Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Simulátor geometrie 3D fotografování

Plzeň, 2018 Matěj Berka

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2017/2018

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Matěj BERKA**
Osobní číslo:	**A16N0005K**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Softwarové inženýrství**
Název tématu:	**Simulátor geometrie 3-D fotografování**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování:

1. Seznamte se s problematikou 3-D fotografování a s tím, jaká rozhodnutí musí fotograf před snímáním učinit.
2. Prozkoumejte simulátor geometrie 3-D fotografování "stereoSim" a identifikujte jeho slabiny, zejména z pohledu uživatelského rozhraní.
3. Identifikujte případy použití (use cases) a navrhněte sadu úloh, na níž se bude testovat nový simulátor.
4. Navrhněte a implementujte uživatelské rozhraní simulátoru, které bude uživatelsky příjemnější než rozhraní programu "stereoSim"
5. Navrhněte a implementujte jádro simulátoru.
6. Implementovaný simulátor prakticky otestujte.

Rozsah grafických prací: **dle potřeby**

Rozsah kvalifikační práce: **doporuč. 50 s. původního textu**

Forma zpracování diplomové práce: **tištěná**

Seznam odborné literatury:

dodá vedoucí diplomové práce

Vedoucí diplomové práce: **Ing. Petr Lobaz**
Katedra informatiky a výpočetní techniky

Datum zadání diplomové práce: **1. září 2017**

Termín odevzdání diplomové práce: **17. května 2018**

Radová

Doc. Dr. Ing. Vlasta Radová
děkanka

L.S.

Západočeská univerzita v Plzni * Fakulta aplikovaných věd *

Brada

Doc. Ing. Přemysl Brada, MSc. Ph.D.
vedoucí katedry

V Plzni dne 14. září 2017

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 10. května 2018

Matěj Berka

Abstract

In this master's thesis was created a new version of education program used in classes focused on 3D photography. The new version is based on knowledge from previous version and removes some of its drawbacks. The most significant changes have been made in graphical user interface. The program has also been expanded with new features that will help during the study of 3D photography. The program is written in programing language JavaScript and new version offers support for mobile devices. Mobile devices can also use the program as an „native“ application. The program has an new architecture which adds easy extensibility and only technologies with promising future were picked for the new version.

Abstrakt

V této diplomové práci vznikla nová verze výukového programu, který se používá při výuce 3D fotografování. Nová verze vychází ze zkušeností předchozí verze a odstraňuje některé jejich nedostatky. Nejvýraznější změny byly provedeny u grafického uživatelského rozhraní. Do programu bylo dále přidáno několik nových funkcí, které pomohou při výuce 3D fotografování. Program je vyvíjený v programovacím jazyce JavaScript a nově je možné aplikaci používat i jako „nativní“ aplikaci na tabletech. Program má kompletně novou architekturu, která zaručí snadnou rozšiřitelnost v budoucnosti. K realizaci nové verze programu byly vybrány moderní technologie, které by měly mít slibnou budoucnost.

Poděkování

Tímto bych chtěl poděkovat panu Ing. Petru Lobazovi, Ph.D. za jeho pomoc a rady při tvorbě diplomové práce a velice vstřícný přístup.

Obsah

1 Úvod **1**

2 Úvod do 3D fotografování **2**
2.1 Pořízení 3D fotografie 2
2.2 Rozpoznávání hloubky 3
2.3 Mechanismus stereopsie 5
2.4 Konflikt akomodace a vergence 6
2.5 Manipulace s vjemem hloubky 7
2.6 Postprodukce stereoskopických snímků 11

3 Analýza použití simulátoru **14**
3.1 Seznámení s programem 14
3.1.1 Vizualizace 15
3.1.2 Panel konfigurace 16
3.1.3 Diagnostika 16
3.2 Definice případu užití 16

4 Návrh nové aplikace **19**
4.1 Identifikace nových funkcí a rozšíření simulátoru 19
4.1.1 Nové objekty scény 19
4.1.2 Práce s objekty scény 20
4.1.3 Import/export simulace 21
4.1.4 Výpočet a vizualizace hloubky ostrosti 21
4.1.5 Více simulovaných scén 22
4.1.6 Schránka 22
4.1.7 Práce s pohledy simulace 22
4.2 Orientace v programu 24
4.2.1 Nápověda k prvkům konfigurace 24
4.2.2 Použití ikon 25
4.3 Využití ilustrací 26
4.3.1 Detaily 27
4.4 Návrh základního rozložení GUI 27
4.4.1 Umístění panelu s konfigurací 28
4.5 Návrh architektury 31
4.5.1 MVC, MVVM, MVP nebo MVW 32
4.5.2 Rozdělení na komponenty 34

4.5.3 Komunikace mezi komponentami . . . 35
4.5.4 Datový model . . . 36
4.5.5 Přístup k datům . . . 37
4.5.6 Import a export . . . 38
4.5.7 Shrnutí . . . 38
4.6 Výběr nástrojů . . . 38
4.6.1 Výběr frameworku . . . 38
4.6.2 Výběr syntaxe . . . 40
4.6.3 Výběr UI knihovny . . . 42

5 Implementace **44**
5.1 Adresářová struktura . . . 44
5.2 Struktura aplikace . . . 45
5.3 Použití knihovny React . . . 47
5.4 Tok programu . . . 48
5.4.1 Generované události . . . 49
5.5 Import a export . . . 50
5.6 Rozšiřitelnost simulátoru . . . 51
5.6.1 Přidání nového objektu scény . . . 51
5.6.2 Přidání nového konfiguračního panelu . . . 51
5.6.3 Přidání nového pohledu . . . 53
5.7 Implementace distribuce událostí . . . 53
5.8 Implementace vzoru Service Locator . . . 54
5.9 Nové datové typy . . . 54
5.10 Implementace vykreslování vizualizací . . . 55
5.11 Progressive Web Apps . . . 55
5.12 Sestavení . . . 56
5.13 Podpora webových prohlížečů . . . 57
5.13.1 Výkonnostní rozdíly prohlížečů . . . 57
5.14 Podpora mobilních zařízení . . . 58

6 Praktické otestování **59**
6.1 Výsledky testování . . . 60

7 Zhodnocení a závěr práce **63**

A Programátorská dokumentace **67**
A.1 Proces vývoje . . . 67
A.2 Proces nasazení . . . 69

B Seznam návrhů dalších úprav programu **70**

C Použité zkratky **72**

1 Úvod

Cílem práce je vytvořit novou verzi výukového programu, který se používá při výuce 3D fotografování. Stávající verzi programu vytvořil vedoucí mé práce, který také předmět o 3D fotografování vyučuje (předmět Fakulty designu a umění Ladislava Sutnara KVU/UF3D 3D fotografie a alternativní techniky ve fotografii). Program používají především studenti, kterým pomáhá pochopit probíranou látku (především když se učí techniku stereoskopie). Kromě studentů ale program používají i zkušení fotografové, kteří jej používají k návrhu parametrů 3D snímání.

U stávající verze programu je nutné nejprve identifikovat její slabiny a navrhnout sadu vhodných úprav, které povedou k jejich odstranění. Úpravy jsou cílené především na uživatelské rozhraní a práci se simulátorem (možnosti interakce). Současná verze výukového programu je realizována jako webová aplikace a i nová verze bude realizována jako webová aplikace. Program je tak snadno dostupný na všech platformách a není nutné nic stahovat a instalovat. Do nové verze bude také přidána podpora mobilních zařízení a navržena nová architektura aplikace, která zajistí snadnou rozšiřitelnost programu do budoucna a lepší interní logiku.

K identifikaci slabin stávajícího programu a návrhu vylepšení je nejprve nutné pochopit princip 3D fotografování a čím může program fotografovi prospět. Kapitola 2 se proto věnuje geometrické podstatě 3D fotografie, tj. vztahem mezi technikou fotografování a výsledným 3D vjemem. Kapitola 3 následně popisuje stávající verzi programu a případy jeho užití. V kapitole 4 následuje návrh nových funkcí programu a nové podoby grafického uživatelského rozhraní. S ohledem na stávající implementaci programu také probírám a porovnám několik možností realizace nové verze a vytvářím architekturu nové verze. Pro navrženou architekturu vybírám vhodné technologie a odůvodňuji jejich výběr. Navržená architektura a vybrané technologie jsou následně použity k implementaci nové verze programu a implementace je popsána v kapitole 5. V této kapitole jsou také uvedeny možnosti rozšiřování programu, způsob sestavení programu a podpora webovými prohlížeči. U implementovaného programu v kapitole 6 provádím praktické otestování, kde ověřuji, jak jsou uživatelé s novou verzí programu spokojeni. Uživatele při práci se simulátorem pozoruji a zaznamenávám si jejich aktivitu. Následně provádím vyhodnocení testování a sepisuji návrh dalších úprav simulátoru pro budoucí verze. V závěru pak provádím zhodnocení celé práce.

2 Úvod do 3D fotografování

Při studiu problematiky 3D fotografování jsem používal především výukové materiály (zdroj [4]), které jsou používané právě při výuce 3D fotografování společně se zmiňovaným simulátorem na Fakultě designu a umění Ladislava Sutnara Západočeské univerzity v Plzni. Také je to jediný zdroj informací, který s největší pravděpodobností použije většina studentů, kteří se simulátorem pracují. Právě tento fakt mi poskytl jedinečnou příležitost získat stejný rozsah znalostí jako studenti předmětu.

2.1 Pořízení 3D fotografie

Abych získal představu o tom, jak u 3D fotografie vzniká prostorový efekt, tak jsem se nejprve seznamoval se základním postupem tvorby 3D fotografie.

Člověk vnímá svět dvěma očima, tj. každé oko vidí trochu jiný obraz. Pokud chceme vytvořit 3D iluzi, tak musíme zařídit aby každé oko vidělo trochu jiný snímek. Takovému páru snímků se říká **stereoskopický snímek** a vjemu hloubky, který tyto dva snímky vytvářejí **stereopsie**. Příklad takové fotografie je vidět na následujícím obrázku. Je vidět, že rozdíly v obou fotografiích nejsou příliš velké. Stereoskopický snímek obvykle fotografujeme dvěma fotoaparáty, které jsou umístěny kousek vedle sebe.

Obrázek 2.1: Ukázka stereoskopického snímku (převzato z [12]).

Ne každý pár fotografií však vytvoří správný vjem hloubky. Odlišnosti mezi fotografiemi musí být tak velké, aby vjem vznikl. Současně ale musí

být odlišnosti tak malé, aby mozek vyhodnotil snímky jako dva pohledy na tutéž scénu. I pokud vynakládáme velkou snahu o korektní pořízení snímků, někteří pozorovatelé 3D vjem neuvidí. Důvodem je, že asi 3 % lidí nevidí stereoskopicky a asi 20 % lidí má se stereoskopií problémy. U zbytku populace je vnímání značně odlišné, co je pro někoho „akceptovatelně velký vjem“, je pro někoho „přehnaný vjem“ (viz zdroj [4]). Je proto vhodné pořízený snímek vždy otestovat na více lidech. Tvorba kvalitních 3D snímků je tedy často velmi pracná. 3D snímky mají dále na rozdíl od „2D“ fotografie tu nevýhodu, že pokud se nepovede vytvořit správný vjem hloubky, tak jeho pozorování může vést i k nevolnosti.

Pokud by rozdíly (vzájemný posun) mezi fotografiemi byly příliš velké, tak by mohlo docházet k tzv. konfliktu akomodace a vergence. Abych tento konflikt mohl vysvětlit, tak je nutné si nejprve představit, jak rozpoznáváme hloubku a mechanizmus stereopsie.

2.2 Rozpoznávání hloubky

Člověk používá k rozpoznává hloubky několik mechanizmů a vodítek:

- osvětlení,
- relativní velikost známých objektů,
- překryv,
- jemnost textury,
- vzdušnou perspektivu,
- lineární perspektivu,
- hloubku ostrosti,
- pohybovou paralaxu,
- binokulární paralaxu,
- vergenci,
- akomodaci.

