [2])

Tento prístup umožní zobrazenie 3D projekcií z JSROOT vo virtuálnej realite. V tomto prípade vieme získať neinteraktívny objekt, ktorý je možné zobraziť v rozšírenej realite XR ako objekt zameraný na vizualizáciu geometrie projektorov.

1.3.8 Komunitné rozšírenia A-Frame a LIRKIS G-CVE

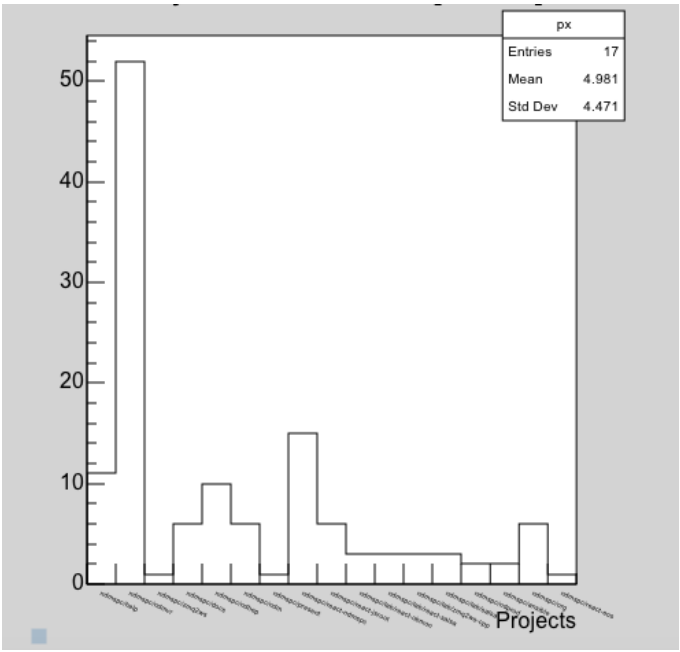
A-Frame komunita je zložená z developerov a inštitúcií podieľajúcich sa na vývoji riešení spájajúcich virtuálnu realitu. Riešenia sú vyvíjané v podobe komponentov, ktoré sú zverejnené pre použitie pri riešeniach podobných problémov. Samotné komponenty sú k dispozícií v A-Frame registri [27].

LIRKIS G-CVE je zdieľané virtuálne kolaboratívne prostredie vyvíjané na FEI KPI. Rozširujúce komponenty vyvíjané na katedre umožňujú následne rozšírenie možností a zlepšenie interakcie používateľov v online virtuálnom kolaboratívnom prostredí [28]. Ako príklad je možné uviesť komponent umožňujúci používanie chytrého zariadenia ako ovládača vo virtuálnej realite. Rovnako aj komponent poskytujúci informácie a kontrolu nad používateľmi, ktorí zdieľajú virtuálne prostredie [29]. Je vhodné spomenúť aj komponent HoloGOM umožnujúci používateľovi virtuálnej reality manipulovať s objektami pomocou gest rúk. Samotný komponent bol vyvinutý pre kolaboratívne prostredie LIRKIS, ale jeho využitie je možné aj v iných aplikáciách založených na A-Frame. Cieľom samotného vývoja komponentov je vytvorenie virtuálneho prostredia, v ktorom bude možné pripojenie a kolaborácia používateľov v reálnom čase a používatelia budú môcť plnohodnotne interagovať využitím rôznych ovládačov a chytrých zariadení [30].

Pre samotnú vizualizáciu, ktorá je cieľom tejto práce bude v neskorších fázach potrebné implementovať komponenty pre zabezpečenie zdieľania a vizualizácie fyzikálnych údajov v CVE. Pre komplexnosť a rozsiahlosť bude táto časť súčasťou ďalšej práce.

1.3.9 Vizualizácia 2D histogramu v A-Frame

Otázka spočíva aj v možnosti zobrazenia jednorozmerných histogramov s označením TH1 ako objekt v rozšírenej realite. Táto časť rozširuje analýzu JSROOT riešenú v časti 1.2.4 hlavne kvôli potrebe znalostí z A-Frame a komponentových rozšírení samotnej komunity. Tento typ histogramov je špecifický hlavne údajmi, ktoré sú ľahko zobrazované v 2D priestore, a preto klesá výhoda reprezentácie tohto zobrazenia v rozšírenej realite. V scéne tak najefektívnejším riešením je zobrazenie na elemente, ktorý by mohol predstavovať tabuľu, kde bude zobrazený tento histogram. Príklad projekcie je zobrazený na obrázku 1.20.



Obr. 1.20: TH1 histogram

Problém je v zložitej interaktivite, ktorá nemusí ideálne fungovať v prostredí rozšírenej reality. Pre zistenie možností sme sa zamerali na analýzu tohto typu histogramu a jeho existencie v DOM štruktúre, kde histogram je vykreslený ako SVG element. Ako uvádza zdroj [31], SVG môžeme charakterizovať ako jazyk pre tvorbu grafiky použitím jazyka XML. Hlavné uplatnenie nachádza hlavne vo vektorovej grafike kde využíva matematický popis objektov. Zdroj však uvádza, že SVG nie je len statická vektorová grafika ale rovnako môže obsahovať aj pokročilejšiu interaktivitu a animácie.

V našom prípade sme skúsili umiestniť element z DOM do scény v A-Frame [2]. Ako výsledok sme získali zobrazené SVG v scéne, stratili sme však možnosť pohybu v scéne. Rovnako sme skúsili aj SVG vložiť priamo do entity v scéne. Výsledok bol identický a pri vstupe do VR modu sme samotné SVG ani nevideli v

scéne.

Ako ďalší krok sme sa rozhodli otestovať verejné komunitné komponenty pre A-Frame. Ako prvý komponent sme skúsili použiť komponent, určený na vnorenie HTML kódu do scény [32]. Pokus bol vytvorený na jednoduchej demo A-Frame scéne kde sa nainštaloval komponent podľa dokumentácie a následne sa pokúsil zobraziť histogram priamo na "webe"vo virtuálnej realite. Samotný SVG element komponent nezobrazil histogram. Ako výsledok sme evidovali varovanie priamo v konzole prehliadača o šírke a výške elementu, ktorá nadobudla hodnotu 0. Samotné varovanie vyprodukovala funkcia knižnice JSROOT ako odpoveď na zlý formát údajov o rozmeroch výsledného objektu. Pri vykreslení histogramu je potrebné vytvoriť cieľový div element, ktorému je potrebné zadefinovať rozmery formou CSS štýlu. Samotné rozmery elementu sa dajú zadefinovať rovnako aj pomocou atribútov **height** a **width** elementu, a tak v tomto prípade nie je potrebné používať CSS štýl na nastavenie rozmerov. Funkcia *draw* alebo *redraw* knižnice JSROOT potrebuje pre vytvorenie svg elementu histogramu rodičovský komponent s rozmermi nastavenými CSS štýlom. Dôležité je aj spomenúť fakt, že samotné SVG vytvorené v DOM nemá definované pevné rozmery, čo je dôležité v prípade manipulácií a rôznych konverziách s ním. Pevná šírka a výška SVG elementu je definovaná jeho rodičovským elementom, ktorý je potrebné vytvoriť manuálne. Viac o princípe vytvárania histogramu je v časti 1.2.4. V závere tohto experimentu môžeme potvrdiť, že komponent nedokáže vytvoriť ekvivalent zobrazenia histogramu vo virtuálnej realite.

1.4 Ďalšie použité softvérové knižnice

Cieľom práce je predovšetkým vytvoriť vizualizáciu dát a zabezpečiť možnosť interakcie. Je pravdepodobné, že nastanú viaceré problémy spojené hlavne s princípom obmeny dát. Dáta sa môžu meniť v závislostí od vstupu používateľa a riešenie by následne malo prezobraziť len isté dáta, ktoré sú v danom čase potrebné. Rovnako aj schopnosť asynchrónne reagovať na zmeny dát a následne na to reagovať. Hlavnou ideológiou bude vytvoriť riešenie na úrovni A-Frame bez využitia ďalších technológií aby sme predišli problémom s kompatibilitou a aby sa na riešení dalo stavať aj v budúcnosti. Na návrh RNDr. Martina Vaľu, PhD a vzhľadom na to, že riešenie by malo mať potenciál, aby bolo využiteľné v už používaných aplikáciách založených na React-e [33]. Z tohto dôvodu sa pokúsime v rámci možností vytvoriť variant integrácie primárne s React-om. Pre riešenie problémov komunikácie a synchronizácie medzi komponentami na rozdielnych

vrstvách, kde neexistuje žiadna možnosť výmeny dát v reálnom čase sme sa rozhodli na návrh konzultanta využiť knižnicu RxJS [34].

1.4.1 React

React, ako je definovaný v dokumentácii [33] je knižnica využívaná pri vývoji používateľských rozhraní aplikácií na webe. Samotná Virtuálna realita sa dá považovať za druh používateľského rozhrania, v ktorom je častokrát potrebné reagovať na vstupy používateľa zmenami scény. V tomto prípade je možné nájsť optimalizované riešenia použitím tejto knižnice. React 360 [35] je knižnica podporujúca vytváranie zážitkov vo virtuálnej realite priamo vyvíjaná ako rozšírenie Reactu.

Pre samotnú vizualizáciu v rozšírenej realite sme sa rozhodli na pokyn vedúceho pre voľne dostupný softvérový rámec A-Frame a preto sme hľadali možné riešenia využitia tohto softvérového rámca s knižnicou React. A-Frame je zložený z primitív, ktoré sú podobné html elementom. Zdroj [36] zdôrazňuje možnosť využitia A-Framu nie len v jednoduchej webovej stránke bez využitia moderných softvérových rámcov ale aj aplikáciách v spojení s Reactom [33], Angularom [37] a Vue.js [38].

1.4.2 Rx JS

Pri používaní viacerých softvérových rámcov a knižníc v jednom projekte je často potrebné zabezpečiť synchronizáciu a komunikáciu medzi súčasťami, medzi ktorými nevieme inak zabezpečiť synchronizáciu. "RxJS je knižnica na vytváranie asynchrónnych programov a programov založených na udalostiach pomocou pozorovateľných sekvencií"[34]. V tomto kontexte tak hlavným účelom tejto knižnice bude možnosť komunikácie v reálnom čase medzi viacerými programovými časťami. Dáta je potrebné vygenerovať a následne kompetentné ďalšie súčasti budú schopné reagovať na tieto udalosti asynchrónne.

1.5 Zhrnutie analýzy A-Frame a JSROOT

V tejto záverečnej kapitole zhodnotíme výsledky celej analýzy, načrtneme možné riešenia. Porovnáme jednotlivé prístupy a naznačíme riešenia, ktoré budeme aplikovať pri vytváraní implementácie. Viaceré skúsenosti sme získali už v časti 1.2.5 a rovnako aj pri analýze 1.3.2.

1.5.1 Aktuálna vizualizácia pomocou Three.js

Čiastočné riešenie problému je vizualizácia týchto dát na webe využitím knižnice JSROOT a knižnice *Three.js*, ktorý ponúka zobrazenie *Three.js* 3D objektov histogramov v scéne a následne scéna je zobrazená ako celok na webe spolu aj s obsahujúcimi objektami. Interaktivita v tomto prípade je riešená cez raycaster, ktorý predstavuje vektor citlivý na prieniky s objektami v scéne. Dôležité je spomenúť aj podstatu 3D objektu, ktorý predstavuje množinu obsahujúcu podobjekty s definíciou geometrie a ďalšími potrebnými informáciami na úrovni knižnice *Three.js*. Samotná interaktivita je riešená mimo 3D objekt, samotný objekt môže obsahovať dáta, ktoré sa však zobrazia po určitej interakcii.

1.5.2 Vizualizácia pomocou A-Frame

Pre vytvorenie zobrazenia histogramu pomocou knižnice *Three.js* je potrebné okrem objektu vytvoriť scénu a vytvoriť aj renderer pre zobrazovanie jednotlivých framov. A-Frame je založený na *Three.js* a obsahuje už radu týchto problémov vyriešených. Pri samotnej analýze sme zistili, že nevieme získať *Three.js* objekt histogramu z funkcií, ktoré poskytuje JSROOT ale je možné takýto objekt 3D, ktorý je určený na vloženie do scény a rovnako je možné tento objekt vložiť do entity vo A-Frame a tým ho zobrazíme v scéne A-Frame. Horšie to vyzerá s prenosom samotnej interakcie objektu z úrovne *Three.js* na úroveň A-Frame, kedže A-Frame obsahuje koncepcie, ktoré neumožňujú definovanie interakcií ako pri vytváraní scény s *Three.js* objektom a rendererom. V analýze bolo zistené, že metóda *draw*() v knižnici JSROOT vytvára scénu a tento objekt vkladá do scény. Objekt je vytváraný na základe informácií obsiahnutých v objekte histogramu, ktorý je definovaný v JSROOT.

1.5.3 Zhrnutie možných riešení

Nasledujúce faktory pri výbere optimálneho vyhovujúceho riešenia je možné zhrnúť do týchto bodov:

- Ako nevýhodu A-Frame objektu môžme považovať hlavne vyššiu náročnosť na vyrenderovanie entity, keďže entita obsahuje množinu objektov identických objektu *Three.js*. Pri virtuálnej realite sú to najčastejšie napríklad objekty definujúce geometrický objekt a objekt definujúci svetlo, čo je naviac ako v *Three.js*. Nevýhoda je hlavne pri vysokom počte entít v jednej scéne a rastúcim počtom náročnosť vykreslenia stúpa. Možnosť zobrazenia viace-

rých binov v scéne je náročnejšie na výkon hardvéru ako vizualizácia binov pomocou *Three.js* objektov.

- Samotný A-Frame poskytuje oveľa väčší potenciál pri vytváraní interakcií v AR scéne. Jednoduchšia možnosť definovania interaktivity na jednotlivé entity cez komponenty a rovnako aj možnosť rozpoznania viacerých nových eventov. Hlavne pri využití špeciálnych zariadení pre zobrazovanie AR. Využitie *Three.js* objektu a prenos interaktivity do A-Frame by vyžadoval zásah do štruktúr softvérového rámca, a to nie je ideálne riešenie.

- Pri vizualizácií 2D histogramov sme nenašli vhodný komponent alebo technológiu, ktorá by nám umožnila zobraziť interaktívny SVG element priamo v scéne. Preto sme sa v tomto prípade rozhodli aspoň získať dvojrozmerný obraz projekcie z DOM a umiestniť ho na entitu vo forme textúry, ktorá bude vo VR móde signalizovať aktuálnu projekciu bez možnosti interaktivity pre približnú orientáciu. Pre hlbšie preskúmanie binu tak bude potrebné opustiť VR mód a skúmať dáta využitím klasickej funkcionality, ktorú ponúka JSROOT.