I při pohledu na 2D fotografii používáme většinu výše zmíněných mechanizmů a jsme schopni zrekonstruovat zachycený prostor. Při pozorování 3D scény používáme ještě **binokulární paralaxu, akomodaci a pohybovou paralaxu**. Ty si nyní krátce představíme.

Pohybová paralaxa

Pojem paralaxa označuje změnu v relativní vnímané pozici objektu. Při pohybu hlavou vnímáme jak moc se změní pozice objektu, který sledujeme. U objektů, které jsou blízko, dochází k výrazné změně. U vzdálených objektů dochází naopak k minimální změně. Tento mechanizmus je možné si snadno ověřit při pohledu do dálky. Uděláme-li mírný pohyb hlavou nebo úkrok stranou, tak vzdálené objekty se téměř nepohnou, kdežto u blízkých zaznamenáme výraznější posun. U jedno-pohledových statických 3D snímků však není možné tento efekt reprodukovat.

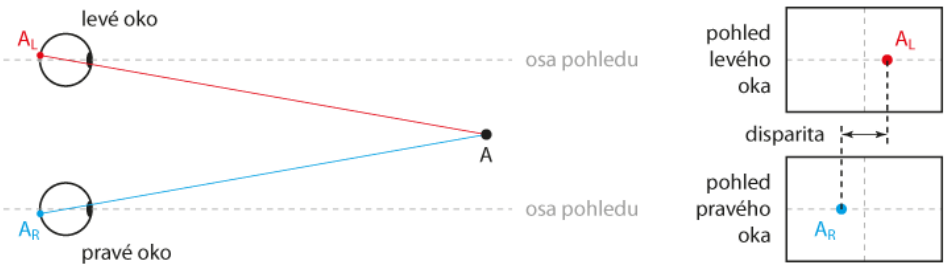
Binokulární paralaxa

Kromě pohybové paralaxy používáme k odhadu vzdálenosti objektu ještě binokulární paralaxu. Opět se jedná o relativní změnu vnímané pozice objektu. V tomto případě, změna mezi levým a pravým okem. Každé oko vidí trochu jinou scénu (svět) a odlišnosti v obrazech používáme k odhadu prostorovosti. Rozdílu v obrazech, které se nám na sítnice promítají, říkáme **disparita**.

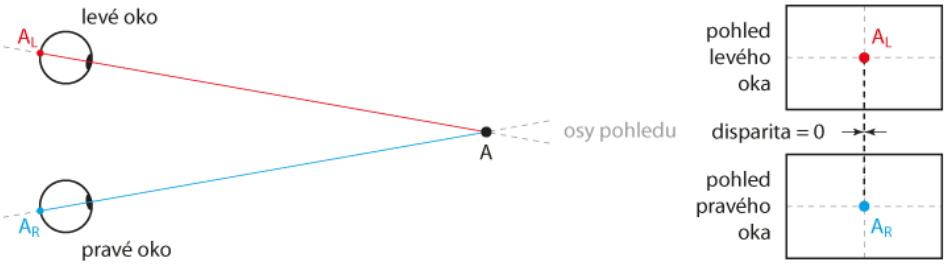
Pojem disparita lze ještě lépe objasnit následujícím způsobem. Představíme si, že máme místo očí dva fotoaparáty a každý z nich pořídí snímek bodu, který „pozorují“. Pořízené snímky položíme přes sebe a když se bod bude nacházet na stejném místě v obou snímcích, tak říkáme, že mají nulovou disparitu. Pokud by se ale pozice bodu v každém snímku mírně lišila, tak říkáme, že bod má nenulovou disparitu. Rozlišujeme přitom mezi horizontální disparitou (vodorovná vzdálenost obrazů bodu) a vertikální disparitou (svislá vzdálenost obrazů bodu). Vznik disparity je také ilustrován na obrázcích 2.2 a 2.3.

Vergence

Každé oko má svou pomyslnou optickou osu. Při uvolněném očním svalstvu obě osy očí směřují rovnoběžně před oči. Chceme-li si detailně prohlédnou nějaký bod a disparita bodu je mozku nepříjemná, tak použijeme oční svaly. Optické osy očí natočíme tak, aby se střetávaly v místě zaostřovaného bodu. Mozek minimalizuje sítnicovou disparitu a může si objekt prohlédnout. Mozek také může odhadnout, jak daleko se objekt nachází, protože ví jak se oči musely natočit. Tomuto mechanizmu se říká vergence. Následující obrázky poskytují ilustraci tohoto procesu.



Obrázek 2.2: Pozorování bodu A při uvolněných vergenčních svalech. Vzniká nepříjemná disparita (převzato z [3]).



Obrázek 2.3: Oči své optické osy natočí a minimalizují disparitu objektu (převzato z [3]).

Akomodace

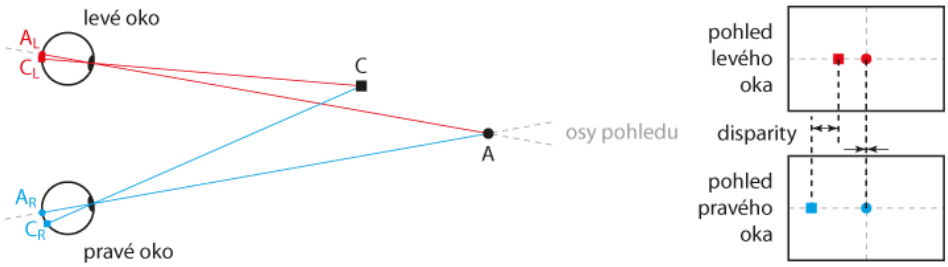
Akomodace je proces, při kterém se upravuje zakřivení oční čočky pomocí akomodačních svalů. Jak víme, ostře vidíme jen v jisté vzdálenosti (oči mají omezenou hloubku ostrosti), mimo ni je obraz více či méně rozostřený. Chceme-li si prohlédnout nějaký předmět, očekáváme, že jej uvidíme ostře. Pokud jej ostře nevidíme, reflexivně se pokusíme změnit akomodaci (zaostření) očí tak, aby ostrý byl. Jelikož mozek ví, jak muselo oko akomodovat, tak je také schopen odhadnout vzdálenost objektu.

2.3 Mechanismus stereopsie

Po představení jednotlivých mechanismů rozpoznávání hloubky se můžeme seznámit s mechanismem stereopsie. Stereopsie popisuje, jak konkrétně funguje rozpoznávání hloubky prostřednictvím binokulárního vidění.

Jak už bylo výše popsáno, vergenci a akomodaci používáme při určování vzdálenosti objektu. Na obrázcích 2.2 a 2.3 bylo ukázáno, jak používáme vergenci ke snížení nepříjemné disparity objektu. Zároveň při tom používáme i akomodaci a na bod zaostřujeme, abychom si ho mohli detailně prohlédnout. Bodový objekt A se při správném zaostření zobrazí na sítnici levého oka jako bod A_L, na sítnici pravého oka jako bod A_R.

Přidáme-li do scény další bod B a jeho obrazy nám budou na sítnicích opět splývat, tak budeme usuzovat, že i bod B se nachází ve stejné vzdálenosti. Pokud by nám ale obrazy nesplývaly, tak mozek usoudí, že objekt se nachází v jiné vzdálenosti. Když jsou polohy obou obrazů dostatečně malé (mají malou nenulovou disparitu), tak si s tím dokáže mozek poradit a bod stále vidí ostře. Pokud by ale rozdíly poloh byly velké, tak by se projevilo dvojité vidění. V tomto případě, pokud by si mozek chtěl bod podrobně prohlédnout, tak by se musel na objekt opět zaměřit (provést akomodaci a vergenci). Na následujícím obrázku je ukázka takového bodu (bod C).



Obrázek 2.4: Nová scéna s bodem C[2], který už má nenulovou disparitu (převzato z [3]).

V praxi provádíme jakési „mapování“ scény (pozorovaného světa). Například, když přijdeme do místnosti, oči začnou těkat po prostoru místnosti a výsledný obrázek prostoru postupně skládají. Tím si pozvolně vytváříme představu o prostoru, ve kterém se nacházíme.

2.4 Konflikt akomodace a vergence

Nyní k samotnému konfliktu akomodace a vergence. Hledíme na 3D snímek (pomocí akomodace a vergence zaostřuje na rovinu snímku), na kterém se nachází bod s tak velkou snímkovou disparitou, že ji nedokážeme

[2]Přísně vzato se takové body (s nenulovou disparitou) na sítnici zobrazují jako rozostřené skvrny.

ignorovat, a u bodu začne docházet k dvojitému vidění. Mozek na to zareaguje vergenčním pohybem a disparitu sníží na přijatelnou úroveň. Tím se reflexivně změní i akomodační vzdálenost, což pravděpodobně povede k tomu, že se rovina snímku dostane mimo hloubku ostrosti a celý snímek uvidíme rozmazaně. To je ale proti naší běžné zkušenosti, kde změna akomodace/vergence vede ke snížení sítnicové disparity a současně se nám zostří i snímek.

Mozek se tedy pokusí opět zaostřit na snímek. Upraví akomodační vzdálenost na původní hodnotu. S tím se reflexivně upraví i vergenční vzdálenosti a dostaneme se zase na začátek. Mozek si v těchto případech musí vybrat jestli se snažit ostřit na bod a vidět snímek rozmazaně a nebo se snažit rozostření ignorovat a vidět bod dvojitě.

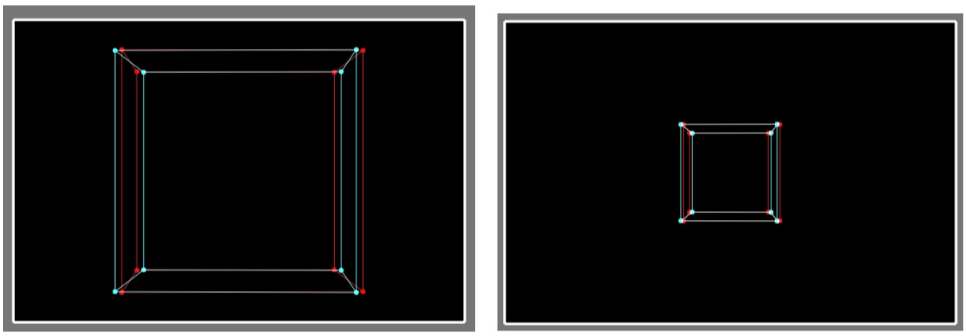
Při tvorbě snímku je proto nutné se tomuto problému vyvarovat a pro fotografa je hlídání maximálních disparit prvořadý úkol. Objekty tedy mohou vystupovat před nebo za plátno pouze v oblasti ostrosti lidského zraku.

2.5 Manipulace s vjemem hloubky

Základem k tvorbě 3D snímku je vytvořit takový pár snímků, u kterého nám nebude vznikat zmiňovaný konflikt akomodace a vergence a 3D efekt bude působit realisticky. Existují přesné postupy, jak snímky vytvořit aby byl 3D efekt co nejvěrnější. V praxi ale často chceme úmyslné potlačovat, zdůrazňovat nebo jinak upravovat výsledný 3D vjem snímků. Takovému snímání se říká kreativní. Když ale chceme upravovat 3D vjem snímku, tak musíme vědět, co zejména má vliv na vjem hloubky obrazu.

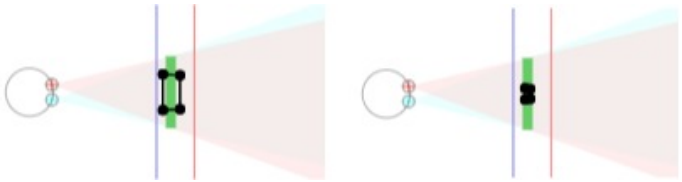
První je vliv ohniskové vzdálenosti. Zvětšení ohniskové vzdálenosti potlačí vjem hloubky perspektivou na jednom snímku, ale pomůže zdůraznit 3D vjem u stereopáru (dojde ke zvětšení disparity). Zmenšení ohniskové vzdálenosti naopak zdánlivě zmenšuje objekty, tedy zmenšuje i disparity a vede k potlačení 3D vjemu vlivem zmenšení disparity. Pro ilustraci jevu ještě přikládám následující obrázky. Pravý snímek byl pořízený s krátkou ohniskovou vzdáleností (18 mm) a levý s delší (35 mm).