2 Návrh riešenia

V tejto časti bude kladený dôraz na návrh riešenia a jeho implementáciu. V analýze bolo získané množstvo informácií a riešení, ktoré sa počas nej otestovali na experimentoch. Ako vyplynulo z analytickej časti budeme pri implementácii najčastejšie používať softvérový rámec A-Frame [2] a knižnice React [33], Three.js [26] a JSROOT [1].

Obsah tejto kapitoly bude rozdelený do dvoch častí. V prvej časti sa bude klásť dôraz na určenie finálnej podoby riešenia, ako by samotné riešenie malo byť vytvorené a používané. V tejto časti budú popísané prípady použitia riešenia, návrh používateľského rozhrania a samotný konceptuálny návrh riešenia. V druhej časti bude popísaná implementácia riešenia na základe návrhu z prvej časti tejto kapitoly. V prvom rade analýza objektov a operácií, ktoré musí riešenie poskytovať.

2.1 Podstata riešenia

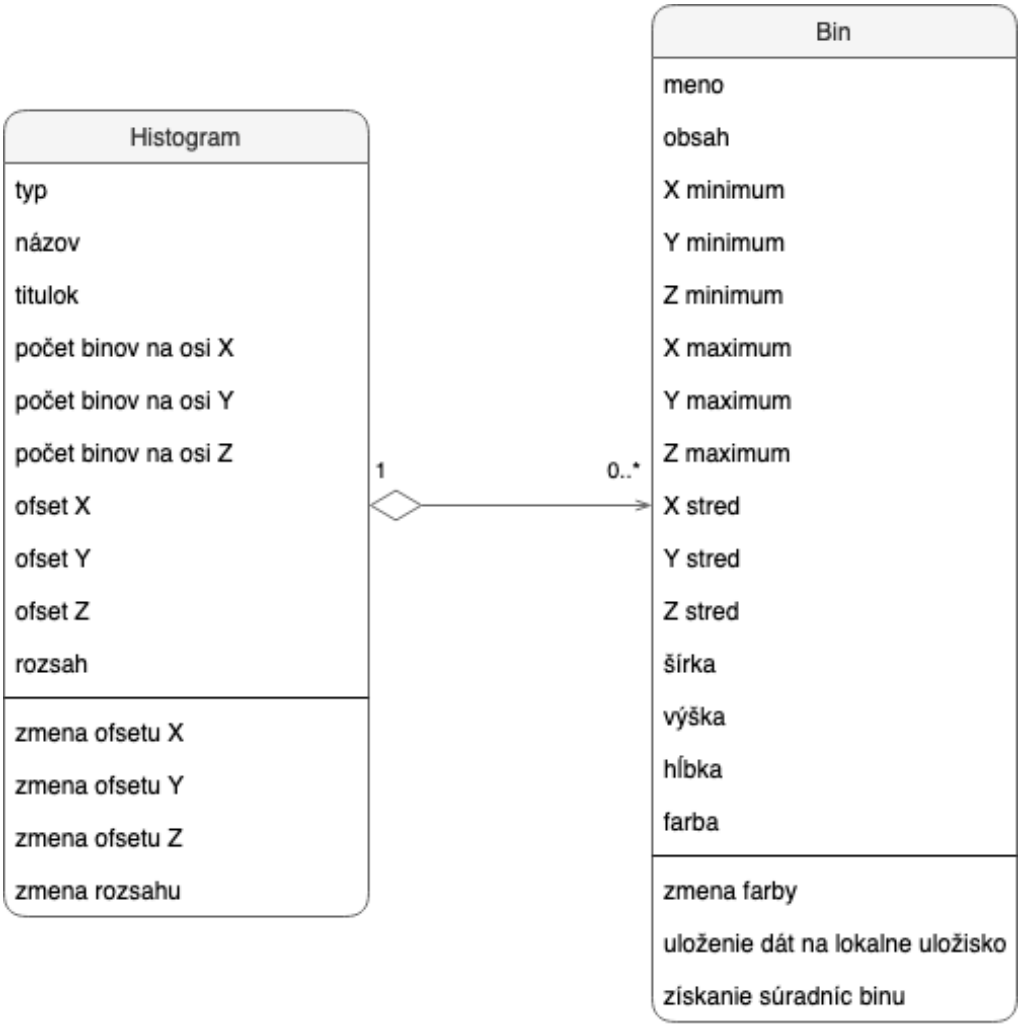
Riešenie tohto problému spočíva vo vytvorení zobrazenia dát v XR. Ako bolo zistené pri analýze dát tak podstata dát spočíva v ich kvantite a ich charaktere. Cieľom zobrazenia týchto dát je na základe všeobecnejších informácií zobraziť určitú sekciu jednoducho a v primeranej kvantite. Dáta sú neskôr konkretizované na základe interakcie používateľa a je zobrazená iná sekcia s konkrétnejšími dátami. Tento cyklus sa opakuje až dôjde ku konkrétnym dátam, ktoré sú zobrazované formou histogramov a vzniká cesta z počiatočného stavu do cieľového stavu. Ako počiatočný stav môžeme považovať histogram s dátami, ktoré sú veľmi obšírne a všeobecné. Následne každý uzol umiestnený v tejto ceste obsahuje konkrétnejšie dáta až dôjde do cieľového stavu. Cieľový stav nám zobrazuje najpodrobnejšie a najkonkrétnejšie dáta.

2.1.1 Výber najoptimálnejšieho riešenia

Ako zhrnutie vieme povedať, že na základe informácií z objektu histogramu (získame ho ako výsledok načítania z root súboru alebo vytvorením) je možné vy-

tvoriť objekt *Three.js*, ale rovnako aj objekt zložený priamo z A-Frame entít, ktorý bude reprezentovať histogram.

Samotný histogram vieme definovať ako množinu, ktorá obsahuje biny. Histogramy, ktoré obsahujú obrovský počet binov je nutné prispôsobiť tak, aby boli optimálne na zobrazenie. Pre tento prípad je potrebné definovať atribúty, ktoré budú určovať aktuálnu podmnožinu zobrazených binov ako je zobrazené v diagrame 2.1. Biny obsahujú dáta o sebe.



Obr. 2.1: Vzťah histogramu a binu zobrazený pomocou diagramu tried

Ako riešenie sme sa rozhodli vytvoriť knižnicu, ktorá bude obsahovať všetky potrebné metódy a A-Frame komponenty, ktoré zabezpečia vytvorenie histogramu z A-Frame entít. Každá entita bude predstavovať bin histogramu. Samotná knižnica na základe histogramu vytvoreného využitím knižnice JSROOT vytvorí histogram a vráti entitu histogramu, ktorá sa vloží do scény. Samotný problém pri vysokej dimenzionalite dát sme sa rozhodli vyriešiť definovaním sekcie zloženej z ofsetov na všetkých osiach a koeficientom, ktorý určí počet binov zobrazený v

každej dimenzii. Podľa týchto informácií sa zobrazí len určitá sekcia histogramu. Ofsety a koeficient si bude môcť definovať používateľ, aby v prípade väčších hardvérových možností mohol zobraziť viac binov.

Pre manažment zobrazovania sekcií sme sa rozhodli využiť softvérový rámec React na základe dobrej kompatibility s A-Frame. Knižnica bude integrovaná do balíku (package) obsahujúceho React komponent určený na jednoduché použitie v klientskej aplikácií. Používateľ si nainštaluje tento balík a následne využije komponent vo svojej aplikácií na zobrazenie dát.

2.2 Prípady použitia

Riešenie problému nepredstavuje hotovú aplikáciu. Riešenie predstavuje nástroj na vytvorenie histogramu. Zapuzdrenie umožní využiť tento balík v rôznych aplikáciách založených na React-e a umožní tak pridať istý druh funkcionality v cieľovej aplikácií. Funkcionalita sa do aplikácie pridá formou React komponentu. Komponent vytvorí histogram určený na zobrazenie histogramu z objektu histogramu, ktorý prijme z aplikácie ako parameter rovnako aj s údajmi o sekcii, ktorú má zobraziť, keďže častokrát histogramy obsahujú vysoký počet binov.

React komponent je v našom prípade priorita, ale rovnako sa pokúsime vytvoriť aj komponent zabezpečujúci vytvorenie histogramu na webe s absenciou React-u a podobných softvérových rámcov. V tomto prípade komponent musí byť sebestačný a musí obsahovať dobre implementované funkcie pre manažment stavu histogramov a rovnako aj renderer schopný efektívne aktualizovať DOM elementy. V prípade React komponentu sa o túto funkcionalitu stará samotný softvérový rámec.

Pre návrh knižnice je však potrebné dôkladne analyzovať prípady použitia, pretože len zobrazenie dát nie je postačujúce a potrebná je aj interaktivita a, aby bolo možné vykonať potrebné akcie. Ako cieľového používateľa môžme považovať vedeckého pracovníka podieľajúceho sa na experimentoch, ale aj pracovníka v inej oblasti, ako medicína alebo oblasť financií keďže samotný JSROOT je knižnica využívaná nie len v CERN-e.

2.2.1 Zobrazenie časti histogramu

Z charakteristiky dát vyplýva viac dôležitých informácií. Medzi hlavnú charakteristiku môžme zaradiť hlavne obrovské množstvo dát, ktoré je spracované každú sekundu. Z tohto dôvodu by bolo nepraktické a veľmi náročné zobrazovať všetky entity z hľadiska výpočtovej náročnosti a rovnako aj z pohľadu používateľa, čo by

mohlo spôsobiť chaos. Príklad použitia môže obsahovať klientsku aplikáciu, ktorá obsahuje formulár pre používateľa. Vedecký pracovník vie vybrať približne sekciu histogramu, ktorej dáta chce skúmať a následne sa histogram spolu s údajmi o sekcii poskytne komponentu, ktorý vytvorí aktuálny náhľad sekcie histogramu pre používateľa.

2.2.2 Zmena zobrazenej časti histogramu

Vedecký pracovník v mnohých prípadoch nevie aké dáta si chce zobraziť v konkrétnejšej podobe. V tomto prípade použitia si zobrazí histogram za účelom analýzy dát a nevie vopred, kde sú hodnoty, ktoré chce zistiť. Za úspešný prípad použitia je možné považovať stav, kedy používateľ si v danom zobrazení dokáže postupne posúvať zobrazenú sekciu a postupne tak systematicky prechádzať celý histogram, a to po osiach **x**, **y**, **z**.

Dôležité je aj zabezpečiť túto interaktivitu bez nutnosti opustenia VR módu.

2.2.3 Označenia binov a zobrazenie príslušných informácií

Ďalšia z dôležitých interakcií je možnosť vedeckého pracovníka, pri skúmaní zobrazených binov označiť bin, na ktorý sa zameriava. Ako úspešné použitie je možné považovať pohľad na bin a potvrdenie akcie. Následne sa zobrazia informácie o označenom bine vo forme projekcií. V prípade, že žiadny bin nie je označený ostávajú zobrazené údaje o bine, ktorý bol označený ako posledný. Biny prezentujú svoje dáta svojím vzhľadom, ale pre lepšie pochopenie sú dáta aj zobrazované.

2.2.4 Výber binu používateľom

Medzi jeden z kľúčových prípadov použitia, môžeme považovať prípad kedy vedecký pracovník nájde bin, pre ktorý chce zobraziť ďalší histogram s konkrétnejšími dátami. Za úspešné použitie považujeme, keď používateľ zamieri pohľad na bin a vykoná akciu pre uloženie údajov o bine. Údaje sa následne pošlú na vzdialený klaster a po vykonaní potrebných akcií vráti výsledok v podobe URL adresy na root súbor s náležitými údajmi. Táto časť komunikácie a získavanie údajov z klastrov bude riešená v inej práci.

2.2.5 Manažment pozície v histograme

Samotné riešenie bude implementované vo VR a dôležité je zabezpečiť, aby vedecký pracovník mal možnosť meniť svoju pozíciu v histograme pre účely konkretizácie niektorých časti histogramu alebo na zmenu pohľadu na histogram. Ako úspešné použitie je v tomto prípade zmena pozície po vykonaní náležitej akcie napríklad za účelom priblíženia protiľahlej strany histogramu. Rovnako za úspešný prípad použitia môžeme považovať aj teleportáciu vedeckého pracovníka k vybranému binu za účelom jeho detailnejšieho preskúmania.

2.3 Návrh používateľského rozhrania

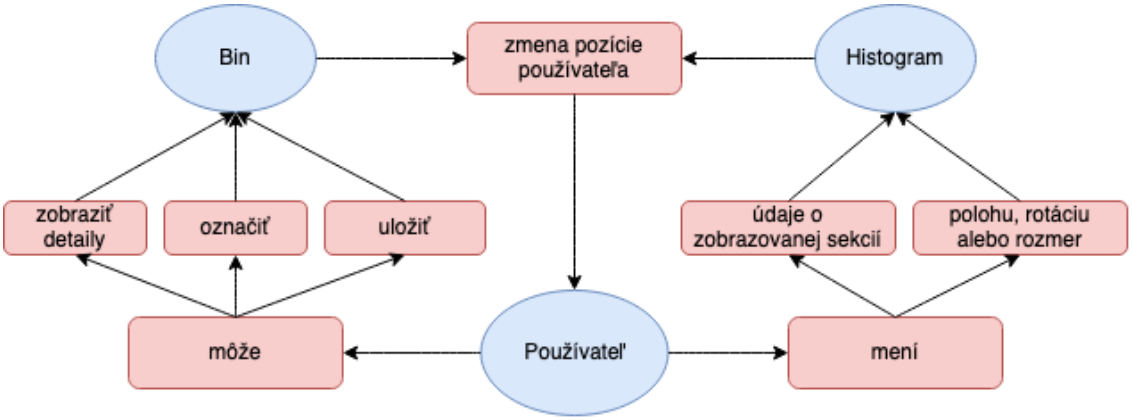
Medzi hlavné výhody VR patrí maximalizácia zážitku používateľa, ktorý má možnosť sa vcítiť do prostredia a interagovať s okolím. Okrem zážitku používateľa však virtuálny priestor poskytuje aj dostatočné rozsiahle prostredie pre zobrazenie dát a samotné informácie môžu byť zakódované do rôznych tvarov, farieb, pozícií, vzdialeností alebo úrovní.

Pre správne vytvorenie efektívneho používateľského rozhrania pre používateľa je dôležité určiť cieľovú doménu zariadení, na ktorých sa bude dané zobrazenie zobrazovať prioritne. Je k dispozícií viac typov zariadení podporujúcich zobrazenie webových stránok vo VR ako počítače alebo špeciálne zariadenia vyvinuté pre účel zobrazovania VR, ako napríklad oculus a mnohé ďalšie. AR mód podporujú hlavne niektoré druhy smartfónov.

Pri tvorbe používateľského rozhrania sme sa rozhodli zamerať hlavne na zariadenie Oculus pre jeho dostupnosť na KPI a pre dostupnosť komponentov v A-Frame. Okrem samotného Oculusu sme sa rozhodli prispôsobiť rozhranie aj pre účely zobrazenia na počítačoch.

2.3.1 Konceptuálny model

Pri analýze prípadov použitia sme vychádzali hlavne z existujúcej implementácie vizualizácie histogramov na webe. Riešenie musí okrem potrebných prídavných akcií obsahovať interaktivitu, ktorá je obsiahnutá v tejto vizualizácií (Obr. 2.2).



Obr. 2.2: Konceptuálny model používateľského rozhrania. Model pozostáva z objektov (modré objekty) a akcií (červené objekty), ktorými medzi sebou jednotlivé objekty interagujú.

Konceptuálny model popisuje všetky možnosti interakcie. Okrem interakcie, ktorá pozostáva z možnosti zobrazenia údajov o bine je potrebné, aby používateľské rozhranie poskytovalo možnosť zmeny zobrazenej časti a aj manažment pozície používateľa v histograme.

Konceptuálny model pozostáva z 3 objektov, medzi ktoré môžme zaradiť samotného používateľa, bin a histogram. Interakciu môžme rozdeliť medzi 2 objekty, medzi interakciu používateľa s binom a interakciu používateľa so samotným histogramom. Používateľ má možnosť označiť viac binov. Používateľ si môže zobraziť aj detailné informácie o jednom bine v istom čase a rovnako aj zvoliť jeden bin, ktorého dáta spracuje a vzdialený server poskytne náležité dáta. Pri zobrazení vo VR a obmedzenom počte zobrazených binov je dôležité zabezpečiť možnosť zmeny podmnožiny binov. Táto akcia umožňuje používateľovi zmeniť údaje o zobrazovanej sekcii a rozsahu. Pri zobrazení na zariadení s klávesnicou je možnosť pre používateľa manipulovať s pozíciou histogramu.

Pri zobrazení vo VR je nutné zabezpečiť aj zmenu polohy používateľa v histograme keďže sa nedá predpokladať, že používateľ bude mať dostatok priestoru na to, aby sa vedel svojvoľne pohybovať po celom histograme. Ako je zobrazené v konceptuálnom modeli (Obr. 2.2) samotná pozícia používateľa sa bude modifikovať podľa pozície zvoleného binu alebo polohovanie po osiach histogramu.

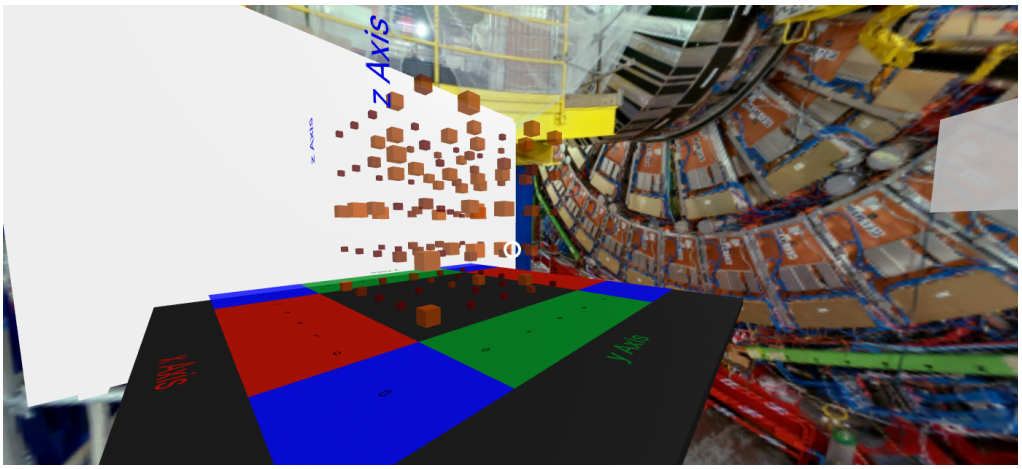
2.3.2 Prototyp používateľského rozhrania vo virtuálnej realite

Ako bolo spomenuté, samotná XR je prostredie v ktorom používateľ interaguje s objektami. Narozdiel od klasických používateľských rozhraní na webe kde používateľ komunikuje s aplikáciou pomocou komponentov zobrazovaných v 2D

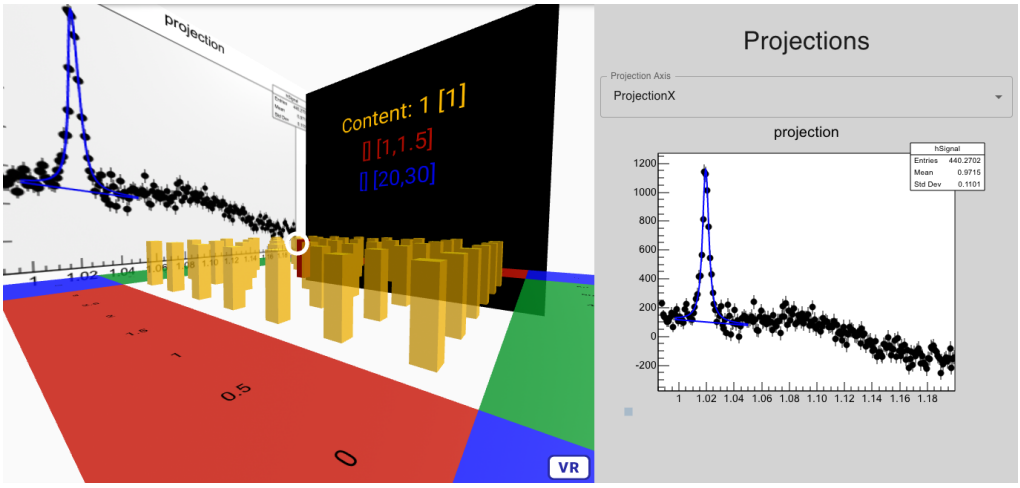
priestore je tento typ používateľského rozhrania špecifický hlavne pozíciou používateľa ako objektu, ktorý interaguje s prostredím. Pre maximalizáciu zážitku je používateľ umiestnený priamo do histogramu. Má možnosť sa prechádzať v histograme, ktorý tvorí areál obsahujúci jednotlivé biny. Biny sú dostatočne veľké a obsahujú rozostupy, čo je aj rozdiel v porovnaní s aktuálnym zobrazením histogramov pomocou funkcií knižnice JSROOT.

Pre zobrazenie informácií o binoch bol použitý baner obdobný informačnej tabuli pre zobrazovanie dát. Po označení binu sa používateľ pozrie na príslušný baner, kde uvidí informácie o jednotlivých binoch a rovnako aj mapovanie údajov k hodnotám na osiach. Osi sú odlíšené farebne a aj súradnice zobrazené v baneri sú zafarbené podľa toho na akej osi sú definované. Po pravej strane sa používateľ dozvedá názov súboru, z ktorého bol načítaný histogram. Po ľavej strane sú zobrazené informácie o jednotlivom type dát, ktoré sú zobrazované na jednotlivých osiach. Text je dostatočne vzdialený a veľký, aby sa zabránilo jeho vychyľovaniu pri pohybe používateľa. Pri pohľade na bin sa bin označí fialovou farbou, čo indikuje používateľovi možnosť vykonať akcie súvisiace s týmto binom ako sú označenie, zobrazenie detailov a uloženie informácií pre ďalšiu interakciu.

Samotný bin je možné aj označiť ,vtedy nadobudne bin oranžovú farbu a zmení sa aj pozícia používateľa voči tomuto binu. Náhľad prototypu vo VR je znázornený na obrázkoch 2.3 a 2.4.



Obr. 2.3: Náhľad prototypu rozhrania vo VR móde



Obr. 2.4: Náhľad prototypu rozhrania vo VR s použitím komponentu pre zobrazenie TH1 histogramu

2.3.3 Zobrazenie na zariadeniach s klávesnicou

Akcie je možné vykonávať zadaním určitých vstupov. V prípade použitia zariadení, ktoré majú klávesnicu je možné tieto vstupy namapovať na jednotlivé klávesy. A-Frame obsahuje komponent **wasd-controls**, ktorý dodá základné klávesové mapovanie pre pohyb používateľa vo virtuálnom priestore. Pri zobrazovaní vizualizácie na stolnom počítači sa tak používateľ môže spoľahnúť na klávesy WASD umožňujúce pohyb po scéne. Pre vykonávanie špecifických akcií sa implementoval ovládač, ktorý nám toto umožní.

2.3.4 Zobrazenie na zariadeniach podporujúcich mód virtuálnej reality

Pri tomto zobrazení sa maximalizuje používateľský zážitok. Pre pohyb v histograme je využitý A-Frame komponent **look-control**, ktorý umožnuje používateľovi sa voľne pohybovať a rozhliadať v histograme.

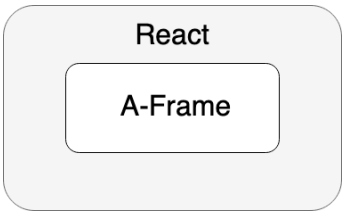
Na implementáciu akcií sa využili komponenty umožňujúce rozpoznávať udalosti na vstupe z ovládačov zariadenia Oculus. Používateľ vykonáva špecifické akcie použitím ovládačov, ktoré má k dispozícii zariadenie Oculus. Pri použití tohto zariadenia, ktoré vťahujú používateľa do virtuálnej reality je pohyb v scéne identický ako používateľov pohyb v reálnom priestore. Je dôležité pre tento typ umožniť používateľovi možnosť aspoň približne meniť pozíciu v histograme pomocou ovládačov, keďže samotný používateľ nemusí mať okolo seba toľko reálneho priestoru koľko si vyžaduje samotná scéna na preskúmanie každého binu v histograme.

2.4 Návrh architektúry komponentu pre vizualizáciu histogramu z hľadiska použitých technológií

Riešenie pozostáva z vytvorenia logiky, ktorá spracuje údaje z objektu histogramu do vizuálnej. Keďže je použitý A-Frame, samotný histogram bude reprezentovať entita. Jednotlivé biny budú taktiež reprezentované entitami a vložené do entity histogramu na pozíciu, ktorá entite (bin histogramu) prislúcha. Týmto riešením vytvoríme objekt histogramu, ktorý po vložení do A-Frame scény bude reprezentovať histogram.

V prípade, že histogram je vizualizovaný v A-Frame scéne je potrebné zabezpečiť interaktivitu používateľa. Samotná interaktivita musí byť na úrovni A-Frame, aby bolo možné zadávať vstupy aj zo zariadení, ktoré používajú špeciálne VR komponenty. Interaktivita bude histogramu dodaná pomocou A-Frame komponentov. Dôležité je aj zabezpečiť komunikáciu medzi A-Frame a React.

Celkový konceptuálny návrh je možné rozdeliť z hľadiska ich bázových funkcionalít a architektúr. Riešenie pozostáva prevažne z dvoch technológií kde ich integráciu by sme mohli definovať prevažne, ako je zobrazené na obrázku 2.5.



Obr. 2.5: Rozdelenie a zapuzdrenie 2 hlavných technológií v projekte

2.4.1 React vrstva

Účel tejto vrstvy v návrhu riešenia je hlavne zabezpečenie manažmentu z najvyššej úrovne aplikácie. Ako sa samotný React využíva v bežných aplikáciách na tvorenie používateľských rozhraní rovnaký cieľ tejto knižnice bude aj v našom návrhu. V našom prípade je potrebná optimalizácia zobrazovaných dát vo forme zobrazovania určitých sekcií histogramov a efektívne vytváranie samotných entít použitím JSX.

Hlavnou úlohou knižnice v našom návrhu bude hlavne aktualizácia zobrazenia histogramu (zmena obsahu zobrazovaných entít). Aj keď knižnica je určená na používateľské rozhrania a pokrýva aj metódy na spracovanie vstupov od používateľa, v tomto prípade sa React použije len na zabezpečenie interaktivity, ktorá nemá priamy súvis so vstupom používateľa. Dôvod je absencia metód pre

evidenciu vstupov pri špeciálnych zariadeniach a ovládačoch určených pre VR. V prípade interaktivity závislej na vstupe od používateľa tak nutné rozdeliť túto interaktivitu. Používateľský vstup spracuje A-Frame vrstva obsahujúca podporu pre rozličné VR zariadenia a formou signálu informuje nadradenú React vrstvu pre vykonanie istej funkcionality.

V praxi to znamená, že hlavné výstupy tejto vrstvy budú vo forme React komponentov. Využije sa najsilnejšia vlastnosť knižnice, ktorá spočíva v efektívnej schopnosti obmieňať DOM na základe rozdielov s virtuálnym DOM-om Reactu.

2.4.2 A-Frame vrstva

Ako riešenie problému prvej vrstvy je nutné zabezpečiť možnosť rozpoznať vstupy od rôznych zariadení. Vrstva je integrovaná v React vrstve. A-Frame obsahuje vstavané komponenty, ktoré nám toto umožnia. Táto vrstva bude obsahovať prevažnú väčšinu interaktivity v našom riešení rovnako aj manažment tvaru entít a efektov (Narozdiel od nadradenej React vrstvy nie zmena obsahu entít, ale len tvary a efekty entít). Samotná interaktivita sa implementuje formou komponentov (entity component system [2]), ktoré spĺňajú istý druh funkcionality a tá sa následne pridelí entite ako atribút podľa potreby. Interaktivita na báze zmeny farby, priehľadnosti, pozície, rozmerov, animácie entít bude pokrytá v tejto vrstve.

V praxi to znamená, že hlavné výstupy tejto vrstvy budú vo forme A-Frame komponentov. Využije sa hlavne vlastnosť evidovať vstupy zo špecifických zariadení a rovnako aj efektívna manipulácia s objektami v XR scéne.