Obrázek 2.5: Fotografie pořízené s různými ohniskovými vzdálenostmi (převzato z [5]).



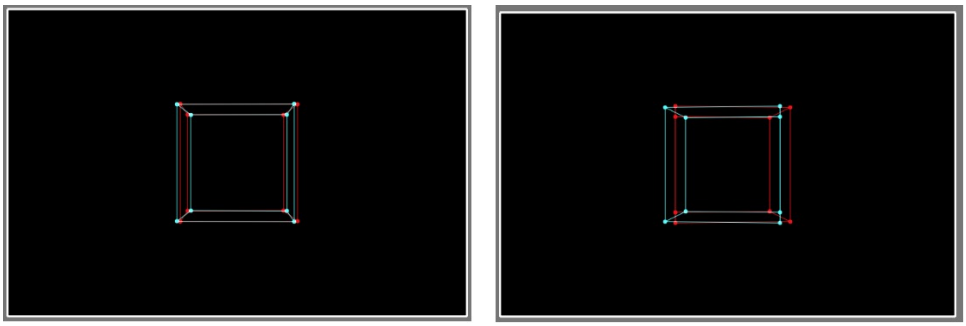
Obrázek 2.6: Ilustrace rozdílu disparit. Pravý snímek byl opět pořízen s velkou ohniskovou vzdáleností a levý s krátkou (převzato ze simulátoru).

Poslední ilustrace (obrázek 2.7) ukazuje, jak změnu vnímá samotný divák. Modrá a červená čára představují komfortní zónu, ve které je pozorování 3D iluze ještě příjemné. Zelená čárá reprezentuje rovinu promítacího zařízení a černý objekt zachycenou scénu (v tomto případě „krychle“).

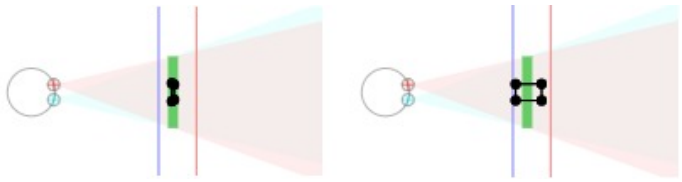


Obrázek 2.7: Ilustrace efektu rozdílu disparit z pohledu diváka. Pravý snímek byl pořízen s velkou ohniskovou vzdáleností a levý s krátkou (převzato ze simulátoru).

Dalším faktorem je vliv vzdálenosti levé od pravé kamery. Velikost této vzdálenosti ovlivňuje náš binokulární 3D vjem. Kromě technických parametrů kamer (stejný objektiv, ohnisková vzdálenost atd.) musejí být kamery umístěny vodorovně ve stejné rovině. Stejně jako tomu je u očí. Vzdálenost mezi kamerami se pak určuje právě v této rovině. Zvyšující se vzdálenost zvyšuje 3D vjem a naopak. Příklad je opět uveden na obrázku 2.8. Levý obrázek ukazuje separaci 63 mm a pravý 200 mm.

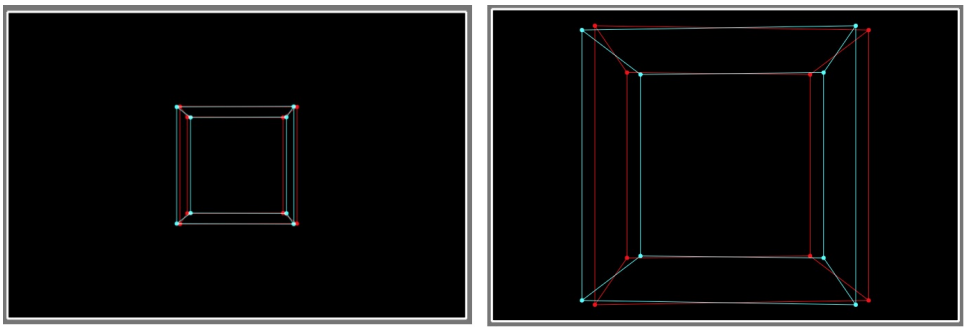


Obrázek 2.8: Ilustrace rozdílu disparit při rozdílné separaci kamer (převzato ze simulátoru).

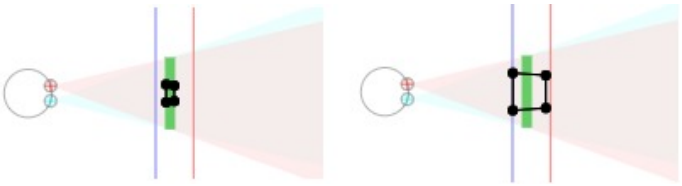


Obrázek 2.9: Ilustrace rozdílu disparit z pohledu diváka (převzato ze simulátoru).

S tím také souvisí vzdálenost kamery od scény. Čím blíže ke scéně (například fotografujeme vázu) kamera je, tím větší 3D vjem bude mít vytvořený snímek. Opět ukázka na obrázku 2.10. Levý obrázek ukazuje vzdálenost kamery 1 m a pravý 0,5 m.



Obrázek 2.10: Ilustrace rozdílu disparit při různých vzdálenostech kamery (převzato ze simulátoru).



Obrázek 2.11: Ilustrace efektu rozdílu disparit z pohledu diváka (převzato ze simulátoru).

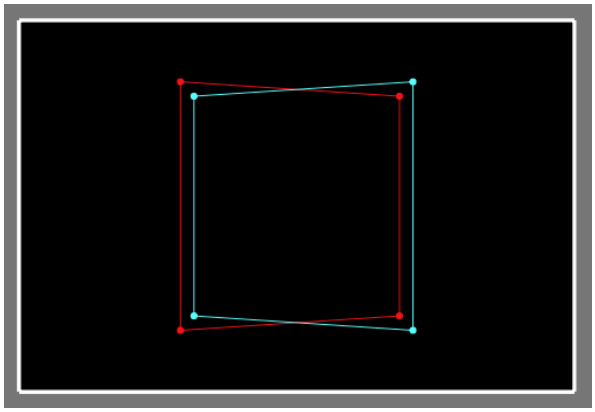
Poslední je vliv displeje a divákovi pozice na 3D vjem. Budeme-li například promítat na plátno velikosti 5 m objekt s disparitou +63 mm (tj. před plátnem), kde 63 mm odpovídá průměrné mezioční vzdálenosti, tak divák, který je ve vzdálenosti 10 m od plátna, uvidí objekt vystupovat 5 m před plátno (poloviční vzdálenost). Zmenšíme-li obraz 10x na 50 cm a divák se přesune do vzdálenosti 1 m (10x menší vzdálenost), tak překvapivě nebude 3D vjem 10x menší (50 cm od displeje), ale jen asi 9 cm. Důvodem je, že mezioční vzdálenost se nemění (zůstává stále 63 mm). Obraz se tedy bude jevit mnohem placatější. Proto 3D obraz, který je připravený pro promítání v kině, bude na 3D televizi vypadat hodně ploše a naopak snímek připravený pro 3D televizi bude v kině vytvářet nepříjemně velkou iluzi hloubky.

2.6 Postprodukce stereoskopických snímků

Právě u 3D snímání jsou užitečné především geometrické korekce. Zde již trochu navážu na samotný simulátor, kde jsou dostupné 4 možnosti korekce a to: *Images Zoom*, *Images Shift*, *Images Stretch* a *Images Keystone*. U všech korekcí je potřeba dát pozor především na výslednou disparitu stereopáru. Ve většině případů chceme, aby vertikální disparita prakticky neexistovala (resp. byla menší než 1 stupeň zorného úhlu) a u horizontální disparity nevznikal silný konflikt mezi akomodací a vergencí.

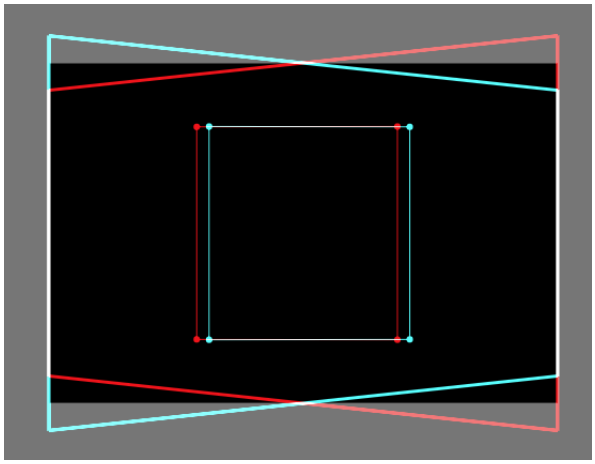
Typickým příkladem použití těchto korekcí je oprava perspektivního zkreslení. Víme, že čtverec fotografovaný běžným fotoaparátem, který je zachycený i jen mírně ze strany, vypadá jako lichoběžník. A pokud vytvoříme takový stereopár, kde jsou oba snímky mírně ze strany, tak dostaneme dva lichoběžníky (vznikne vertikální disparita). Tento problém se snažíme opravit právě přes lichoběžníkovou korekci. Například jsme pořídili snímek čtverce (viz následující obrázek 2.12).

Barevné lichoběžníky na černém pozadí představují jednotlivé snímky (pro levé a pravé oko). Černé pozadí představuje oblast snímku, která bude použita k vytvoření stereopáru. Pokud něco přesahuje až do šedé oblasti, tak už se to na displeji neobjeví. Bílý obdélníkový obrys ohraničuje okraje snímků z levé a pravé kamery. Okraj snímku z levé kamery je ve skutečnosti červený, v pravé kamery azurový. Jelikož na obrázku 2.12 jsou oba snímky přesně na sobě, jejich okraje lícují a tvoří dojem bílé barvy. Na obrázku 2.13 a dále už budou rozdíly mezi okraji obou snímků zjevné.



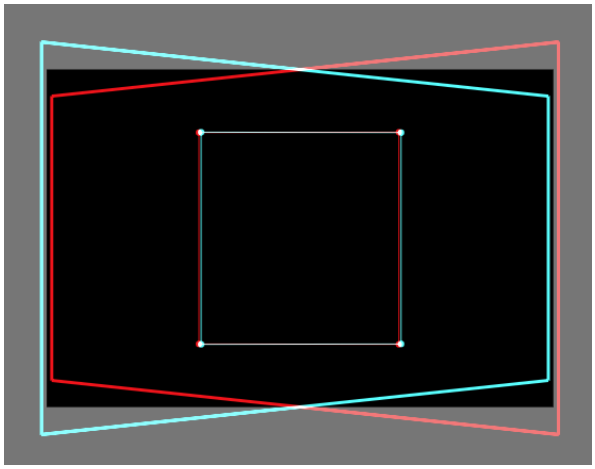
Obrázek 2.12: Snímek s perspektivním zkreslením.

Abychom odstranili tuto vertikální disparitu, tak použijeme lichoběžníkovou korekci (*Images Keystone*).



Obrázek 2.13: Snímek po aplikaci korekce *Images Keystone*.

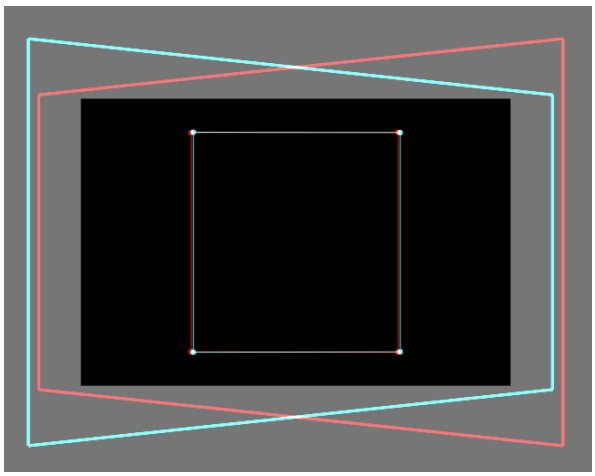
Nyní je vidět, že vertikální disparita již téměř neexistuje, za to se nám zdeformovaly snímky a problémem je, že levý a pravý snímek nyní nepokrývají celou plochu displeje. Než se vrhneme do opravy tohoto problému, tak se často ještě opravuje velikost horizontální disparity pomocí posuvu snímků. Nenulová horizontální disparita znamená, že 3D iluze vznikne mimo rovinu displeje a pokud by 3D iluze byla příliš velká, tak ji můžeme vzájemným posuvem snímků zmenšit a iluzi přisunout k blíže rovině displeje.



Obrázek 2.14: Snímek po aplikaci korekce *Images Shift*.

Nyní vidíme, že vertikální disparita téměř neexistuje a horizontální už

není tak velká. Nakonec tedy ještě použijeme zvětšení a vybereme jen tu část, pro kterou máme oba snímky (viz obsah černého pozadí).



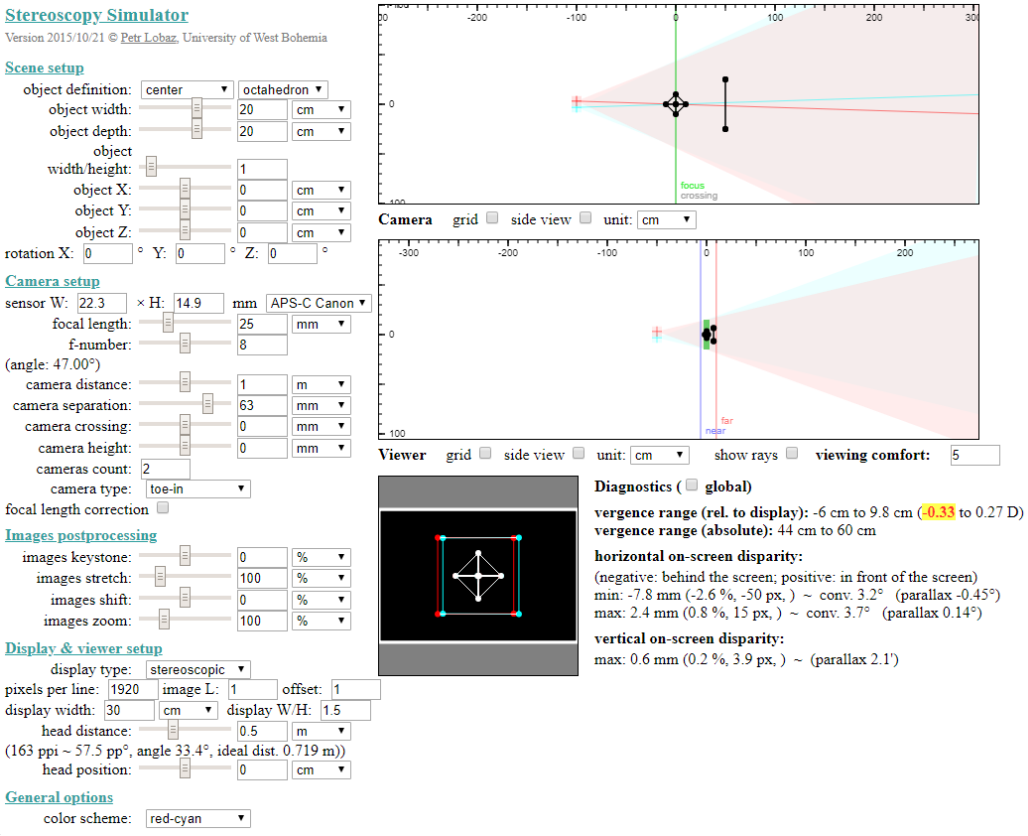
Obrázek 2.15: Hotový snímek po aplikaci korekce *Images Zoom*.

3 Analýza použití simulátoru

3.1 Seznámení s programem

Celý simulátor zde není možné podrobně představit, protože už se nejedná o triviální aplikaci (simulátor nabízí mnoho možností) a není zde dostatek prostoru. Uvedu proto jen některé důležité informace, na které bude navazovat další text.

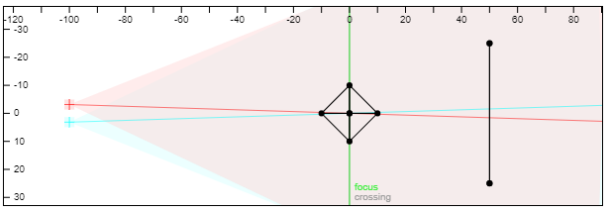
Obsah simulátoru lze rozdělit na tři oblasti. První z nich, jak je z obrázku 3.1 vidět, je levý panel (panel konfigurace). V tomto panelu se nachází veškerá dostupná konfigurace simulátoru. Druhou částí jsou jednotlivé pohledy (vizualizace) simulace. A třetí je oblast diagnostika, která je hned vedle posledního pohledu. V dalším textu jsou krátce představeny jednotlivé oblasti.



Obrázek 3.1: Stará verze simulátoru stereoskopie.

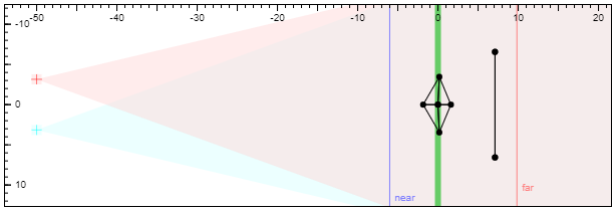
3.1.1 Vizualizace

Vizualizace jsou celkem tři. První ve výchozím nastavení simulátoru ukazuje půdorys scénu (viz obr. 3.2), kterou chceme zachytit. Barevné tečky vyzařující pomyslné paprsky reprezentují fotoaparáty, které tuto scénu snímají, a černé tečky a úsečky reprezentují objekty scény. Světle červená a světle azurová oblast znázorňují zorný úhel fotoaparátů. Červená a azurová polopřímka naznačují osy objektivů, resp. směr pohledů fotoaparátů. Svislé čáry označené texty focus a crossing znázorňují roviny, kam jsou objektivy zaostřeny a kde se osy objektivů kříží.



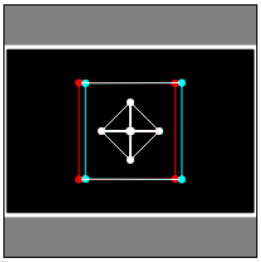
Obrázek 3.2: Vizualizace scény a kamer.

Druhý pohled (viz obr. 3.3) zobrazuje oči diváka (opět barevné tečky), který sleduje zachycenou scénu z první vizualizace (viz obr. 3.2). Tlustá zelená čára reprezentuje pomyslné promítací plátno a zbylé dvě čáry pomáhají určovat, kdy je objekt příliš daleko a kdy příliš blízko. Černé objekty naznačují, jaký 3D vjem divák uvidí (např. zde je vidět, že 3D iluze je plošší než původní scéna a že je mírně hloubkově zdeformovaná.).



Obrázek 3.3: Vizualizace diváka, který pozoruje vytvořený 3D snímek.

Na posledním pohledu (viz obr. 3.4) je vidět kompozice zachycených snímků, tj. to, co by divák na displeji viděl, kdyby neměl nasazeny brýle pro 3D vidění (např. červeno-azurové). Každý snímek je vykreslený jinou barvou a jejich vzájemný posun ukazuje zachycenou disparitu. Popis jednotlivých částí obrázku 3.4 byl uveden v kapitole 2.6.



Obrázek 3.4: Vizualizace jednotlivých fotografií kamer.

3.1.2 Panel konfigurace

V hlavičce panelu se nachází informace o dostupné verzi programu a autorovi. V těle panelu se nachází pět skupin konfigurací simulace. V první skupině (*Scene Setup*) se nachází možnosti manipulace s objekty. Tyto objekty se používají k tvorbě scény, kterou se snažíme zachytit. Přes první kontrolku (*Object Definition*) se vybere objekt, se kterým chceme pracovat, a další kontrolky poskytují různé možnosti manipulace s objektem. Ve druhé skupině (*Camera Setup*) se nacházejí různé možnosti nastavení kamer. Těmi, jak už bylo výše zmiňováno, ovlivňujeme především výsledný 3D vjem. Třetí skupina (*Images Postprocessing*) obsahuje možnosti upravování snímků v postprodukci. Čtvrtá skupina obsahuje sadu parametrů, které nastavují různé vlastnosti displeje, který zobrazuje vytvořený stereo snímek. Těmi také ovlivňujeme výsledný 3D vjem. V současné verzi jsou dostupné dva typy displejů (lentikulární a stereoskopický). Poslední skupina obsahuje obecná nastavení celého simulátoru. V současné verzi ale obsahuje pouze výběr barevného schématu vizualizace.

3.1.3 Diagnostika

Sekce diagnostika je oblast, která se nachází hned vedle posledního pohledu. V této oblasti se zobrazují další důležité informace, které především ukazují, jak kvalitní snímek se povedlo vytvořit. Obsahuje například informaci o velikosti horizontální a vertikální disparity.

3.2 Definice případu užití

Po prostudování problematiky 3D fotografování a seznámení s programem, je nutné definovat případy užití a skupiny uživatelů. Již z předcházejícího

textu vyšlo najevo, že nejčetnější skupinou uživatelů jsou studenti, kteří se učí techniky 3D fotografování. Další skupinou, jak jsem se od svého vedoucího dozvěděl, jsou profesionální fotografové, kteří simulátor používají k usnadnění práce. Fotografové například chtějí dopředu vědět jak nastavit kamery, aby byl výsledný 3D vjem co nejlepší. Jinak by to museli dělat stylem pokus-omyl. U každé skupiny uživatelů jsem pak identifikoval sadu případů užití, které nyní uvádím do přehledné tabulky.

Tyto případy užití byly použity během návrhu nových funkcí a úprav simulátoru. Také byly předlohou pro praktické otestování nové verze simulátoru (kapitola 6).

Případy užití

Skupina: Studenti
Jméno: Princip stereoskopie
Popis: Studenti se učí princip stereoskopie. Učitel podává výklad a současně v simulátoru demonstruje jak vzniká iluze hloubky při sledování 3D displeje. Simulátor tedy musí být schopen interaktivně ukázat, že se tvoří dva snímky, že se mírně liší, jak velký vliv má tato odlišnost a jak snímek či scénu upravit, tak abychom dostali 3D vjem před nebo za rovinou displeje.
Skupina: Studenti
Jméno: Základní aspekty ovlivňující vjem hloubky
Krátký popis: Studenti se učí jaké aspekty mají hlavní vliv na 3D vjem (viz také sekce 2.5). Učitel podává výklad a jednotlivé aspekty prochází. Simulátor musí být schopen demonstrovat jednotlivé aspekty. Jedná se především o možnost nastavování a ilustrace vlivu ohniskové vzdálenosti, mezikamerové vzdálenosti, vzdálenosti kamery od scény, vliv velikosti 3D displeje a vzdálenost diváka od displeje. Studenti si mohou zároveň sami ověřovat probírané možnosti a se simulátorem také pracují.
Skupina: Studenti
Jméno: Pokročilé aspekty ovlivňující vjem hloubky
Popis: Tento případ navazuje na předcházející případ, kde se studenti seznamovali s jednotlivými aspekty, které hlavně ovlivňují 3D vjem. V tomto případě se učí používat kombinace jednotlivých aspektů a jejich vliv na výsledný obraz. Učí se například co se děje s obrazem, když se zvětší ohnisková vzdálenost a současně zmenší mezikamerová vzdálenost. Simulátor tedy musí být schopen ilustrovat i tyto kombinace.