3 Implementácia riešenia

Implementácia riešenia spočíva vo vytvorení komponentov a tried, ktoré zabezpečia zobrazenie histogramu rovnako ako aj možnosť interakcie. V návrhu boli definované jednotlivé vrstvy, ich úloha a funkcionalita, ktorú musia spĺňať na to aby riešenie spĺňalo svoj účel. V implementácii sa popisuje konkrétna architektúra vrstiev spolu s triedami a komponentami, ktoré dodávajú riešeniu finálny rozmer.

3.0.1 Architektúra React vrstvy

Vrstva popisuje finálny React komponent a ďalšie potrebné komponenty pre export ku klientskej aplikácií (Obr. 3.1). Komponenty **NdmVrHistogram3D**, **NdmVrCamera** sú hlavnými komponentami, ktoré umožnia vytvoriť vizualizáciu v scéne XR.

NdmVrHistogram3D komponent vytvorí 3D histogram ako nezávislú entitu, ktorú je možné vložiť do scény.

NdmVrCamera slúži len na vytvorenie entity, ktorá má účel používateľskej kamery v scéne. Dôležité je hlavne poskytnúť správne entity so správnymi identifikátormi pre funkčnosť interaktivity s binmi a histogramom.

Finálny komponent je **NdmVrScene**, ktorý bude dostupný pre klienta a bude predstavovať celú scénu s importovanými komponentami pre vytvorenie kamery a histogramu tak, aby sa zabezpečila súdržnosť. Znemožníme tak klientovi zostavovať scénu a vyhneme sa problémom, ak by klient neposkytol všetky potrebné komponenty pre jej zostavenie.

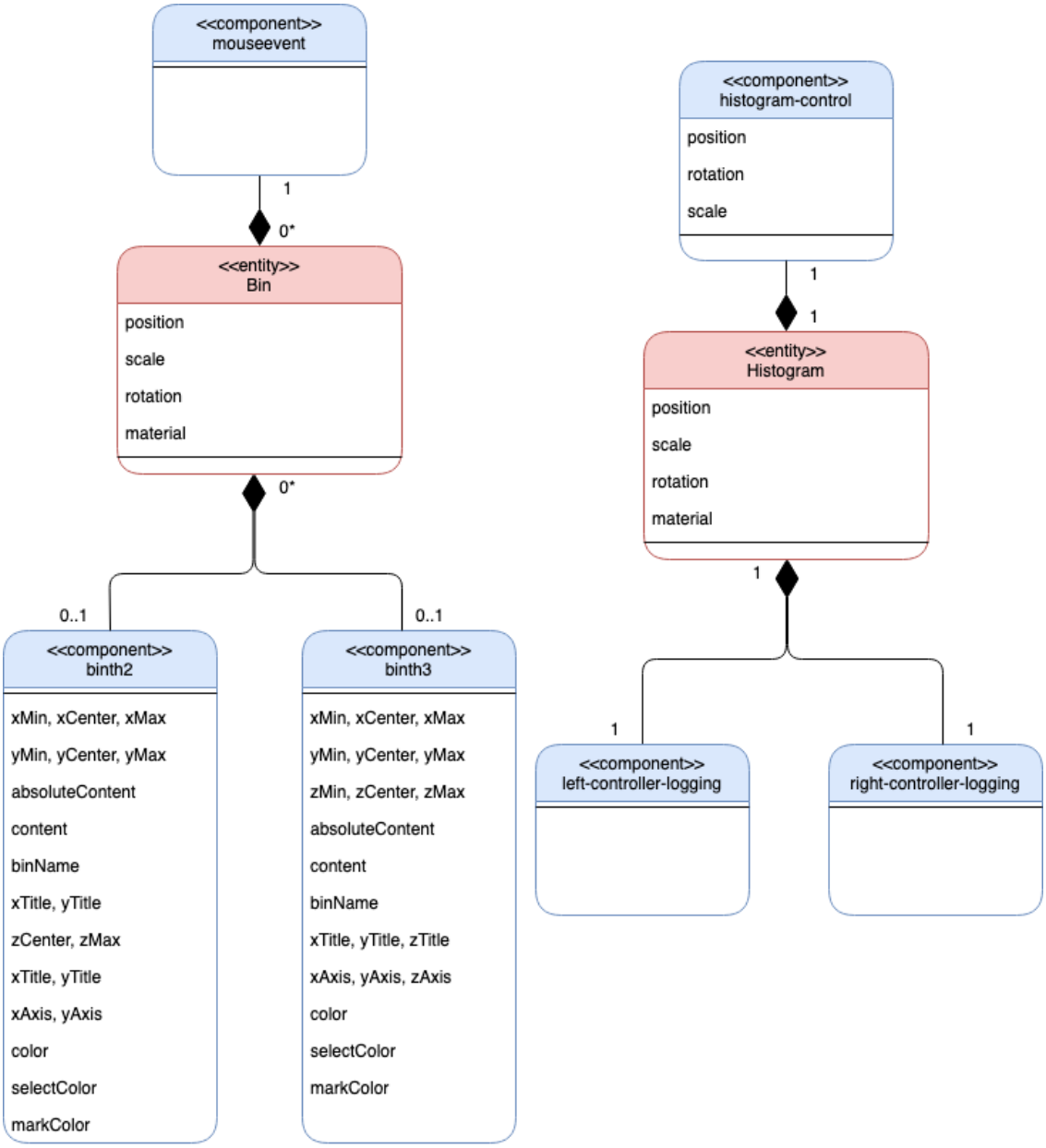
Obr. 3.1: Diagram tried pokrývajúci prvú vrstvu architektúry na úrovni React

JsrootHistogram v spojení s komponentom **NdmVrHistogram3D** vytvorí aktuálny obraz SVG elementu histogramu a vykreslí ho aj do VR na entitu ako rastrovú textúru. Samotnú cieľovú entitu vytvorí komponent **NdmVrHistogram3D** v histograme.

Pre prácu so zdrojmi a s elementami potrebnú v procese vytvárania histogramu slúžia uvedené servisné triedy. Hlavnou triedou pri vytváraní samotného histogramu je trieda **HistogramFactory**, ktorá implementuje tieto servisné triedy. Trieda **ThemeProvider** by mohla byť za istých okolností nazvaná taktiež ako servisná trieda, ale jej úlohou je primárne len evidencia aktuálnej farebnej palety. Provider poskytuje paletu prislúchajúcich farieb triede **HistogramFactory** pri vytváraní histogramu podľa názvu témy. Zabezpečuje hlavne optimalizáciu histogramu vzhľadom na požiadavky používateľa. Farebné palety sú načítavané v súbore v objektoch presne podľa určenej schémy.

3.0.2 Architektúra A-Frame vrstvy

Vrstva pozostáva z implementovaných A-Frame komponentov, ktoré sa zaregistrujú a pridajú do entít ako HTML atribúty. Pre vytvorenie finálneho komponentu pre použitie v klientskych aplikáciách, React komponent vytvára entity s týmito komponentmi a tak sa dosiahne požadovaná interaktivita (viac na obr. 3.2). Komponenty sú rozdelené na komponenty, ktoré pridávajú entite prináležitú funkcionalitu, ako napríklad komponent definujúci **mouseevent** schopnosť binu reagovať na prieniky *raycastera*. Druhou kategóriou sú komponenty, ktorých úloha je uložiť potrebné dáta do entity, napríklad komponenty **binth2** a **binth3** slúžiace na uloženie dát do entity.



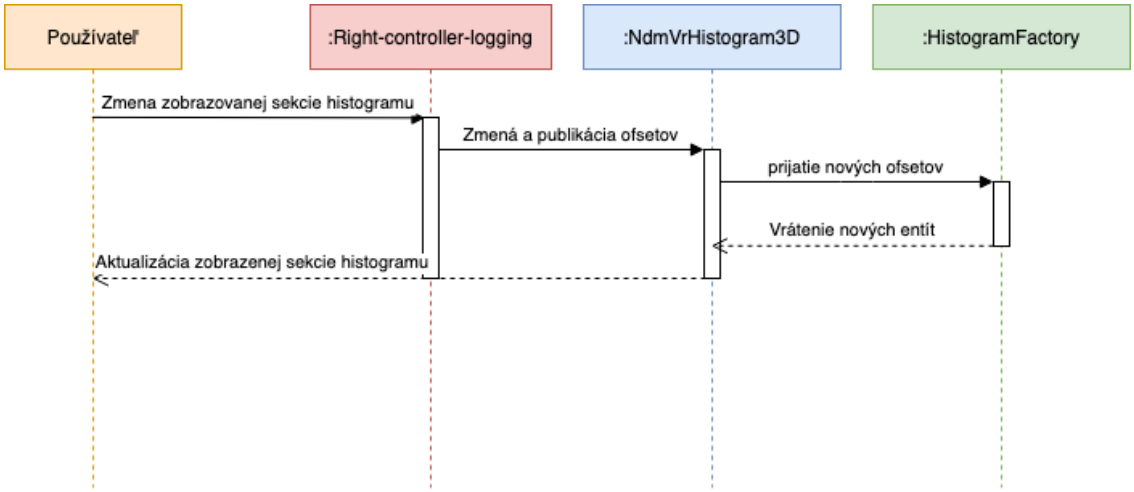
Obr. 3.2: Diagram tried pokrývajúci druhú vrstvu architektúry na úrovni A-Frame

Komponenty **histogram-control** a **left a right-controller-logging** v spojení s entitou definujúcou histogram umožnujú pohybovať histogramom, meniť jeho veľkosť, rotáciu a umožňujú rozpoznávať udalosti vyvoláné použitím ovládačov Oculus a následne reagovať na ne.

3.0.3 Komunikácia medzi vrstvami

Pre optimálne fungovanie interaktivity používateľa s binmi a histogramom je však potrebné niekedy zabezpečiť aj komunikáciu medzi A-Frame-om a React-om. Pre tento účel sme využili knižnicu RxJS. Samotný koncept je založený na

návrhovom vzore observer, kde objekt obsahuje metódy pre publikáciu dát a inštanciu, ktorá sa dokáže prihlásiť na odber dát a po vytvorení komunikačného kanálu sa tak publikované dáta prijmu v komponentoch, ktoré sa prihlásia na odber a následne komponenty môžu vykonať potrebné operácie.



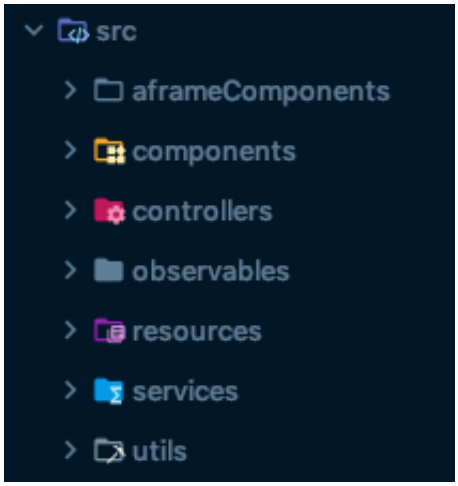
Obr. 3.3: Diagram interakcií znazorňujúci komunikáciu medzi jednotlivými vrstvami

V diagrame 3.3 je simulovaná situácia kedy používateľ chce zmeniť zobrazenú sekciu histogramu použitím ovládača zariadenia oculus. O zobrazenie sekcií sa stará React komponent **NdmVrHistogram3D**, ktorý získa aktualizované entity a následne upraví sekciu v DOM tak, aby spĺňala aktuálne požiadavky. React nedokáže rozpoznať udalosť kedy používateľ zadá vstup pomocou ovládača, a preto potrebuje získať potrebný signál alebo dáta. Vstup nám preto umožní rozpoznať komponent **right-controller-logging**. Komunikáciu nám umožni *subject* z knižnice RxJS, ktorý na strane A-Frame-u pri evidencií vstupu z ovládača publikuje správu s dátami a následne React komponent, ktorý je prihlásený na odber na tomto *subjecte* získa správu v reálnom čase a môže vykonať zmenú entít a aktualizáciu sekcie.

3.1 Súborová štruktúra webpacku

Webpack je balík, ktorý slúži na vytvorenie a distribúciu komponentov klientskej aplikácií. Výhodou tohoto prístupu je fakt, že pre klientskú aplikáciu sa doinštaluje potrebný balík a následne vývojár má prístup ku komponentom, ktoré sú poskytované a môže ich používať. Súborová štruktúra v našom webpacku bola rozčlenená do častí, ktoré majú čo najlepšie rozdeľovať funkcionalitu podčastí

webpacku a správne zapuzdriť logiku. Účel je rovnako a čo najlepšie zabezpečiť škálovateľnosť projektu v budúcnosti. Súborovú štruktúru tvoria podadresáre zobrazené na obrázku 3.4.



Obr. 3.4: Súborová štruktúra webpacku (WebStorm IDE 2019.3.5)

V nasledujúcej časti implementácie sa pozrieme na jednotlivé podadresáre podrobne. Rozoberieme a popíšeme ich účel a všetky skripty si podrobne rozoberieme. Rovnako pozrieme aj na možnosti rozšírenia v budúcnosti.

3.2 Adresár aframeComponents

Podadresár obsahuje implemetované A-Frame komponenty, ktoré sú súčasťou entít histogramu. Komponenty môžme rozdeliť podľa ich primárného účelu na inicializačné komponenty a interakčné komponenty. Tento princíp je výhodný hlavne pre sperhľadnenie kódu, pretože je potrebné často inicializovať do entít viac dát a interakčný komponent obsahuje taktiež väčší objem kódu. Podadresár obsahuje skripty, ktorých úlohou je registrácia komponentov pre ich ďalšie použitie.

3.2.1 Interakčné komponenty

CursorEventAframeComponent.js definuje interakčný komponent určený pre entitu binu. Jeho úlohou je zabezpečiť prevažne funkcionalitu umožňujúcu interakciu používateľa s binom, a to hlavne použitím kurzora (raycaster). Komponent neprijíma žiadné externé dáta jeho funkcionalita je implementovaná pomocou metód životného cyklu *init* a *update*. *Init* sa vykoná pri inicializácií entity, a preto obsahuje všetky naplánované eventListenery pre pokrytie interakcií ako *mouseenter*, *mouseleave*, *doubleclick*. Komponent eviduje aj stav binu podľa ktorého sa bin

prispôsobí. Funkcia *update* obsahuje zmenu atribútov, ktoré sa môžu meniť, ako napríklad zmena farby pri označení binu.

HistogramAframeComponent.js definuje interakčný komponent určený pre entitu hostogramu. Komponent obsahuje predvolenú schému dát určujúcu všetky potrebné údajé o pozícii, rozmeroch a rotácii. Komponent neprijíma žiadné externé dáta a jeho funkcionalita je implementovaná pomocou metód životného cyklu *init* a *update*. *Init* obsahuje inicializáciu ovládača pre ovládanie pomocou klávesnice a eventListenery pre stlačenie a uvoľnenie klávesy. *Update* zabezpečuje aktualizáciu referencie na kameru pri zmene histogramu pre prípad, keď React aktualizuje scénu a vytvorí novú inštanciu kamery.

LeftOculusController.js a **rightOculusController.js** sú interakčné komponenty slúžiace na zabezpečenie rozpoznávania eventov z ovládačov zariadenia oculus. Funkcionalita je definovaná vo funkciách, ktoré sú implementované ako moduly v adresári controllers. Komponent metóde *init* pomocou funkcií *addEvenListener* nastaví náležitú funkciu na rozpoznaný event. Komponenty obsahujú aj stavové premenné definujúce stav tlačidla **grip** na ovládači oculus pre zväčšenie počtu kombinácií pre vstup. Implementácia komponentov je obdobná až na rozdiel funkcií, ktoré sú volané pri jednotlivých eventoch. Pravý ovládač slúži na ovládanie histogramu a ľavý pre ovládanie kamery používateľa. Rozdiel je aj v prípade komponentu **leftOculusController.js**, ktorý obsahuje aj premennú *speed*, kde si používateľ nastaví rýchlosť pohybu po osiach x a y použitím ovládača.

3.2.2 Inicializačné komponenty

Komponenty **th2AframeComponent.js** a **th3AframeComponent.js** sú inicializačné. Ich funkcia je len prijať externé dáta a následne dáta spracovať a uložiť do entity ako objekt *userData*. Funkcionalita je implementovaná vo funkciách*init* a *update*. Obe funkcie majú rovnakú implementáciu a to uloženie dát zo schémy komponentu do *userData* v entite. Pri zmenách entity je potrebné taktiež aktualizovať dáta, o to sa postará funkcia *update*. Škálovateľnosť je zabezpečená v prípade vytvárania nových komponentov podľa potreby pre vytvorenie ďalších funkcionalít.

Komponent **labelHandlerAframeComponent.js** slúži hlavne na zmenu textovej hodnoty označení osí. Pri animácií je potrebné zmeniť hodnotu až po určitom čase.

3.3 Adresár services

Podadresár obsahuje servisné triedy určené na zapuzdrenie určitej funkcionality do jedného celku. Rôzne akcie v iných komponentoch, kde je potrebné pristupovať k lokálnemu úložisku, modifikovať pozíciu kamery alebo vytvoriť projekciu jednorozmerného histogramu.

3.3.1 Trieda Camera

Servis obsahuje funkcionalitu pre manipuláciu s pozíciou a rotáciou kamery. Trieda obsahuje konštruktor a funkciu, ktorá získava a neustále kontroluje existenciu kamery v DOM.

Pre jednotlivé akcie sú vytvorené funkcie, ktoré sa vykonávajú podľa potreby. Každá funkcia v prípade ak existuje kamera v scéne nástaví jej pozíciu alebo rotáciu a umožňuje tak používateľovi spravovať svoju pozíciu v scéne svojvoľne bez nutnosti fyzického kráčania k cieľu. V samotnej implementácií nám tento objekt umožní meniť atribúty volaním funkcií a prispieva tak k väčšej prehľadnosti v projekte.

3.3.2 Trieda Jsroot

Servis obsahuje implentáciu funkcií, ktoré sa starajú o vytváranie projekcií a ich vizualizáciu na úrovní knižnice JSROOT. Obsahuje funkcie *createTH1Projection*, *openTH1Projection* a *displayImageOfProjection*.

- **CreateTH1Projection** vytvorí projekciu na základe zvoleného binu a celého histogramu. Výsledná projekcia je vykreslená funkciou *draw* na výsledný DOM element, ktorého identifikátor zadávame ako parameter funkcie.
- **OpenTH1Projection** nevytvára ale otvára ROOT súbor, z ktorého sa funkciou *readObject* získava TH1 histogram. Simulácia bežného stavu kedy nie je výhodné vytvárať projekciu zložitého histogramu na strane klienta ale je výhodnejšie tieto dáta získať zo vzdialeného klastra v ROOT súbore.
- **DisplayImageOfProjection** slúži na zobrazovanie aktuálneho vizuálneho stavu SVG elementu v DOM na panel v XR. Ak DOM obsahuje SVG element s vizulizovaným elementom, funkcia vytvorí textúru aktuálneho náhľadu a vloží ju ako textúru na entitu v scéne.

3.3.3 Trieda NdmVrStorage

Servis slúži pre manipuláciu s lokálnym úložiskom. Funkcie tohoto servisného objektu umožňujú ukladať dáta o sekciách a označených binoch v správnom tvare rovnako, ako aj načítanie týchto dát z lokálneho úložiska. Hlavná myšlienka tohoto konceptu je hlavne umožnenie ukladať informácie o aktuálnom stave pri analyzovaní aby bolo možné vrátiť sa späť do sekcie z ktorej sme skôr prerušili činnosť. Lokálne úložisko zabezpečuje prehliadač klienta. Servis uľahčuje prácu, keďže funkcie sú implementované tak, aby uložili potrebné dáta v správnom tvare a rovnako aj umožnili tieto dáta načítať z lokálneho úložiska. Servis poskytuje aj správu poľa určeného pre označené biny. Tieto biny sú uchovávané pre ďalšie spracovanie v klientskej aplikácií.

3.4 Adresár controllers

Podadresár obsahuje skripty implementujúce funkcie definujúce funkcionalitu ovládania histogramu a binov pomocou rozličných vstupných zariadení. Konkrétne v našom projekte ide o ovládač klávesnice a ovládač oculusu. Skript nedefinuje inštanciu ovládača ako triedu, ale obsahuje separátne globálne funkcie, aby sa predišlo neskôr problému pri volaní funkcie v metóde *addEventListener* kde prístup k atribútom vo vnútri triedy pomocou *this* je problém z hľadiska toho, že v koncepte JavaScriptu a funkcie addEventListener this odkazuje na inštanciu objektu, na ktorom je volaná funkcia *addEventListener* a nie na premennú, ktorú vlastní daná trieda. Z tohoto dôvodu skripty exportujú potrebné funkcie, ktoré sú potom použité v komponentoch a dodávajú entitám interaktivitu.

3.4.1 Ovládač pre spracovanie vstupov z klávesnice

Ovládač obsahuje prevažne funkcie, ktoré spracovávajú stlačené klávesy na klávesnici. Skript obsahuje implementované funkcie, ktoré zabezpečia rozpoznanie stlačenej klávesy a následne sa vykoná požadovaná akcia. Funkcie sa exportujú a použijú na inicializáciu v A-Frame komponente, kde sa pripnú na histogram. Systém pozostáva z dvoch udalostí, ktoré detegujú stlačenie klávesy a jej uvoľnenie, následne sa stlačené klávesy uložia do poľa aktívnych kláves a vykonávajú sa funkcie. Je možná aj kombinácia viacerých kláves pre vykonanie špecifickej akcie.

Zdrojový kód 3.1: Príklad inicializačnej funkcie A-Frame komponentu, a zaregistrovanie funkcií pre spracovanie vstupov z klávesnice.

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

```
    const el = this.el

    // initialize controller
    initializeKeyboardController(this.schema, el)
    // keyboard listeners
    document.addEventListener('keydown', keyPressHandlerFunction)
    document.addEventListener('keyup', keyReleaseHandlerFunction)
  },
```

V ukážke implementácie 3.1 je zobrazené spracovanie vstupu. Ako prvý krok je inicializácia ovládača funkciou *initializeKeyboardController*(*this.schema, el*), ktorá inicializuje ovládač poskytnutím referencie na element histogramu a údajmi o aktuálnej polohe, veľkosti a rotácií histogramu. Následne je dôležitá funkcia vykonaná po stlačení klávesy a funkcia vykonávaná po uvoľnení klávesy. Funkcia reagujúca na stlačenie klávesy zistí hodnotu stlačenej klávesy, porovná aj ďalšie aktívne klávesy a následne zavolá ďalšiu funkciu implementovanú v ovládači. Príklad časti funkcie *keyPressHandlerFunction* 3.2.

Zdrojový kód 3.2: Implementácia funkcie *keyPressHandlerFunction*. Zobrazená je len časť funkcie.

```
const keyPressHandlerFunction = (event) => {
      // add pressed key to array of the active keys
      keyPressed[event.key] = true

      // call function on press z key
      if (keyPressed.Z || keyPressed.z) {
        handlePositioning(geoAttributes[0], event)
      }
      if (keyPressed.X || keyPressed.x) {
        handlePositioning(geoAttributes[1], event)
      }
      if (keyPressed.C || keyPressed.c) {
    handlePositioning(geoAttributes[2], event)
      }
      if (keyPressed.Shift) {
    handleChangeHistogramSectionByOwnRange(event)
  }
}
```

3.4.2 Ovládač pre spracovanie vstupov z ovládačov zariadenia oculus

Ovládač taktiež obsahuje implementované funkcie podobne ako ovládač pre klávesnicu. Rozdiel je v kvantite funkcií keďže v prípade pripnutia funkcií na histogram je potrebné nastaviť viac listenerov pre udalosti získavané z vstupného ovládača zariadenia Oculus. Každý listener použije špecifickú funkciu pre určitú vstupnú udalosť. Priradenie funkcií k akciám prebieha odlišne ako v prípade ovládača pre klávesnicu. Ako prvé je potrebné vytvorit A-Frame entity, ktoré budú reprezentovať ruky používateľa vo VR a priradiť im komponent, ktorý umožní entite rozpoznať špeciálne vstupné akcie, ktoré sa zadávajú len použitím vstupného ovládača zariadenia Oculus.

Zdrojový kód 3.3: Príklad funkcie ovládača pre oculus. Funkcia rozpozná pozíciu smer posunutia páčky a následne sa vykoná funkcia pre zmenu pozície kamery s požadovanými parametrami

```
const thumbStickPredefinedPosition = (event) => {
      if (event.detail.y > 0.95) {
              cameraService.setPredefinedUpPosition()
      }
      if (event.detail.y < -0.95) {
              cameraService.setPredefinedDownPosition()
      }
      if (event.detail.x < -0.95) {
              cameraService.setPredefinedLeftPosition()
      }
      if (event.detail.x > 0.95) {
              cameraService.setPredefinedRightPosition()
      }
}
```

Zdrojový kód 3.3 popisuje príklad takejto funkcie pričom na registráciu pre vstupné zariadenie je potrebné zabezpečiť aby sa funkcia vykonala vždy pri určitom signále zo vstupného zariadenia. Tento princíp je demonštrovaný v zdrojovom kóde 3.4, ktorý reprezentuje entitu ovládača a pri zachytení vstupného signálu vykonáva požadované funkcie.

Zdrojový kód 3.4: Spracovanie vstupu z používateľského vstupného zariadenia a vykonanie funkcie.

```
el.addEventListener('thumbstickmoved', (event) => {
      if (gripActive) {
```

```
      oculusThumbStickPredefinedCameraPositionWithOffset(event)
        } else {
      oculusThumbStickForMoving(event, speed)
        }
  })
```

V zdrojovom kóde 3.4 je zobrazené zachytávanie signálu pohybu páčky na ovládači.

3.5 Adresár observables

Podadresár obsahuje subjekty knižnice RxJS slúžiace na komunikáciu medzi rôznymi vrstvami v projekte. Subjekt obsahuje funkcie, ktoré publikujú dáta alebo signály, rovnako aj funkciu na získanie inštancie subjektu v komponente, ktorý sa následne prihlási na odber dát.

Zdrojový kód 3.5: Subjekt slúži na zaslanie signálov a dát React componentu, ktorý na ne následne reaguje.

```
class HistogramSubject {
  #subject
  constructor() {
    this.#subject = new Subject()
  }
  // publish signal to change histogram section
  changeHistogramSectionByOffset(
  axis,
  histogramType,
  increment,
  defaultRange) {
    this.#subject.next({
      name: histogramType,
      axis: axis,
      increment: increment,
      defaultRange: defaultRange
    })
  }
  // publish signal to change histogram display mode
  changeHistogramFunction(typeFunction, histogramType) {
    this.#subject.next({
      fFunction: typeFunction,
```

```
      name: histogramType
    })
  }
  // get observable instance for subscribing
  getChangedSection() {
    return this.#subject.asObservable()
  }
}
```

Príklad subjektu 3.5 obsahuje funkcie **changeHistogramSectionByOffset** a **changeHistogramFunction** určené na publikáciu signálov a dát cieľovým komponentom. Funkcia **getChangedSection** predstavuje inštanciu, na ktorej je možné vytvoriť komunikačný kanál a následne získavať publikované dáta. Cieľový komponent zavolá funkciu *subscribe* na inštancií a vykoná funkciu, ktorá sa zavolá vždy po prijatí dát. Funkcia tieto prijaté dáta získava ako parameter.

3.6 Adresár resources

Podadresár obsahuje zdroje, ktoré sú využívané pri tvorbe histogramov, ako sú farebné témy, textúry a podobne. Rovnako obsahuje aj triedu ThemeProvider, ktorá distribuuje tieto farebné palety priamo do objektu, ktorý histogram vytvára. Farebné palety sú uložené v poli objektov v súbore **theme.js**. Pole je exportované a provider podľa názvu farebnej témy vyberá objekt (Obr. 3.5) so všetkými farbami a v ďalšej fáze sa podieľa na dotváranie vzhľadu entít.

Obr. 3.5: Názorná ukážka objektu obsahujúceho informácie o palete farieb s názvom fire, ktoré sa použijú na zafarbenie binov, a ostatných elementov v histograme. (vývojové prostredie WebStorm 2019.3.5)

3.7 Utils

Najdôležitejši podadresár z hľadiska tvorby histogramu obsahuje objekt, ktorý generuje entity binov podľa vstupných parametrov z externého komponentu. Obsahuje súbor **histogramReactFactory.js**, ktorý obsahuje implementáciu triedy. Inštancia sa môže považovať za generátor entít, ktoré vytvoria histogram.