Skupina: Studenti
Jméno: Postprodukce snímků
Popis: Studenti se seznamují s technikami postprodukce. Učí se jednotlivé techniky a jejich vliv na výsledný 3D vjem. V simulátoru si například zkouší opravit perspektivní zkreslení (viz také sekce 2.6). Simulátor musí být opět schopný dostatečně intuitivně ilustrovat tyto techniky postprodukce a poskytnout potřebné ovládání.
Skupina: Fotografové
Jméno: Základní návrh parametrů 3D snímání
Popis: Fotograf má představu o scéně, kterou chce vyfotografovat. Navrhne si parametry snímání a sestaví si v simulátoru scénu. Podle potřeby provede úpravy nastavení a ověří jestli nastavení snímání a vytvořená scéna vedou k přijatelné obrazové kompozici (zejména ve smyslu podání perspektivy) a ověří si jestli nedochází ke konfliktu akomodace a vergence. Simulátor tedy musí být schopen nabídnout i pokročilejší možnosti nastavení scény pro zkušenější uživatele.
Skupina: Fotografové
Jméno: Pokročilý návrh parametrů 3D snímání
Popis: Fotograf má podobně jako v předcházejícím případě představu o scéně, kterou chce fotografovat, a chce opět vybrat parametry snímání. Tentokrát si chce ale vyzkoušet více kombinací. Nezajímá jej pouze, jestli se mu povede vytvořit přijatelný 3D vjem, ale zajímá jej také jak celkově působí. Chce si proto vytvořit několik variant a porovnávat je. Simulátor by tedy měl být schopný uživatelům dovolit vytvářet několik scén, které mohou porovnat.
Skupina: Fotografové
Jméno: Detailní návrh parametrů snímání
Popis: Fotograf opět řeší výběr vhodných parametrů, ale už jej zajímají podrobnější informace. Například jej zajímá, zda při vybrané konfiguraci bude docházet k vertikální disparitě, konkretní hodnoty disparit, jestli nebude docházet k divergentnímu očnímu pohybu (příliš velká disparita), jaká je maximální akceptovatelná disparita na lentikulárním displeji apod.

Tabulka 3.1: Tabulka případů užití.

4 Návrh nové aplikace

V této kapitole je postupně popsán proces návrhu aplikace. Nejprve je popsán návrh uživatelského rozhraní, pak návrh architektury nové aplikace a nakonec postup při výběru nástrojů, které byly použity k realizaci.

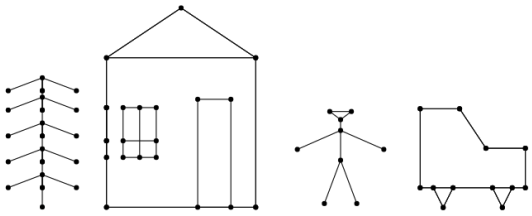
Na začátek je nutné říci, že aplikace bude realizovaná opět jako webová aplikace, která bude použitelná i na mobilních zařízeních. Důvod k tomuto rozhodnutí je jednoduchý. U aplikace jsme chtěli jednoduše zajistit multiplatformní dostupnost a odstranit nutnost instalace programu.

4.1 Identifikace nových funkcí a rozšíření simulátoru

Po seznámení s 3D fotografováním a simulátorem, konzultaci s vedoucím práce a na základě identifikovaných případu užití se došlo k závěru, že by bylo vhodné simulátor rozšířit o následující funkce. Změny a rozšíření simulátoru se především snaží zlepšit práci s uživatelským rozhraním a zvýšit uživatelský zážitek (*User Experience*).

4.1.1 Nové objekty scény

Jak jsem se od svého vedoucího dozvěděl a na dobrovolníku ověřoval, abstraktní objekty, jako je osmistěn a bod, uživatelům příliš nepomáhaly odhadnout, jak by se při stejném nastavení simulátoru (pozice kamer, vzájemné natočení kamer, nastavení snímače kamery atd.) zachovala realistická scéna. Do simulátoru se proto doplní několik objektů z reálného světa, které by studentům dovolily sestavovat „realistické“ scény a odstranily problémovou abstrakci. Příklady nových objektů jsou vidět na následujícím obrázku.



Obrázek 4.1: 2D pohled na nové objekty simulátoru: strom, dům, osoba a auto.

4.1.2 Práce s objekty scény

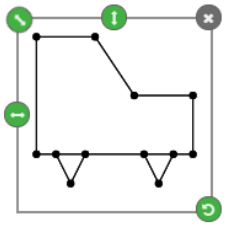
Další navrženou úpravou je práce s objekty. V současné verzi aplikace je možné do scény přidávat pouze 5 objektů. Pokud chci s nějakým objektem pracovat, tak si jej musím nejprve vybrat přes rozbalovací seznam (viz *Object Definition* z obrázku 3.1). Poté můžu objekt například posouvat po scéně úpravou souřadnic x, y a z v levém panelu. Tento zjevně nepohodlný přístup byl v první verzi simulátoru zvolen kvůli nejsnadnější implementaci. Uživatelsky příjemnější ovládání mělo oproti jiným funkcím simulátoru malou prioritu.

Navrhl jsem proto provést dvě větší změny. První z nich dodá možnost přidávat libovolné množství objektů do scény, aby bylo možné vytvářet i komplexnější scény. Přidávané objekty se budou v levém panelu zobrazovat jako seznam. Každou položku (objekt) seznamu bude možné libovolně rozbalit a zobrazit dostupná nastavení objektu (pozice, rotace atd.). Bude možné zobrazit nastavení i několika objektů najednou.

Druhou úpravou je přidání možnosti přímé manipulace s objektem. Možnost přesného nastavování přes souřadnice x, y a z v levém panelu je užitečná především pro zkušené uživatele, kteří už mají například představu o tom, co chtějí nasimulovat a mohou souřadnice přímo zapisovat. Avšak pro uživatele, kteří simulátor vidí poprvé nebo chtějí jen něco vyzkoušet a přesná pozice objektu není důležitá, bylo přesné zapisování pozice omezující. Domluvili jsme se proto do simulátoru přidat možnost s objektem manipulovat přímo pomocí tzv. kontrolního rámce (viz obrázek 4.2). Uživatel bude mít možnost v pohledu **Cameras view** kliknutím na objekt zobrazit tento kontrolní rámec a poté objekt intuitivně přemisťovat a provádět s ním základní transformace (na základě nabídnutých možností v levém panelu).

Při návrhu kontrolního rámce a jeho funkcí jsem se nechal inspirovat různými grafickými editory (jako je například online editor Vectr), které také po kliknutí na objekt zobrazují rámeček okolo vybraného objektu s dostupnými možnostmi manipulace. Protože u objektů jdou měnit rozměry, pozice a provádět rotace, tak jsem do kontrolního rámce objektu přidal možnost neproporcionální změny velikosti (roztažení objektu v jednom směru), proporcionální změny velikosti (zvětšení) a možnost rotace.

Také je nutné zmínit, že pohled scény v simulátoru je vždy 2D na 3D scénu (pohled shora a pohled z boku). Dostupné možnosti manipulace vždy odpovídají nastavenému pohledu. Například u pohledu shora je možné měnit hloubku a šířku objektu a u pohledu z boku hloubku a výšku.



Obrázek 4.2: Kontrolní rámec pro interaktivní manipulaci s objektem.

4.1.3 Import/export simulace

Další žádanou funkcí je import a export simulace. Tato funkce bude velmi užitečná při výuce. Vyučující bude mít možnost pro studenty připravit několik simulací (experimentů), poskytnout jim soubor s exportem simulace a využít simulátor jako pomůcku během vysvětlování probírané látky. Dalším užitečným případem užití je ukládání rozdělané práce. Studenti a fotografové budou mít možnost si své rozdělané experimenty uložit a případně i sdílet.

4.1.4 Výpočet a vizualizace hloubky ostrosti

Další rozšíření, které vyšlo najevo z nedokončené funkce v současné verzi simulátoru, je realizace výpočtu hloubky ostrosti pořízeného snímku a její vizualizace v simulátoru. Hloubka ostrosti nám určuje rozsah vzdáleností, ve kterých bude pořízený snímek ostrý. Příklady vlivu hloubky ostrosti jsou ilustrovány na následujících snímcích.

Obrázek 4.3: Příklady různých hloubek. První obrázek má malou hloubku ostrosti a druhý naopak velkou. (převzato z [10])

Na hloubku ostrosti má vliv několik faktorů, jako je například ohnisková vzdálenost kamery, velikost snímače, dokonce i rozlišení promítacího zařízení

a vzdálenost diváka. Detailní popis výpočtu zde uvádět nebudu, protože především vycházím z poznatků, které mi poskytl vedoucí práce. Důležité je pouze vědět, že výstupem jsou dvě hodnoty, které určují začátek a konec (vzdálenost od snímače) oblasti, ve které je snímek ostrý.

Podobně jako u pohledu *Viewer* se do pohledu *Cameras* přidají dvě vodící čáry, které budou ukazovat, kde oblast ostrosti začíná a končí. To uživatelům napoví, jak mají sestavit scénu nebo upravit zmíněné parametry, aby jejich scéna byla dostatečně ostrá. Informace o začátku a konci oblast ostrosti se přidá také do panelu diagnostik, aby zkušenější uživatelé mohli rychleji zjistit jejich přesnou vzdálenost a nemuseli informaci hledat v pohledu *Cameras*.

4.1.5 Více simulovaných scén

Již z identifikovaných případů užití vyšlo najevo, že je někdy výhodné mít možnost vytvořit si několik experimentů najednou, například si chci vyzkoušet několik konfigurací kamer, a následně je vzájemně porovnávat. V současné verzi simulátoru taková možnost není. Pokud bych chtěl v současné verzi simulátoru vytvořit několik experimentů najednou, musel bych si otevřít několik záložek v prohlížeči. Do každé záložky si načíst simulátor a pak mezi záložkami přepínat. Tento přístup však není moc praktický a uživatel například nemá ani možnost si jednotlivé experimenty pojmenovat, aby se v nich mohl lépe orientovat. Do simulátoru se tedy přidá možnost vytvářet několik scén najednou, jednoduše mezi nimi přepínat, pojmenovávat je a mazat.

4.1.6 Schránka

Toto rozšíření navazuje na již výše popsané rozšíření (Více simulovaných scén). Do schránky bude možné jednoduše ukládat části konfigurace simulátoru, jako je například nastavení kamer, a dle potřeby vkládat zpět do scény. Bude tak možné kopírovat části simulace mezi scénami a částečně nahradit chybějící funkci *undo* při práci se simulátorem.

4.1.7 Práce s pohledy simulace

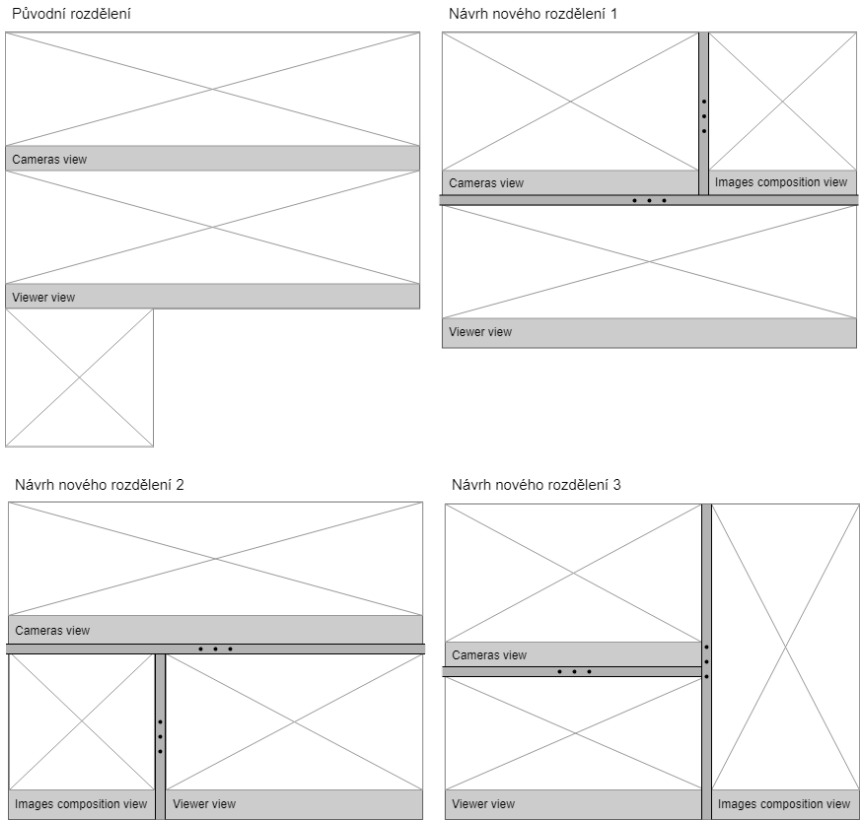
Další výraznou změnou je přepracování pohledů simulace. V současné verzi aplikace jsou pohledy rozděleny do tří řad, jak je vidět na prvním wireframu[1] v obrázku 4.4. První dva pohledy se přizpůsobují velikosti

[1] Wireframe („drátěný model“ nebo častěji „skica webu“) se u vývoje webových stránek používá pro náhled nového řešení.

obrazovky a poslední pohled (pohled Images composition view) má fixní velikost. Během práce se simulátorem je však často výhodné si rozměry pohledů podle potřeby různě upravovat nebo vytvářet více pohledů na jednu část simulace (především u scény s fotoaparáty) nebo pohledy dokonce úplně skrývat.