3.7.1 Trieda HistogramReactFactory

Najdôležitejšia trieda v celej implementácií. Inštancia tejto triedy obsahuje všetky potrebné atribúty pre vytvorenie histogramu, ako sú informácie o aktuálnych sekciách a limitoch, ktoré sú taktiež dôležité z hľadiska správneho určenia hraníc histogramu. Obsahuje aj pomocné atribúty, ktoré sa podieľajú na tvorbe animácií alebo na vytváranie špeciálnych zobrazení, ako napríklad atribúty *shift*, ktorý určuje koeficient posunu, dôležitý pre správne nastavenie smeru posunu entít v animácií a rovnako aj atribút *previousBins*, obsahujúci všetky predošlé rozmery binov aby sme zabezpečili pri zmene entít animáciu rastu binu z predchádzajúceho rozmeru na nový rozmer.
Dôležitým atribútom je aj inštancia *themeProvider*, ktorá obsahuje aktuálnu fa-

rebnú paletu a je taktiež dôležitá pre tvorbu entít. Všetky spomínané atribúty, môžme nazývať, ako stav objektu.
Objekt vygeneruje všetky potrebné entity na základe svojho stavu, ktoré potom React komponent vyrenderuje funkciou *render*.
Pre obmieňanie sekcií a vzhľadov histogramu sú implementované funkcie, ktoré modifikujú tento stav, ako napríklad funkcia *updateSection*, zabezpečujúca obmenu informácií o aktuálnej zobrazovanej sekcií a aj funkcia *initializeFactory* umožňujúca nastavenie viacerých atribútov naraz, ako napríklad farebnú tému, histogram, projekcie, identifikátor a ďalšie.
Najdôležitejšie funkcie tohoto objektu sú *createTH3Histogram* a *createTH2Histogram*, ktoré na základe stavu vytvoria všetky potrebné entity.

Zdrojový kód 3.6: Ukžka výtvorenie entity binu použitím JSX.

```
 elements.bins[count] = (
   <a-box
     key={id}
     class='clickable'
     binth2={binData}
     mouseevent
     animation={`property: scale; from: ${
width} ${this.#getPreviousBinScales(id)} ${
depth}; to: ${width} ${c} ${
depth}; delay:100; dur: 1500; easing: linear;`}
     animation__1='property: material.opacity;
from: 0; to: 0.75; dur: 2000;'
     animation__2={`property: position; from: ${
posX + this.#shift.xOffset} ${centeredYPosition} ${
posY + this.#shift.yOffset}; to: ${posX} ${
centeredYPosition} ${posY}; dur: 1000;`}
     animation__3={`property: material.color; from: ${
this.#getPreviousBinColor(id)}; to: ${
markedColor}; dur: 2600;`}
    />
 )
```

Príklad vytvorenia binu 3.6 obsahuje vytvorenie entity so všetkými potrebnými komponentmi, ktoré sú nevyhnutné pre správne fungovanie a správanie binu v histograme. Komponenty ako **binth2**, **binth3**, **mouseevent** dodávajú špecifickú funkcionalitu entite, keďže ide o komponenty A-Frame vrstvy, implementované v adresári aframeComponents. Komponenty **animation** určujú fyzický vzhľad a

pozíciu entity rovnako aj pekný vizuálny spôsob inicializácie v scéne. Pre vkladanie hodnôt do komponentov sú použité atribúty objektu, pomocné funkcie *centeredYPosition*, *getPreviousBinColor* a premenné *id*, *c*, *width*, *depth*, *marked*, *posX*, *posY* obsahujúce už overené a upravené hodnoty.

3.8 Adresár components

V predchádzajúcich sekciách boli popísané triedy a servisy, obsahujúce potrebné funkcionality. Pre konečný efekt je potrebné tieto triedy a funkcie vhodne zlúčiť a vytvoriť tak ucelený komponent, ktorý sa postará o vizualizáciu histogramu a nastaví všetko potrebné pre možnosť interakcie priamo v klientskej aplikácií. Ide o komponenty takzvanej React vrstvy, kde komponenty fungujú na princípoch tohto softvérového rámca a starajú sa o neustálu kontrolu stavu a následnú aktualizáciu DOM-u.

3.8.1 Komponent NdmVrHistogram

Komponent zlučuje servisy a triedy a vytvára všetky potrebné entity, ktoré tvoria histogram. Rovnako sa stará o spracovávanie signálov zo vstupných zariadení a prezobrazuje entity podľa aktuálneho stavu.

Zdrojový kód 3.7: Prihlásenie sa k odberu signálov v React komponente.

```
// subscription after rendering histogram
useEffect(() => {
  if (histo.typeName.includes('TH2')) {
    subscription = histogramTH2Service
      .getChangedSection()
      .subscribe(handleSubscription)
  } else if (histo.typeName.includes('TH3')) {
    subscription = histogramTH3Service
      .getChangedSection()
      .subscribe(handleSubscription)
  }
  return () => subscription.unsubscribe()
}, [elements])
```

V našej implementácií sme uprednostnili písanie React komponentov formou funkcií pred triedami. V tomto prístupe k tvorbe komponentov React obsahuje špeciálne funkcie inak nazývané aj ako *hooks*, ktoré sa spúšťajú za istým účelom v

určitých fázach životného cyklu komponentu. Využívali sme *useState* na nastavenie a definíciu stavových premenných a *useEffect*, ako funkciu, ktorá vykoná určitú funkcionalitu po aktualizácií DOM-u. Funkcia *useEffect* 3.7 zabezpečí detekciu signálov od komponentov A-Frame vrstvy a následne funkciou *handleSubscription* vykoná potrebnú modifikáciu stavu komponentu. Funkcia sa vykonáva vždy pri zmene atribútu *elements*.

3.8.2 Komponent NdmVrCamera

Komponent obsahuje vytvorenie a spravovanie kamery v scéne. Dôležité je nastaviť potrebné identifikátory pre entity, s ktorými následne komponent histogramu musí interagovať a používateľ by mohol pri implementácií v svojej aplikácií v tomto smere zlyhať a riešenie by mu nefungovalo tak, ako má. Komponent obsahuje stavy, ktoré používateľovi zobrazia určitý obsah na paneloch. Principiálne, komponent taktiež prijíma signály od komponentov A-Frame vrstvy a následne na nich reaguje zmenami svojho stavu a aktualizáciou DOM-u.

Zdrojový kód 3.8: Príklad funkcie, ktorá sa vykoná po zaznamenaní signálu.

```
const handleSubscription = (data) => {
  if (data.device) {
    if (inputDevice !== data.device) setInputDevice(data.device)
    setShow(!show)
  } else if (data === 'shift') {
    compileRotation()
  } else if (data === 'show') {
    setShowProjection(!showProjection)
  }
}
```

Funkcia 3.8 sa zavolá vždy po prijatí signálu. Prijaté dáta získava, ako parameter *data* a spracuje tieto dáta. Používa aj ďalšie pomocné funkcie, ako *compileRotation* pre nastavenie rotácie zobrazovaných panelov a nastavuje stavové premenné na základe, ktorých sa aktualizuje DOM.

3.8.3 Komponent NdmVrHistogramScene

Komponent spracováva vstupné parametre a vytvára scénu, do ktorej pridáva komponenty **NdmVrCamera** a **NdmVrHistogram**. Komponent je určený na export. V klientskej aplikácií používateľ použije tento komponent, ktorý vytvorí celú scénu

s pozadím, farebnou témou akú si používateľ zvolí správnym zadefinovaním parametrov.

3.8.4 Komponent JsrootHistogram

Komponent nemá nič spoločné s A-Frame ani s iným softvérovým rámcom pre virtuálnu realitu. Účel komponentu je vytvoriť sekciu s elementami, určenými pre zobrazenie projekcií, ktoré sú vytvorené knižnicou JSROOT. V tomto prípade je úlohou zobraziť histogram ako SVG element, tak ako sa táto knižnica používa aj v klasických webových aplikáciách. Komponent obsahuje referenciu na entitu vo VR, na ktorú následne dokáže aktuálny vizuálny stav SVG elementu zobraziť ako textúru (pozri 3.9).

Zdrojový kód 3.9: Funkcia zabezpečí vytvorenie textúry náhľadu SVG elementu a nastaví ju entite vo VR.

```
// create the histogram projection
jsrootService.createTH1Projection(
  axis,
     { obj: histogram, bin: data },
     'projectionContainer',
     projectionAxes
)
// draw the current SVG view on the entity
setTimeout(() => {
  jsrootService.displayImageOfProjection(
     // id of SVG element
     'projectionContainer',
     // id of entity in VR scene
     'th-mapping',
     // width of texture
     '500px',
     // height of texture
     '500px'
  )
}, 500)
```

4 Testovanie riešenia

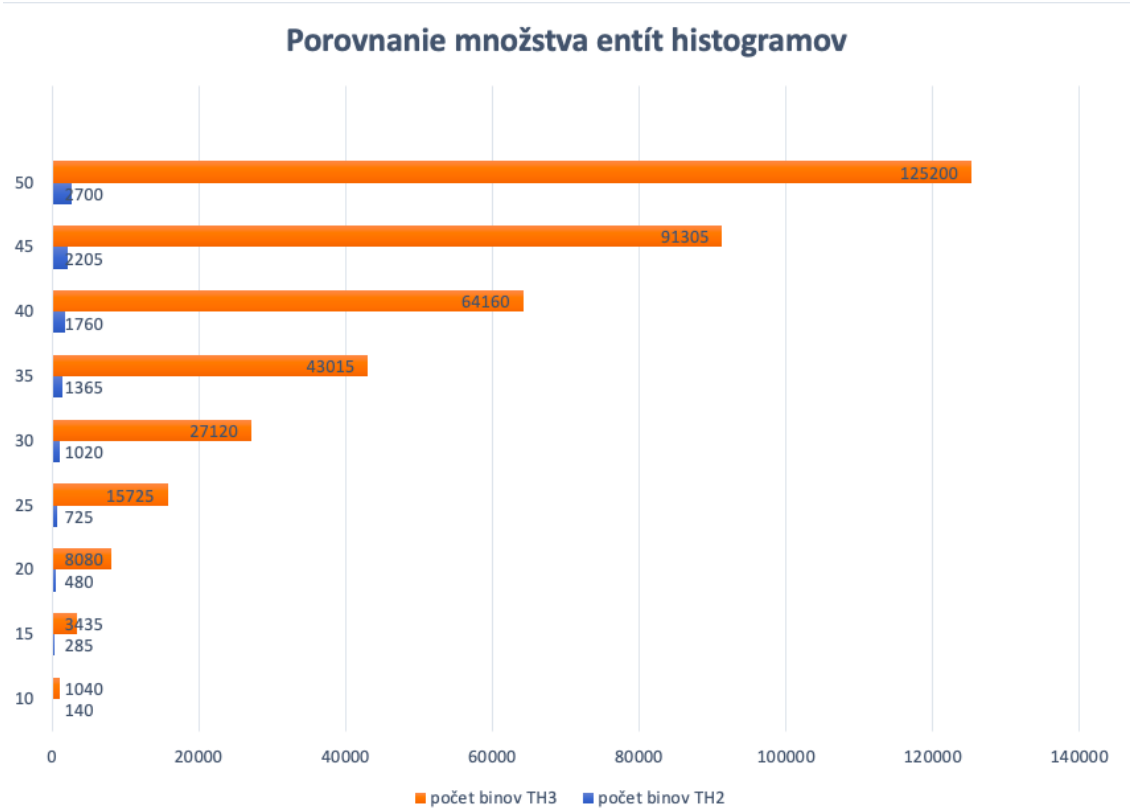
Testovanie je veľmi dôležitá súčasť každého vývoja produktu. Vyvinutý produkt musí spĺňať určitú kvalitu a musí byť aj dostatočne spoľahlivý. Vyvíjané riešenie predstavuje sadu univerzálnych komponentov, ktoré bude možné integrovať do klientskej aplikácie. Z tohto dôvodu je preto nutné zabezpečiť hlavne dobrú integritu, výkon a jednoduchú použiteľnosť. Pri určení kvality riešenia sme brali do úvahy hlavne výkon a použiteľnosť.

4.1 Výkon

Výkon patrí k veľmi dôležitým možno aj najdôležitejším aspektom, ktoré musí riešenie spĺňať. Z ekonomického hľadiska je používateľská skúsenosť najdôležitejšia na to aby riešenie bolo predané a používané zákazníkmi, avšak v tomto prípade je používateľská skúsenosť priamo závislá na dobrom výkone. V prípade zlého výkonu je len veľmi malá pravdepodobnosť, že produkt bude mať dobrú používateľskú skúsenosť. Medzi aspekty ovplyvňujúce výkon patrí hlavne množstvo entít v scéne, hardvér klientskeho zariadenia. V nasledujúcich častiach charakterizujeme jednotlivé aspekty podrobne.

4.1.1 Počet entít

Prvým faktorom ovplyvňujúcim výkon zobrazovania je množstvo jednotlivých entít, ktoré je potrebné zobraziť. Entity reprezentujú biny a rovnako aj všetky ostatné objekty. Záleží na type histogramu, kde TH2 histogram dokáže zobraziť viac binov v jednej sekcií vzhľadom na to, že výpočet všetkých binov pri rozsahu n je n^2. V prípade histogramu typu TH3 je to až n^3. K tomu ešte potrebujeme pripočítať počet entít definujúcich označenia osí, ten je pre oba typy rovnaký a to $4 * n$.



Obr. 4.1: Graf znázorňujúci rozdiel počtu entít oboch typov histogramov pri rovnakom rozsahu.

Z grafu 4.1 je zjavný nepomer a rapídny nárast počtu entít v prípade typu TH3 a preto v prípade pevného rozsahu by komponent bol neefektívny na použitie. Dôležité je spomenúť, že zobrazené údaje 4.1 predstavujú najmenej optimálne riešenie, najhorší prípad kde v histograme majú všetky biny obsah. Pri vysokom počte binov tak prirodzene vzniká problém pri vykresľovaní a je potrebný výkonnejší hardvér zariadenia. Na elimináciu tohoto problému sme implementovali systém pre zobrazovanie len určitej sekcie histogramu s možnosťou nastavenia rozsahu klientom. Rovnako častokrát v histograme boli biny s obsahom rovným 0 a preto tieto biny nereprezentuje žiadna fyzická entita.

Riešenie eliminuje počet binov na zobrazenie ale vzniká nový problém a to problém pri zmenách sekcií, kde je potrebná schopnosť efektívne aktualizovať potrebné entity novými.

4.1.2 Obmieňanie entít

Pre riešenie efektívnej obmeny entít je potrebné otestovať systém, ktorý umožní vykonanie čo najmenšieho množstva operácií tak, aby došlo k potrebným zmenám entít a zároveň aby si zmeny nevyžiadali príliš vysokú výpočtovú náročnosť.

Vytvorený komponent môžme považovať za efektívny a optimálny hlavne z dôvodu použitia softvérového rámca React, ktorý je stavaný na tvorbu reaktívnych používateľských rozhraní. Softvérový rámec funguje presne na tomto princípe obmieňania DOM-u na základe odlišností s jeho virtuálnym DOM-om. Zmeny nastávajú najskôr vo virtuálnom DOM-e až potom sa na základe odlišností aktualizuje reálny DOM. V tomto prípade sa entity nemažú iba upravujú do potrebného stavu a vykonáva sa minimálny počet operácií čo prispieva k lepšiemu výkonu pri obmene.

4.1.3 Vplyv hardvéru na zobrazovanie entít

Vo svete existuje veľká rôznorodosť zariadení, ktoré dokážu zobrazovať webové aplikácie. Niektoré sú výkonnejšie a iné menej, čo z nášho pohľadu môže spôsobiť problém, pretože nevieme, aké rozmery môže nadobudnúť histogram, určený na zobrazenie a rovnako, akým hardvérom disponuje klient. Pre vykresľovanie objektov v scéne je veľmi dôležitá grafická karta, procesor a veľkosť operačnej pamäte. Problém spočíva v tom, že výkonnejší hardvér zobrazí viac binov v rovnakom čase bez toho aby to malo vplyv na používateľskú skúsenosť. Nedochádza k zdĺhavému prezobrazovaniu binov.

Z tohto hľadiska riešenie ponúka možnosť určiť množstvo binov, ktoré sa klientovi zobrazia. Riešenie je optimalizované na zariadeniach so slabším hardvérom. Používateľ nastaví rozsah zobrazených binov na rozmer, pri ktorom bude manipulácia s histogramom svižná a nebude príliš zaťažovať zariadenie.

Pre otestovanie vplyvu kvality hardvéru na zobrazovanie entít histogramu sme vyskúšali experiment s tromi zariadeniami s odlišnými parametrami. Experiment pozostával zo zobrazenia scény s histogramom TH3 s rozsahom 16 binov čo podľa predošlej štatistiky 4.1 predstavuje zhruba 3500 binov. Ako metriky sme použili čas, ktorý potrebovalo zariadenie na zobrazenie ďalšej sekcie a rovnako aj priemerné FPS.

Tabuľka 4.1: Porovnanie výkonu zariadení pri zobrazovaní entít histogramu TH3 s rozsahom 16 binov

Názov zariadenia	pamäť	Grafická karta	procesor	trvanie zmeny sekcie	priemerné FPS
MacBook Air	8GB DDR3	Intel HD Graphic 6000 1536MB	Intel Core i5 1.8GHz	9 sekúnd	4,09
Lenovo legion Y520	8GB DDR4	NVIDIA GTX 1050	Intel Core i5 7300HQ 2,50GHz	7 sekúnd	6,82
ASUS TUF Gaming A15	16GB DDR4	NVIDIA RTX 2060	AMD Ryzen 72800H 2,90GHz	5,5 sekúnd	9,52

Výsledky experimentu sú zobrazené v tabuľke 4.1, ktorá obsahuje parametre zariadení spolu s časom, ktorý bol potrebný od zadania vstupu pre zmenu sekcie až po aktualizovanú sekciu binov v scéne. Z údajov 4.1 vyplýva, že vyše 3500 binov má problém plynule prezobraziť aj zariadenie s pomerne výkonným hardvérom, a preto je nevyhnutné aby bolo možné nastaviť nižší rozsah pre lepší používateľský zážitok aj na úkor, že bude zobrazená len časť histogramu. Priemerné FPS sme merali nástrojmi pre vývojárov v prehliadači mozilla firefox na všetkých zariadeniach. Rovnako zariadenia boli v stave kedy ich nezaťažovali žiadne spustené programy, ktoré by mohli ovplyvniť toto meranie.

4.2 Používateľské testovanie

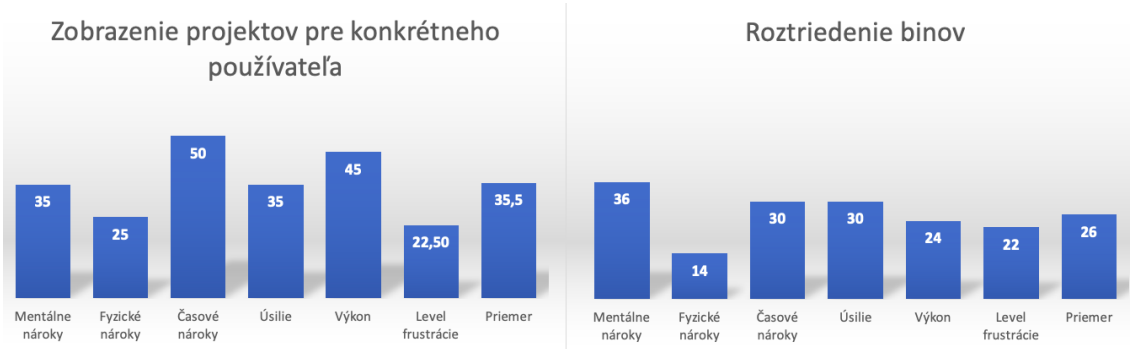
V prípade ak zobrazenie histogramov je zladené s primerane výkonným hardvérom prichádza na rad samotné používateľské testovanie. Riešenie musí poskytnúť používateľovi pocit, že má kontrolu nad histogramom v scéne a rovnako umožniť používateľovi interagovať intuitívne. Na samotné vyhodnotenie používateľskej skúsenosti sme použili vzorku respondentov, ktorí mali za úlohu zoznámiť sa s prostredím XR a následne vykonať jednoduché úlohy. Po vykonaní sme zbierali spätné väzby a typy pre ďalší vývoj. Vzorku respondentov tvorili prevažne štu-

denti, ktorý mali aspoň najmenšiu skúsenosť s dátovou analýzou a boli poučený o okolnostiach súvisiacich s touto vizualizáciou. Je dôležité podotknúť, že riešenie v tejto fáze nepredstavuje hotový produkt určený pre širokú používateľskú skupinu. Preto cieľom bolo otestovať interaktivitu používateľa s histogramom v XR, nie samotnú klientskú aplikáciu, v ktorej boli zobrazené histogramy a získať tak prvú spätnú väzbu a tipy. Pred analýzou výsledkov je potrebné zdôrazniť, že používatelia čelili úplne novému spôsobu analýzy dát. Respondenti boli rozdelení do dvoch skupín podľa skúsenosti s analýzou dát.

Pre vyhodnotenie celkovej použiteľnosti rozhrania sme použili metodiky SUS [39] a NASA TLX [40] na určenie používateľskej pracovnej záťaže pri vykonaní úlohách.

V prípade celkového SUS skóre sme získali 71,88 a 79 čo činí dobrých 75,44. Hranicu pre úspešnosť používateľského riešenia tvorí hodnota 68, ktorú sme prekonali. Skóre definuje priemer a stav kedy je, čo vylepšovať, ale určite používateľské rozhranie nemôžme považovať za zlé. Všetci používatelia testovali vizualizáciu na počítačoch, kde rozsiahlejšia funkcionalita nebola až taká intuitívna, ako by bola v prípade zariadení podporujúcich VR.

Druhý test prebiehal na konkrétnych úlohách, kde sa zisťovala pracovná záťaž používateľov pri vykonávaní úloh spojených s dátovou analýzou. Boli vytvorené 2 úlohy a každá skupina respondentov dostala jednu úlohu. Úlohy a skupiny boli vytvárané prevažne tak, aby každý respondent vykonával úlohu, ktorá mu je vzhľadom na jeho skúsenosti najbližšie. Riešenie otestovalo 9 respondentov, ktorí boli rozdelení do 2 skupín v pomere 4 k 5.



Obr. 4.2: Porovnanie výsledkov testovania úloh

Výsledky pracovného zaťaženia používateľov na jednotlivých úlohách sú zobrazené na obr. 4.2, kde je zaťaženie v rámci oboch skupín. Cieľom je, aby úlohy nespôsobovali nadmerné zaťaženie. Maximálna hodnota bola pre naše meranie 100 a dôležité je, aby zaťaženie malo vo všetkých meraných oblastiach čo najniž-

šiu hodnotu. V prípade výkonu je najnižšia hodnota v grafe považovaná za najlepší výkon. Priemerné hodnotenie neprekročilo hranicu 50 ani v jednej meranej oblasti, čo je určite pozitívne pre naše riešenie, ale opäť je priestor pre hľadanie vylepšení.

Ako zhrnutie testovania je možné vyhlásiť, že riešenie obstálo solídne z hľadiska samotnej použiteľnosti, ale aj estetickej stránky. Používatelia ohodnotili celkový dojem známkou 9,125 z 10 rovnako, ako aj označili, že riešenie určite bude prínosom pre dátovú analýzu.

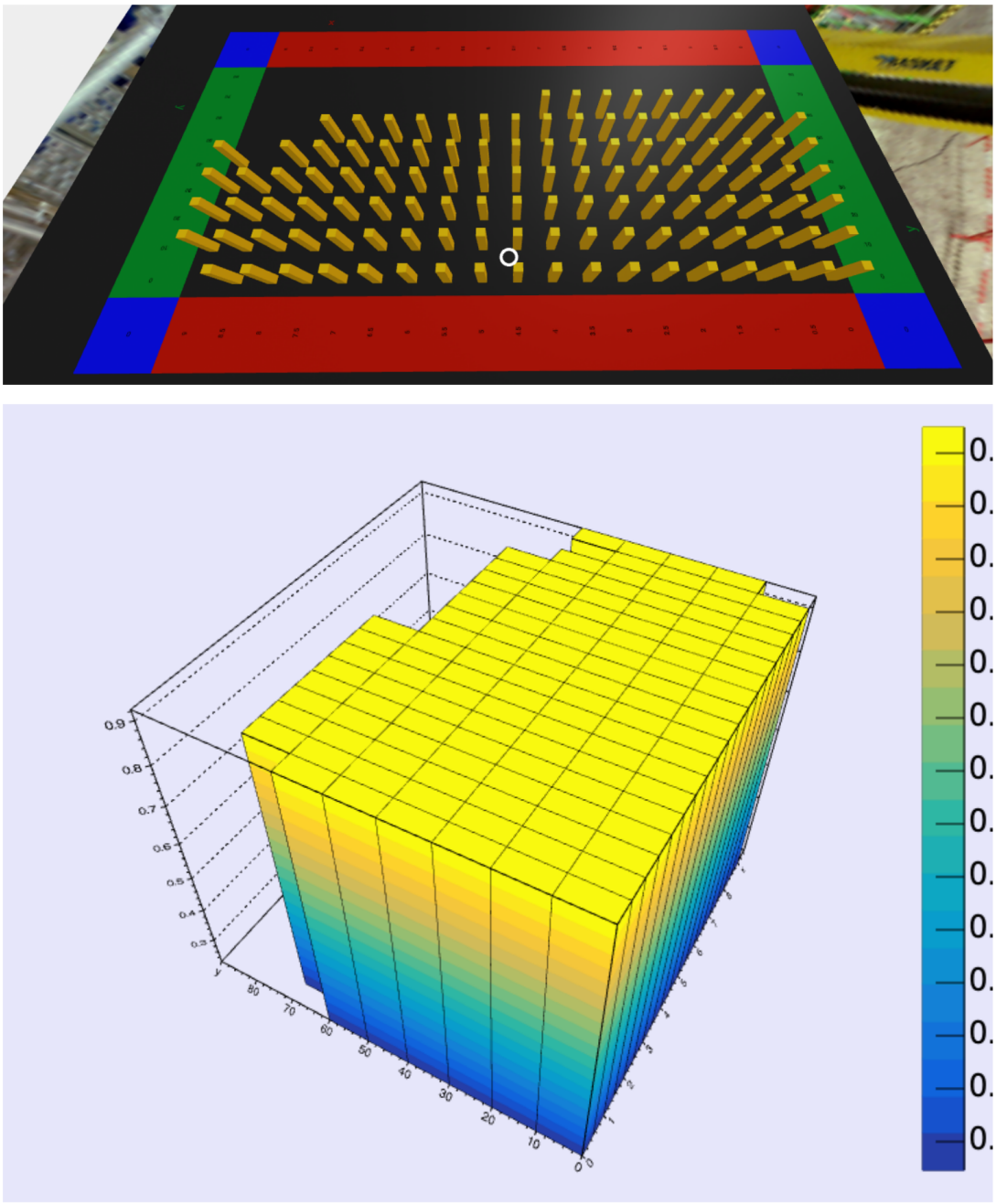
4.3 Vyhodnotenie riešenia a ďalšie možnosti rozšírenia

V tejto kapitole sa pozrieme na celkové vyhodnotenie riešenia, prínosy a ďalšie možnosti vylepšenia v budúcnosti. Pre vytvorenie takejto podoby riešenia sme neraz museli improvizovať a hľadať riešenia, ktoré neboli bežne dostupné v tvare v akom sme ich potrebovali.

Z celkového hľadiska môžeme byť s výsledkom spokojní, keďže komponent je univerzálny a výkonný. Klient si môže podrobne komponent prispôsobiť pre svoju aplikáciu tak ako uzná za vhodné. Rovnako musíme spomenúť aj samotný vývoj, v ktorom sa veľmi prihliadalo na možnosť rozširovania komponentu v budúcnosti.

V porovnaní histogramov na obrázku 4.3 vidíme odlišné prevedenie zobrazení. Naše riešenie reprezentuje väčšie rozostupy medzi binmi čo umožňuje používateľovi nový a zaujímavý pohľad na analýzu, v ktorej je používateľ súčasťou histogramu a vie sa pohybovať medzi binmi ako uzná za vhodné.

Prínos nášho riešenia spočíva aj v možnosti označovania binov, kde vieme kurzorom namieriť na bin a vytvoriť 3 odlišné stavy, čo v bežnom svete je reprezentované len jedným stavom zvyčajne nazývaným ako *hover*. Stavy závisia od prieniku raycastera s hranou entity. Stavy sú ekvivalentné s označením osí a reprezentuje ich hrana, ktorá je rovnobežná s patričnou osou a záleží na tom, ktorú hranu pretne reycaster ako prvú. Tento systém nie je implementovaný vo vizualizácií knižnicou *Three.js* a v našom riešení to umožňujú aj rozostupy medzi binmi.



Obr. 4.3: Vizualizácia rovnakého histogramu. Vyššie je použitie JSROOT a nášho komponentu ndmVr a na obrázku nižšie je histogram vytvorený použitím JSROOT a *Three.js*

Z hľadiska prínosov táto vizualizácia predstavuje začiatok novej etapy, pretože prostredie XR poskytuje nespočetné množstvo spôsobov, ako môžeme zobrazovať dáta a poskytuje nový a zaujímavý spôsob interakcie priamo v prostredí. Riešenie však obsahuje obrovský potenciál pre ďalší vývoj a nasledujúce časti popisujú možnosti pre rozširovanie riešenia.