K realizaci této funkcionality je vhodné vyjít ze znalostí různých 3D modelářů (jako je například AutoCAD), které už podobné problémy řešily a nabízí různé pokročilé možnosti, například často obsahují různá předdefinovaná rozložení pohledů. Avšak realizace takto rozsáhlé funkcionality by byla časově velmi náročná. Navrhl se proto následující kompromis. Pohledy budou stále tři, ale seskupí se do dvou řádků, protože u aplikace se předpokládá, že bude zobrazována především na šířku a u prvních dvou pohledů není nutné aby vyplnily místo až do konce obrazovky. Dále se mezi pohledy přidají posuvníky, které dovolí upravovat velikost pohledu dle potřeby. Posuvníky budou označeny třemi tečkami a kurzor myši se po najetí bude také odpovídajícím způsobem měnit na typ *resize*, aby účel posuvníků bylo možné snadno rozpoznat. Nakonec se řešilo vzájemné uspořádání pohledů. Do úvahy se vzaly tři varianty, které jsou vyobrazené v obrázku 4.4. Po delší diskuzi s vedoucím práce jsme však nedošli k závěru, že by jedna z variant byla výrazně lepší. K realizaci se proto zvolila první varianta.

Pohledům se také přidalo výraznější značení (popisky), aby orientace v aplikaci byla o něco snazší. Následující wireframe však zmiňované zlepšení nezachycuje.



Obrázek 4.4: Návrhy rozložení pohledů. První wireframe ukazuje původní rozdělení a následující tři jsou nové návrhy.

4.2 Orientace v programu

V předcházejícím textu byl uveden seznam nových funkcí, které budou do simulátoru implementovány. Nyní následuje seznam důležitých změn, které jsou cílené především na nové uživatele a snaží se jim usnadnit orientaci v programu (zvýšit *User Experience*).

4.2.1 Nápověda k prvkům konfigurace

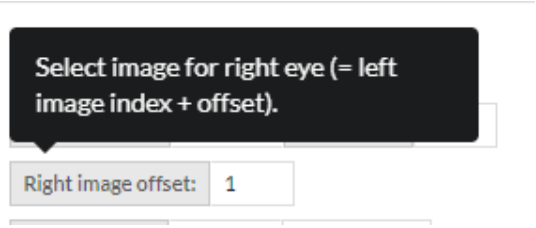
Protože už se nejedná o triviální aplikaci a ani dostupná nastavení nepatří mezi běžně známá a používaná, bude nutné do nové verze simulátoru realizovat nějakou formu nápovědy, která by orientaci v programu usnadnila.

Při návrhu nápovědy jsem především řešil jakou formou jí realizovat. Mezi časté realizace, na které jsem narazil, patří vytvoření souhrnné příručky,

interaktivní průvodce a „vyskakující“ nápověda (angl. tooltip).

Souhrnná příručka bývá dostupná vždy někde na poslední záložce aplikace a poskytuje ucelený přehled informací. Interaktivní průvodce (angl. termíny *Walkthrough Guide* nebo také *Step-by-step Guide*), postupně prochází aplikací a vysvětluje jednotlivé funkce programu. Pro webové aplikace například existuje modul Intro.js, který umožňuje snadný vývoj tohoto průvodce. „Vyskakující“ nápovědy se umisťují k jednotlivým kontrolkám. Například po najetí na ikonku otazníku nebo popisek u kontrolky se zobrazuje nápověda.

Já jsem se rozhodl v tomto případě vybrat poslední verzi, kdy nápovědu umisťujeme ke kontrolkám, protože u souhrnné nápovědy bývá zdlouhavé nalézt potřebné informace a také její časté otevírání je přinejmenším frustrující. Souhrnná nápověda by byla užitečná v případě, že bych chtěl vysvětlit nějaký složitý jev. K tomu jsou však určeny především přednášky o 3D fotografování. U interaktivní nápovědy může být opět zdlouhavé nalézt potřebné informace, obzvláště pokud nejde přeskakovat. Navíc je tato nápověda vhodná především pro první seznámení s programem. Je zbytečné uvádět zde například informaci, jak probíhá nějaký výpočet, protože to uživatel s největší pravděpodobností stejně brzy zapomene. Na následujícím obrázku je pak ukázka nápovědy. Nápověda je vždy hned po ruce a není potřeba nikde složitě hledat. Tyto nápovědy nebudou umístěné u každé kontrolky, ale především u kontrolek, u kterých nemusí být na první pohled jasné k čemu slouží.



Obrázek 4.5: Ukázka „vyskakující“ nápovědy.[1]

4.2.2 Použití ikon

Drobnější úpravou, kterou jsem navrhl, je do simulátoru na různá místa přidat ikony. Při výběru ikon jsem se řídil radami z článku Icon Usability [6] a ikony umisťoval pouze na místa, kde pomohou urychlit předání důležité

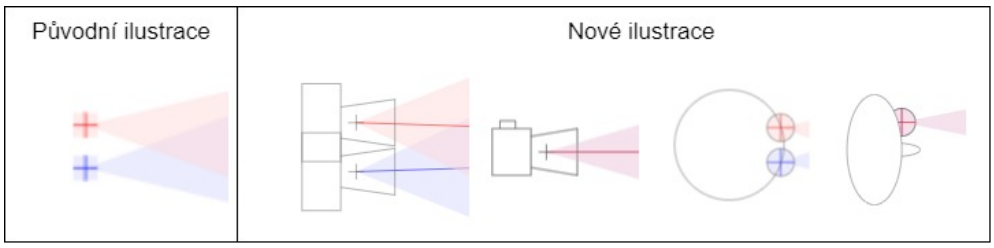
[1] Protože návrh odpovídal realizaci, tak byla použita ilustrace přímo z nové verze.

informace a nedojde spíše ke zmatení. Proto například jednotlivé kategorie nastavení simulátoru (Jako je Camera settings, Scene Configuration atd.) nemají žádné ikony, protože pro ně žádné standardní/zažité ikony neexistují a simulátor by musel být uživatelem využíván dennodenně, aby se vyplatilo definovat pro nastavení nové ikony, které by si uživatel musel nejprve zapamatovat.

Také jsem se řídil základním pravidlem a k ikonám doplňoval textový popis, aby nemohlo docházet k víceznačnosti. Například nové funkce simulátoru, jako je import a export, budou mít ikonu a textový popis. Pouze u ikon, kde k problému nedocházelo, nebo byl její význam hned po prvním použití jasný (například ikony + a - pro zoom), jsem textovou informaci vynechával.

4.3 Využití ilustrací

Další úpravu, kterou jsem navrhl, je nahrazení barevných teček, které v pohledu *Cameras* reprezentovaly fotoaparáty snímající scénu a v pohledu *Viewer* reprezentovaly oči pozorovatele, ilustracemi fotoaparátu a hlavy, aby vizualizace byly více intuitivní. Toto zlepšení má především pomoci uživatelům se rychleji zorientovat v pohledech simulátoru a pochopit jejich význam. Při první orientaci v programu mě tyto abstraktní tečky pouze zmátly, než aby mi pomohly simulátoru porozumět. Popsaný problém jsem ještě ověřoval u kolegy a došel jsem ke stejnému závěru. Na následujícím obrázku 4.6 je vidět navržená úprava. První obrázek ukazuje původní reprezentaci pomocí barevných bodů a následující obrázky nové návrhy. V pořadí vždy horní a postranní pohled na 3D scénu.

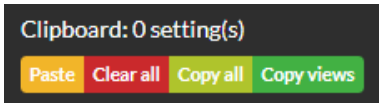


Obrázek 4.6: Návrh nových ilustrací fotoaparátu a očí.[1]

4.3.1 Detaily

Kromě výraznějších změn jsem navrhl i drobnější změny, které budou práci se simulátorem usnadňovat. Prvním je částečné zpřehlednění pohledu diagnostiky. Do tohoto pohledu, na základně navržené úpravy, přibudou ještě informace o hloubce ostrosti (angl. Depth of field) a další informace by měly ještě v budoucnu přibýt. Proto jsem obsah diagnostik rozdělil do bloků, podle toho k jakému pohledu zobrazované informace patří. Například informace o hloubce ostrosti byla zařazena do bloku Cameras view (viz nová implementace simulátoru).

Druhým drobným zlepšením je doplnění barevné informace. Například tlačítka akcí, jako je smazání objektu ze scény, obarvím červeně, abych tím dal najevo, že jde o určité riziko (například objekt smazat nechci). Jiným příkladem je skupina tlačítek s různými funkcemi, kde každé tlačítko označím trochu jinou barvou, abych je odlišil. Uživatel tak bude moci (po nějaké době) tlačítka využívat bez nutnosti čtení popisků.



Obrázek 4.7: Obarvení skupiny tlačítek (převzato z nové verze simulátoru).

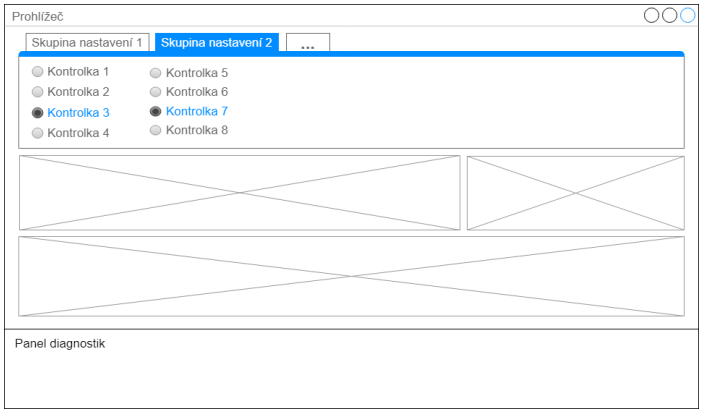
4.4 Návrh základního rozložení GUI

Kromě specifických změn simulátoru bylo také potřeba navrhnout novou podobu celé aplikace tak, aby byla lépe použitelná i na mobilních zařízeních a byla responzivní. Jak už jsem v předcházejícím textu zmiňoval, pro pohledy jsem navrhl nové rozložení a funkce. U panelu diagnostik jsem se rozhodl jej umístit do patičky stránky, protože tento panel obsahuje doplňkové informace k pohledům. Navíc k panelu diagnostik přidám možnost jej jednoduše schovat, aby uživatelé s nižším rozlišením displeje mohli panel zavřít a získali více prostoru pro pohledy. Panel bude také automaticky zavřený na mobilních zařízeních. Nakonec bylo potřeba vyřešit pozici panelu s konfigurací simulátoru, zde se nabízelo více možností a postup výběru je popsán v následující sekci.