- **Možnosť použitia vlastných funkcií pre vytváranie projekcií**

Toto rozšírenie umožní poskytnúť funkcie pre vytváranie projekcií podľa používateľa a jeho potrieb. Funkcie by sa mohli zadať komponentu ako parametre a následne na základe údajov o binoch by sa projekcie vytvárali a zobrazovali. Cieľom je aby si každý používateľ zadefinoval svoju vlastnú funkciu a následne aby ju vedel komponent použiť pri vizualizácií.

- **Možnosť spúšťať úlohy na vzdialených klástroch**
  Možnosť interaktívne spúšťať úlohy na klástroch klikom na bin, rovnako aj systém na zobrazovanie údajov o spustených úlohách a ich progrese používateľovi.

- **Možnosť zdieľania scény s viacerými používateľmi**
  Implementovanie zdieľanej virtuálnej reality a vytvorenie kolaboratívneho prostredia pre spoločné konferencie.

- **Vytvorenie komponentu pre vizualizáciu TH1 histogramov**
  TH1 histogram je charakteristický hlavne 2D zobrazením. Implementácia vo virtuálnej realite by bola ideálna na paneli ako 2D objekt ale aj s potrebnou interaktivitou. Aktuálne máme vo virtuálnej realite len textúru projekcie tohto histogramu bez zachovanej interaktivity.

- **Vytvorenie knižnice JSROOT pre virtuálnu realitu**
  Vytvorenie ekvivalentu JSROOT-u pre virtuálnu realitu. Implementácia funkcionality, ktorú poskytuje JSROOT pre vizualizáciu v celom rozsahu. Implementovanie všetkých funkcionalít k nášmu riešeniu tak, aby výsledné vizualizácie obsahovali všetko potrebné, čo obsahuje vizualizácia pomocou *Three.js*

- **Rozšírenie množiny zariadení**
  Keďže komponent na zachovanie potrebnej interaktivity potrebuje podporu pre každé zariadenie v podobe špeciálneho ovládača, pripadá do úvahy rozmýšľať o ďalších ovládačoch. Vytvorenie ďalších ovládačov pre ďalšie vstupné zariadenia a rozšíriť tak komunitu používateľov. V blízkej budúcnosti napríklad pre zobrazovanie a interakciu na smartfónoch, alebo ďalších zariadeniach určených pre XR.

5 Záver

Práca mala za cieľ preniesť vizualizáciu histogramov z pôvodného 3D zobrazenia pomocou knižnice *Three.js* implementovanú v knižnici JSROOT do rozšírenej reality.

Problém nebol prebádaný a preto táto práca mala prevažne experimentálny charakter s nie presne definovanými cieľmi.

Prvoradou úlohou bolo nájsť spôsob, ako efektívne a jednoducho zúžitkovať prepojenie *Three.js* a A-Frame čo sa nepodarilo a jadro riešenia sa podarilo implementovať trochu zložitejším spôsobom, ktorý následne vykazuje väčší potenciál pre ďalší vývoj.

Ruka v ruke s vizualizáciou bolo potrebné aj zachovanie potrebnej interaktivity, ako aj nájdenie systému ako menežovať zobrazovanie dát efektívne a jednoducho. Pre tento účel bolo riešenie spojené so softvérovým rámcom React, kde bolo taktiež potrebné zabezpečiť výmenu dát a signálov medzi React komponentami a A-Frame komponentami. V tomto prípade nám dopomohla knižica RxJS, kde sme už vedeli komponenty prepojiť a vytvoriť tak ucelené riešenie spĺňajúce vizualizačný a interakčný charakter.

Ako posledná fáza vývoja, bolo už len vytvoriť vhodnú architektúru tak, aby sme neimplementovali redundantné funkcie a zabezpečili škálovateľnosť projektu, keďže v priebehu implementácie vyplývalo stále viac možných vylepšení.

Za rok sústavného vývoja a pravidelného konzultovania s RNDr. Martinom Vaľom, PhD sa nám podarilo vytvoriť balík rozšírujúci klientské react aplikácie o komponent pre vizualizáciu histogramov v XR. Nevytvorili sme len samotné komponenty, ktoré vytvoria z objektu histogram tak, ako je to v JSROOT s použitím knižnice *Three.js* ale vytvorili sme predovšetkým nový systém a nadobudli nové poznatky a skúsenosti, o ktoré sa môžeme oprieť aj v ďalších fázach vývoja.

Balík obsahuje potenciál pre pridávanie funkcionalít a predstavuje aj zaujímavú možnosť analyzovať dáta tým, že používateľ je súčasťou histogramu a pre výkonávanie svojho povolania analytika môže vykonávať aj pohybovú činnosť v XR scéne. Rovnako je dôležité doplniť, že virtuálna realita poskytuje lepšie pro-

stredie pre zobrazenie viacdimenzionálnych dát, pridaním textúr, tvarov, pozícií, a udalostí čo je zárukou veľkej budúcnosti v tejto oblasti vizualizácie.

Ako výsledný produkt vznikol publikovaný balík komponentov, ľahko implementovateľný v klientských aplikáciách. Používateľ si vie tento balík nainštalovať pomocou správcu balíkov a následne poskytnutím vhodných parametrov integrovať do svojej aplikácie a patrične ho využívať.

Literatúra

1. SERGEY LINEV, Manraj Singh. JavaScript Root github documentation. [B. r.]. Dostupné tiež z: https://github.com/root-project/jsroot/blob/master/docs/JSROOT.md.
2. MARCOS D. McCurdy D., Ngo K. Introduction - A-Frame. 2020. Dostupné tiež z: https://aframe.io/docs/1.1.0/introduction/.
3. SCIENCE, CERN accelerate. CERN official site. 2020. Dostupné tiež z: https://home.cern/about.
4. SCIENCE, CERN accelerate. LHC experiments. 2020. Dostupné tiež z: https://home.cern/science/experiments.
5. MATIS, Dominik. *Webové rozhranie pre systém SALSA*. 2020. Dipl. pr. Technická unierzita v Košiciach. bakalárska práca.
6. HOSCHEK, Wolfgang; JAEN-MARTINEZ, Javier; SAMAR, Asad; STOCKINGER, Heinz; STOCKINGER, Kurt. *Grid Computing — GRID 2000*. Data Management in an International Data Grid Project. Ed. BUYYA, Rajkumar; BAKER, Mark. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000. ISBN 978-3-540-44444-2.
7. SCOTT, David W. On optimal and data-based histograms. *Biometrika*. 1979, roč. 66, č. 3, s. 605–610.
8. JSROOTUI. *JSROOT user interface*. [B. r.]. Dostupné tiež z: https://root.cern.ch/js/latest/api.htm#url_syntax_open_file.
9. BRUN, René; RADEMAKERS, Fons; PANACEK, Suzanne; ANTCHEVA, Ilka; BUSKULIC, Damir. The ROOT Users Guide. *CERN, http://root. cern. ch*. 2003.
10. PIPARO, D.; QUAST, G.; M.ZEISE. The ROOT Guide For Beginners. *CERN, http://root. cern. ch*. [B. r.].
11. BELLENOT, Bertrand; LINEV, Sergey. ROOT I/O in JavaScript. *Journal of Physics: Conference Series*. 2014, roč. 513, č. 5, s. 052005. Dostupné z DOI: 10.1088/1742-6596/513/5/052005.

12. D3.JS. *JavaScript library for manipulating documents based on data*. [B. r.]. Dostupné tiež z: http://d3js.org/.

13. THREE.JS. *a JavaScript 3D library*. [B. r.]. Dostupné tiež z: http://threejs.org/.

14. BELLENOT, Bertrand; LINEV, Sergey. JavaScript ROOT. *Journal of Physics: Conference Series*. 2015, roč. 664, č. 6, s. 062033. Dostupné z DOI: 10.1088/1742-6596/664/6/062033.

15. JQUERY. *JavaScript library*. [B. r.]. Dostupné tiež z: https://jquery.com/.

16. CERN. *th2 example*. [B. r.]. Dostupné tiež z: https://root.cern.ch/js/latest/api.htm#custom_html_th2.

17. CERN. *th2 srcExample*. [B. r.]. Dostupné tiež z: https://root.cern.ch/js/latest/api.htm#custom_html_th2_src.

18. LINEV, Sergey. *Project TGeo example source code*. [B. r.]. Dostupné tiež z: https://github.com/root-project/jsroot/blob/master/demo/tgeo_build.htm.

19. ITC, Vector. Virtual reality, a new tool to understand big data. *TECH MAGAZIN*. 2019, roč. 1. Dostupné tiež z: https://www.vectoritcgroup.com/en/tech-magazine-en/data-ecosystem-en/virtual-reality-a-new-tool-to-understand-big-data/.

20. BRYSON, Steve. Virtual reality in scientific visualization. *Communications of the ACM*. 1996, roč. 39, č. 5, s. 62–67.

21. BOUD, Andrew C; HANIFF, David J; BABER, Chris; STEINER, SJ. Virtual reality and augmented reality as a training tool for assembly tasks. In: *1999 IEEE International Conference on Information Visualization (Cat. No. PR00210)*. 1999, s. 32–36.

22. WILSON, JR. Virtual environments and ergonomics: needs and opportunities. *Occupational Health and Industrial Medicine*. 1998, roč. 1, č. 38, s. 2–3.

23. DRASCIC, David; MILGRAM, Paul. Perceptual issues in augmented reality. In: *Stereoscopic displays and virtual reality systems III*. 1996, zv. 2653, s. 123–134.

24. ONG, Soh K; NEE, Andrew Yeh Chris. *Virtual and augmented reality applications in manufacturing*. Springer Science & Business Media, 2013.

25. 41, North of. What really is the difference between AR / MR / VR / XR ? *Medium*. 2018, roč. 1. Dostupné tiež z: https://medium.com/@northof41/what-really-is-the-difference-between-ar-mr-vr-xr-35bed1da1a4e.

26. MRDOOB. Three.js. 2020. Dostupné tiež z: `https://threejs.org/`.
27. COMUNITY, A-Frame. *A-Frame Registry*. [B. r.]. Dostupné tiež z: `https://aframe.io/aframe-registry/`.
28. HUDÁK, Marián; KOREČKO, Štefan; SOBOTA, Branislav. Enhancing Team Interaction and Cross-platform Access in Web-based Collaborative Virtual Environments. In: *Proceedings of 2019 IEEE 15th International Scientific Conference on Informatics*. IEEE, 2019, s. 160–164.
29. HUDÁK, Marián; KOREČKO, Štefan; SOBOTA, Branislav. LIRKIS Global Collaborative Virtual Environments: Current State and Utilization Perspective. *Open Computer Science*. 2020 (accepted).
30. HUDÁK, Marián; KOREČKO, Štefan; SOBOTA, Branislav. Advanced User Interaction for Web-based Collaborative Virtual Reality. In: *2020 11th IEEE International Conference on Cognitive Infocommunications (CogInfoCom)*. 2020, s. 343–348.
31. FERRAIOLO, Jon; JUN, Fujisawa; JACKSON, Dean. *Scalable vector graphics (SVG) 1.0 specification*. iuniverse Bloomington, 2000.
32. BRUNT, Paul; BENETOU, Fabien; MASSON, Paul. [B. r.]. Dostupné tiež z: `https://github.com/supereggbert/aframe-htmlembed-component`.
33. INC., Facebook. [B. r.]. Dostupné tiež z: `https://reactjs.org/`.
34. TRONCONE, Brian. rx.js. 2021. Dostupné tiež z: `https://www.learnrxjs.io/`.
35. INC., Facebook. [B. r.]. Dostupné tiež z: `https://github.com/facebookarchive/react-360?fbclid=IwAR1PEkf8-O2AmH07Hd5K6L2zzvxv0BEE5NSpsfQN2ioJDNxEu5BcVDuU5YM`.
36. JORDAN, Papaleo; SOPHIA, Shoemaker. React and WebVR using A-Frame. 2017. Dostupné tiež z: `https://www.newline.co/fullstack-react/articles/react-and-webvr-using-aframe/`.
37. PITIPON, Pluemworasawat. *A-frame with Angular : Setup Project*. [B. r.]. Dostupné tiež z: `https://medium.com/@pitipon/a-frame-with-angular-setup-project-5797b2f2a03b`.
38. FREDERIC, Schwarz; FIDELTHOMET. *aframe-vuejs-3dio*. [B. r.]. Dostupné tiež z: `https://github.com/frederic-schwarz/aframe-vuejs-3dio/blob/master/index.html`.

39. [B. r.]. Dostupné tiež z: https://www.trymyui.com/sus-system-usability-scale.

40. [B. r.]. Dostupné tiež z: https://ext.eurocontrol.int/ehp/?q=node/1583.

Zoznam skratiek

1D Jednorozmerný.

2D Dvojrozmerný.

3D Trojrozmerný.

API Application Programming Interface.

AR Augmented Reality.

CSS Cascading Style Sheets.

DOM Document Object Model.

ECS Entity Component System.

FEI Fakulta Elektrotechniky a Informatiky.

FPS frames per second.

G-CVE Global Colaborative Virtual Environments – globálne kolaboratívne virtuálne prostredia.

HTML Hyper Text Markup Language.

JSON Java Script Object Notation.

JSX JavaScript XML.

KPI Katedra počítačov a informatiky.

LIRKIS Laboratórium Inteligentných Rozhraní Komunikačných a Informačných systémov.

MR Mixed Reality.

SVG Scalable Vector Graphic.

TLX task load index.

URL Uniform Resource Locator.

VR Virtual Reality.

XML eXtensible Markup Language.

XR Extended Reality.

Zoznam príloh

Príloha A CD médium – záverečná práca v elektronickej podobe.

Príloha B Používateľská príručka

Príloha C Systémová príručka

Adresár *src* v **prílohe A** obsahuje projekt **ndmvr** s všetkými potrebnými zdrojovými súbormi. Okrem projektu **ndmvr** obsahuje aj adresár *experimentResults* s projektmi, ktoré slúžili na experimentálne overenie riešení počas analýzy. Tento adresár obsahuje 2 projekty na ktorých sa experimentovalo.

Adresár *userFeedback* v **prílohe A** obsahujuje anonymizované výsledky používateľského testovania.

Adresár *lib* v **prílohe A** obsahuje súbory dokumentácie zdrojového kódu projektu. Súbory vygenerované pomocou **jsdoc**.