4.4.1 Umístění panelu s konfigurací

K výběru vhodného místa (varianty) jsem použil metodu, které se říká A/B testování. U této metody se iterativně vytváří dvě soutěžící varianty návrhu a na základě zpětné vazby se vybírá nový šampion (varianta A). Po výběru šampiona začíná nové kolo. Je vytvořen nový vyzyvatel (varianta B) a opět se vybírá šampion. Počet iterací není nijak omezen. V každé iteraci by se mělo rozhodovat o jednom funkčním či designovém prvku.

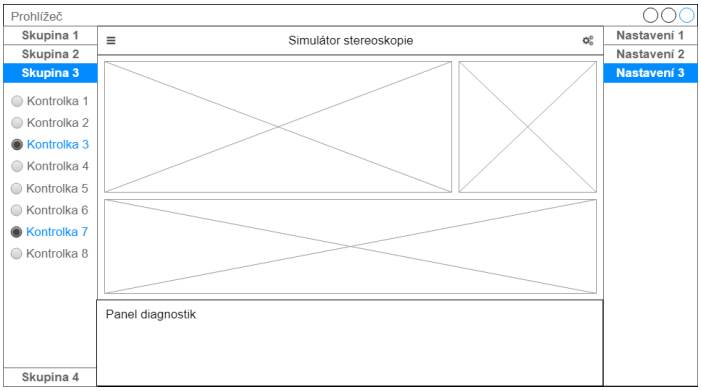
Vytvořil jsem tedy dva možné návrhy (A a B), které obsahovaly různé pozice panelu s konfigurací. U první varianty jsem uvažoval o přesunu prvků do horní části obrazovky a vytvoření karet pro jednotlivé skupiny prvků konfigurace, protože toto rozložení právě často mívají různé 3D modeláře (například AutoCAD). V případě rostoucího počtu prvků konfigurace (na jedné kartě) nebo u menších rozlišení displejů by se skupiny nastavení skládaly pod sebe. Místo toho, aby se řadily vedle sebe a vytvořily by například dva řádky (namísto jednoho). Následující *wireframe* ukazuje návrh možného řešení.



Obrázek 4.8: Wireframe prvního návrhu.

U druhého návrhu jsem se přesunul od spíše tradičního způsobu návrhu rozhraní k přístupu modernějšímu, který je vidět především u mobilních aplikací. Navrhl jsem panel s prvky konfigurace ponechat na levé straně (pro další výklad označím jako levý panel). Uživatelé však budou mít možnost panel dle potřeby schovat a také jednotlivé kategorie (např. Camera Configuration) nastavení bude možné dle potřeby zobrazovat a schovávat. Kromě levého panelu se vytvoří ještě pravý panel, do kterého se umístí některé globální funkce, které nesouvisí přímo se simulací. Jako je například možnost simulaci exportovat a importovat. Pravý panel bude možné také

dle potřeby zobrazit a schovat. U levého panelu by se v případě rostoucího počtu prvků konfigurace objevila možnost skrolování. Tento návrh je vidět na wireframu 4.9.



Obrázek 4.9: Wireframe druhého návrhu.

Obě varianty jsme s vedoucím práce probírali a dohodli jsme se, že by bylo lepší vybrat variantu číslo dva. K tomuto rozhodnutí mě dovedlo několik vlastností obou návrhů. Prvním je fakt, že chceme zaručit podporu mobilních zařízení, u kterých se cílí především na používání aplikace v horizontální poloze. Předpokládá se tedy, že aplikace bude používána na monitorech, které mají výrazně větší šířku než výšku, protože je to mnohem častější případ. Obzvláště na menších displejích pak nezbývá na výšku moc místa a první návrh by nezanedbatelnou část tohoto místa využil k zobrazování dostupné konfigurace. Pro pohledy simulace by tak nezbývalo moc místa a v tomto případě se volba prvního návrhu nezdála být vhodnou volbou.

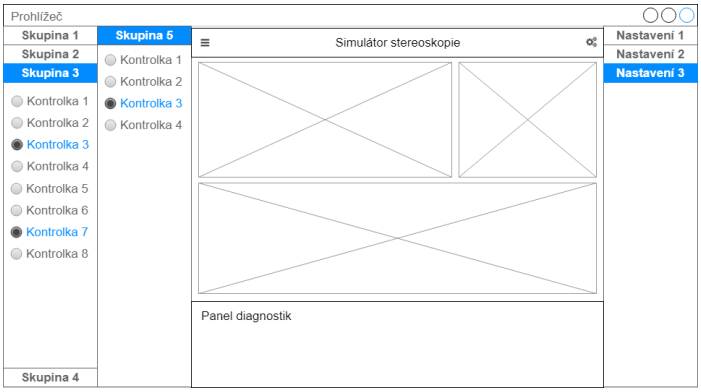
Výhodu kterou má první varianta, je velikost jedné záložky. Jak bylo zmíněno, předpokládá se, že displeje jsou širší než vyšší a v tomto případě by bylo možné na jednu kartu (záložku) naskládat více kontrolek než u varianty dva.

Dalším drobným problémem u prvního návrhu je nutnost definovat nějaké standardní velikosti jednotlivých kontrolek a pak je po skupinách uspořádat do sloupců, například po 4 (viz návrh). U řádků by byl problém ještě horší, jak bude z dalšího textu vidět. Tento přístup je značně omezující, protože se v simulátoru nacházejí různorodé kontrolky, které chceme například seskupovat podle jiných pravidel, než je zrovna sloupec o 4 prvcích (viz stereoskopický displej a lentikulární). V porovnání s druhou variantou by tak realizace byla o něco složitější a jednotlivé bloky kontrolek by musely mít různé velikosti.

Poslední uvažovanou výhodou návrhu dva je možnost otevření několika záložek najednou. To je užitečné například, když se snažíme nastavit kamery pro snímání a následně upravit vlastnosti promítacího zařízení, pracujeme přitom se skupinami Camera Settings a Display & Viewer Configuration. U návrhu by teoreticky šlo problém vyřešit také. Místo toho, aby názvy kategorií sloužily jako záložky, kdy si můžeme otevřít jednu záložku v jednom okamžiku, tak by fungovaly jako přepínače a v případě aktivace by jednoduše přidaly svůj obsah mezi zobrazené kontrolky. To však není úplně běžná realizace a mohla by být pro uživatele matoucí, navíc by bylo potřeba řešit problém s velkým množstvím zobrazených kontrolek, které se do zobrazované oblasti nemusí vejít.

Tím jsme tedy vybrali novou variantu A (návrh 2). Nyní ale bylo nutné ještě vyřešit již zmíněný problém s nedostatkem místa v případě, že otevřeme několik kategorií a zobrazená nastavení už se na výšku zobrazit nevejdou. Zde se opět vzaly do úvahy dvě varianty. První variantou je nechat kontrolky přetékat a přidat možnost skrolovat. Některé kategorie by se mohly poskládat za sebe, aby mohly být viditelné obě najednou, jako je například zmíněná kategorie Camera Settings a Display & Viewer Configuration. Druhou variantou je přetékající nastavení přesunou do nového sloupce. To znamená, že levý panel by měl například tři sloupce, pokud se otevřená nastavení nevejdou ani do dvou sloupců. V tomto případě se opět vybrala varianta druhá. Protože u první kategorie zmiňované seřazení kategorií fungovalo jen pro některé kategorie a pokud bychom například chtěli ještě vidět, k těm dvou zmíněným kategoriím, ještě informace o objektu scény, tak už by byla velká pravděpodobnost, že bude nutné skrolovat.

U vybrané druhé varianty se však ještě omezí přidávání sloupců a dovolí vytváření pouze dvou sloupců, aby se opět nestalo, že pro pohledy simulace nezbude žádné místo. Pokud by ani dva sloupce nestačily, tak by se zobrazovalo zmiňované skrolování. Tím se vytvořila finální varianta a wireframe výsledného řešení je vidět na následujícím obrázku.



Obrázek 4.10: Wireframe finálního návrhu.

4.5 Návrh architektury

Na začátku kapitoly jsem zmínil, že se bude jednat o webovou aplikaci. V této sekci sestavuji architekturu pro novou verzi aplikace a popisuji provedená rozhodnutí, která tvorbu nové architektury doprovázela.

Webová aplikace je klient-server aplikace. Jako klient vystupuje webový prohlížeč, ve kterém aplikaci spouštíme. Server je zdroj, ze kterého získáváme klientskou část. Za jako úplně základní rozdělení webových aplikací lze označit následující výčet:

- Webové aplikace s tenkým klientem.
- Webové aplikace s tlustým klientem.

První varianta je starším řešením, kdy je velká část logiky aplikace vykonávána na straně serveru. Uživatelům jsou posílány statické stránky s několika akcemi, které se následně opět zpracovávají na straně serveru. Klientská část aplikace často obsahuje jen pár řádek JavaScriptového kódu. Takové aplikace jsou také někdy označovány jako *Multi-page Application*.

U druhé varianty už se naopak většina logiky přenáší na stranu klienta a snaha je, aby server toho dělal minimum. Často na straně serveru bývají uložena pouze data aplikace se základní logikou. Klientská aplikace přitom může využívat ještě i další služby jiných serverů pomocí technologie AJAX.

Když ke druhé variantě ještě přidáme dynamické aktualizace stránky bez potřeby načítání celé stránky znovu, dostaneme webovou aplikaci, která se označuje jako *Single-page Application* (SPA). A právě jako SPA bude realizována i nová verze aplikace.[2] Samotné SPA mi však poskytuje jen

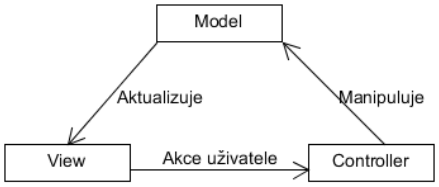
[2]I původní verzi aplikace lze označit jako SPA.

částečnou odpověď a k sestavení finální podoby architektury musím využít ještě další informace a architektonické vzory.

4.5.1 MVC, MVVM, MVP nebo MVW

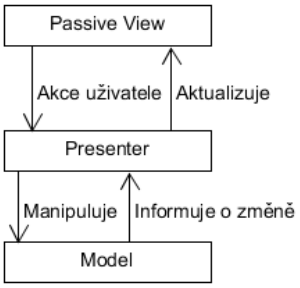
Nyní je potřeba určit základní strukturu aplikace. Možností je hned několik. Zkratky uvedené v nadpisu sekce představují architektury (architektonické vzory), které by bylo možné k realizaci aplikace použít. Kromě těchto architektonických vzorů existují i další, jako je *Hierarchical Model–view–controller* (HMVC). Z dalšího textu však bude jasné, že nebylo potřeba je brát v úvahu. Nejprve jen stručně představím 3 možné architektury a následně uvedu důležité kritérium, které pomůže architekturu určit.

Základní variantou je *Model–view–controller* (MVC), která se objevila už někdy v 80. letech. Architektura aplikaci rozděluje do 3 základních částí a určuje směr interakce (viz obrázek 4.11).



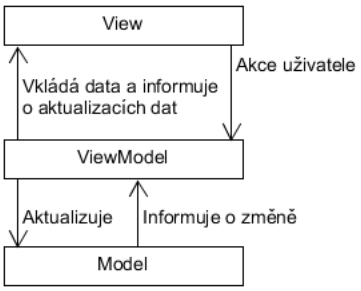
Obrázek 4.11: MVC architekura.

O něco později se objevily architektury *Model–view–viewmodel* (MVVM) a *Model–view–presenter* (MVP), které jsou upravené alternativy MVC. MVP byl poprvé definován někdy začátkem 90. let autory objektově orientovaného operačního systému Taligent. Tento architektonický vzor je určený především pro tvorbu uživatelských rozhraní a zlepšuje aplikaci procesu rozdělování zodpovědností (*Separation of Concerns* – SoC) v prezentační logice. Základním rozdílem, jak je z obrázku 4.12 vidět, je nahrazení *kontroleru* prvkem, který se označuje jako *presenter*. Na rozdíl od MVC už *pohledy* (angl. *views*) nejsou v přímém kontaktu s modelem, ale komunikaci zajišťuje právě *presenter*, který získává data z modelu. Získaná data podle potřeby dále transformuje a odpovídajícím způsobem aktualizuje pohled.



Obrázek 4.12: MVP architekura.

MVVM (také známý jako *Model–view–binder*) je dílem softwarových architektů, kteří pracující ve firmě Microsoft. Jeho základní myšlenkou je oddělení vývoje grafického uživatelského rozhraní od vývoje aplikační logiky a datového modelu. Při pohledu na obrázek 4.13 je vidět, že se opět změnily akce mezi jednotlivými částmi architektury a *kontroler* z MVC nahradil *view model*. Tento vzor je velmi podobný vzoru MVP a rozdíl je u komunikace s pohledem. Jednoduše řečeno, *view model* funguje jako prostředník mezi modelem a pohledem, kdežto *presenter* funguje jako vlastník pohledu a často je vazba 1:1.



Obrázek 4.13: MVVM architekura.

Posledním je *Model-view-whatever* (MVW), který není tak úplně vzorem, ale vznikl jako výsledek diskuze nad architekturou frameworku AngularJS. Pravděpodobně nejlepším popisem je komentář samotného autora MVW (zdroj [18]), pro který zde přikládám český překlad.

Po několik let byl AngularJS blíže k MVC (přesněji řečeno k jedné z jeho variant), ale postupem času se díky refaktorování kódu a zlepšení API přiblížil k MVVM ($scope objekt může být považován za ViewModel, který je obohacený funkcí kterou voláme Kontroler). Být schopný zařadit framework

do jednoho MV kbelíku má své výhody. Může to pomoci vývojářům získat větší důvěru k frameworku a usnadnit vytváření mentálního modelu aplikace. Také to může pomoci stanovit vývojáři používanou terminologii.*

Nicméně, raději bych viděl vývojáře vytvářet ohromující aplikace, které jsou dobře navržené a sledují Separation of Concern. Než abych je viděl mrhat čas dohadováním se o MV nesmyslech. Z toho důvodu tímto prohlašuji, že AngularJS je MVW framework, kde W znamená „whatever" (cokoliv ti vyhovuje).*

Z předcházejícího krátkého rozboru vyplynuly dva důležité závěry. Protože k realizaci aplikace budu ještě vybírat nějaký framework, tak není možné v tomto okamžiku vybrat jeden z popisovaných vzorů, protože bude především záležet na zvoleném frameworku.[3] Základní rozdělení všech vzorů je však stejné. Vždy obsahují nějakou formu pohledu, model a člen zpracovávající akce (*kontroler*, *presenter*, nebo *view model*). A toto rozdělení bez konkrétní formy budu vyžadovat i u výsledného řešení.

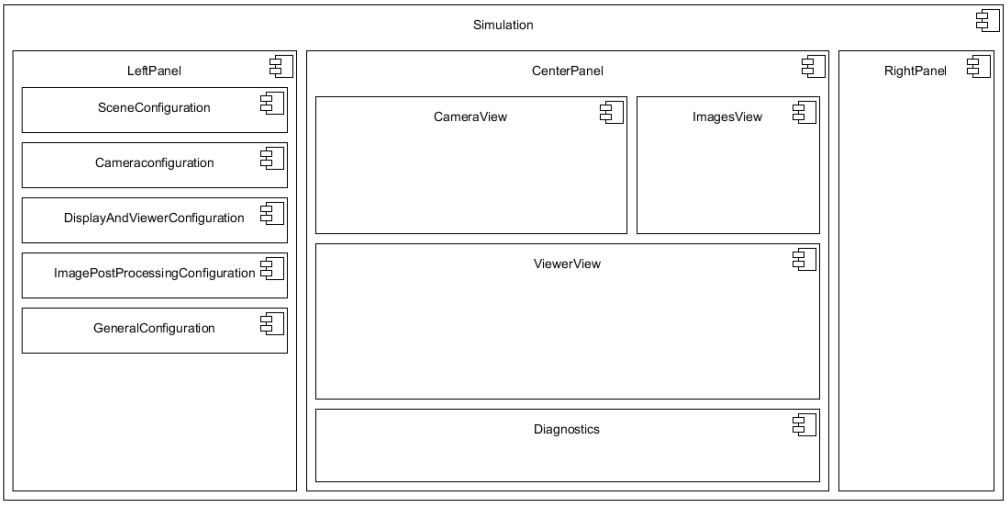
4.5.2 Rozdělení na komponenty

Dalším krokem je rozdělení jednotlivých částí aplikace do logických celků, přesněji řečeno komponent. Komponentový návrh je u webových aplikací stále relativně nový přístup (*Component-based Architecture* – CBA), při srovnání s dalšímy oblastmi softwarového vývoje (zdroj [19]). Základní myšlenou je rozdělení webové stránky (zobrazovaného obsahu) do oblastí, které pak mohou (do určité míry) pracovat nezávisle na ostatních. Každá komponenta má svou vykreslovanou oblast (HTML kód, DOM element [4]), aplikační kód, který spravuje vykreslovanou oblast, a model, který obsahuje data oblasti.

Protože části (oblasti) webové stránky jsou do sebe hierarchicky skládány, tak i komponentový návrh bude mít hierarchickou strukturu, kde rodič bude rozhodovat o vykreslování (použití) potomka. Na následujícím diagramu je vidět výsledné rozdělení aplikace do komponent.

[3] Důvodů k volbě frameworku je hned několik, ale v rámci této práce to již dále nerozvádím.

[4] Document Object Model je objektově orientovaná reprezentace HTML dokumentu.



Obrázek 4.14: Rozdělení aplikace na komponenty.

Základ tvoří komponenta simulace, která bude především určovat, jak se budou vykreslovat potomci, a bude jim poskytovat základní informace o prostředí (webovém prohlížeči). Například když dojde ke změně velikosti okna. Komponenty levý panel, pravý panel a centrální panel definují základní rozdělení aplikace. Komponenty levý a pravý panel se budou zobrazovat na vyžádání uživatele, tak jak bylo navrženo v sekci 4.4.1. Komponenta levého panelu bude také používat navrženou logiku na rozdělování panelu do dvou sloupců a bude zobrazovat obsah komponent, které obsahují dostupnou konfiguraci simulace. Každá komponenta uvnitř levého panelu představuje jednu konfigurační kategorii, tak jak byly představeny v sekci 3.1.2. Centrální panel bude uživatelům poskytovat rozhraní umožňující zakládání modelů simulace a zařizovat zobrazení komponent vizualizací (pohled na scénu, kompozice snímků a pohled pozorovatele na výslednou scénu) a diagnostika.

4.5.3 Komunikace mezi komponentami

Z předcházejícího rozdělení do komponent je jasné, že komunikace bude probíhat striktně z rodiče do potomka (rodič využívá služeb potomka). Pokud se rodič rozhodne potomkovi předat jako parametr *Callback* funkci, tak i definovaně z potomka do rodiče. Aby bylo možné zpracovávat některé globální události, které může například vysílat jedna z komponent, tak bude potřeba realizovat nějakou formu distributora událostí. Příkladem takových událostí je import a export simulace, které bude mít na starosti

komponenta pravý panel. Nebudu zde přímo předepisovat jaké API by měla implementovaná forma distributora událostí mít, ale musí komponentám poskytovat možnost se přihlásit a odhlásit k odběru událostí a také jim dovolit události rozesílat. Konkrétní realizace bude popsána v kapitole Implementace.

4.5.4 Datový model

Dalším důležitým rozhodnutím je umístění datového modelu, přesněji řečeno konfigurace simulace.[5] Nejprve bylo nutné určit, jestli data ukládat na straně serveru, nebo u klienta. Ukládání dat na serveru má očividnou výhodu dlouhodobé persistence dat. Protože se ale do aplikace také navrhla rozšíření, jako je přímá manipulace s objektem scény (viz sekce 4.1.2), tak by například při translaci objektu muselo docházet k nestálému aktualizování dat na serveru a k intenzivní komunikaci s klientem. To by mohlo způsobit trhavé překreslování scény při vzniku zpoždění. Alternativou by bylo data dočasně udržovat u klienta a s určitým intervalem je pravidelně synchronizovat. Tím už bych ale data ukládal na dvou místech. Dále by bylo nutné realizovat nějakou formu přihlašování do aplikace, aby se uživatelé mohli vždy přihlásit ke svým datům a celá implementace by se výrazně zkomplikovala. I přes uvedené možnosti se však získaná výhoda stálé perzistence dat nejevila natolik užitečná, aby se její implementace v současné době vyplatila. Dalším důvodem je navržené rozšíření simulátoru o funkci importu a exportu, které mohou uživatelé v případě potřeby použit k uložení své nastavené simulace.

Na straně klienta je možné data ukládat na několika místech. Pro malé množství dat je možné použít například *Cookies* nebo novější webová úložiště *Local Storage* a *Session Storage*. Tato úložiště však nejsou nejlepší volbou pro ukládání celých konfigurací aplikací. Už jen kvůli zmíněným velikostem[6]. Webové prohlížeče pro ukládání celých aplikací nabízejí vhodnější variantu. Kromě těchto úložišť prohlížeče ještě nabízejí IndexedDB databázi, která již je vhodná pro ukládání značného množství dat a poskytuje pokročilejší možnosti, jako je například tvorba tabulek a indexů. Nevýhodou však je, že ani jedno ze zmíněných úložišť neposkytne trvalé uložení dat[7]. Také by u nich

[5]Stará verze simulátoru má všechna data dostupná pouze skrze JavaScriptové objekty.

[6]Webová úložiště nabízejí min. 5 MB místa a konfigurace nové verze by se bez problému vešla. Simulátor se ale bude i dále rozšiřovat a úložiště by nemuselo stačit. Například z nějakého důvodu budeme chtít do simulátoru přidat nějakou formu práce s obrázky/diagramy.

[7]IndexedDB má ještě nastavení *Persistent*, které zabrání automatickému smazání dat ze strany prohlížeče. Uživatel však stále může data kdykoliv smazat a to často i nevědomě, když se rozhodne vyčistit stará data v prohlížeči.

mohlo opět docházet ke zpoždění při intenzivním ukládání dat (například při přímé manipulaci s objektem scény) a trhavému překreslování scény.

Na základě předcházejícího průzkumu jsem došel k následujícímu závěru. Protože z pohledu uživatele není důležité kde se data nachází, je tedy možné vybrat obě varianty. Ukládání dat na straně klienta má však jednu výraznou nevýhodu. Uživatelé by mohli dojít k mylnému závěru, že jejich nastavená simulace je bezpečně a trvale uložena a neprováděli by zálohu pomocí funkce export. Ukládání dat na straně serveru má značné výhody, ale jak jsem výše zmínil, realizace by byla o něco náročnější a pro novou verzi simulátoru je to spíše *Nice-to-Have* vlastnost. Proto jsem jsem se rozhodl data ponechat pouze v paměti, jako v původní verzi, ale rozdělit je do datových entit. Tyto entity bude možné v dalších verzích aplikace použít například při realizaci objektově relačního zobrazení (*Object-Relational Mapping* – ORM). Aby aplikace byla odstíněna od konkrétní implementace modelu, budou přistup k entitám zařizovat *Data Access Objects* (DAO).

Datové kolekce

Kromě hlavních dat (konfigurace simulátoru) aplikace také obsahuje několik předdefinovaných kolekcí dat, jako jsou například typy senzorů fotoaparátu. Pro tyto data není potřeba vytvářet entity stejným způsobem jako u modelu simulace (entity a DAO). V tomto případě bude stačit realizace pomocí konstant výčtového typu.

4.5.5 Přístup k datům

Protože budou data rozdělena do entit a přístup bude zařizovat DAO, tak pokud některá z komponent bude chtít přistoupit k nějaké konkrétní entitě, využije k tomu odpovídají DAO. Některé komponenty ale budou potřebovat přístup ke všem entitám. Takovou komponentou je například pravý panel, který při provádění akcí import a export potřebuje pracovat s celou konfigurací. V kódu pravého panelu se bude muset postupně získávat přístup ke všem entitám. To znamená, že by pravý panel musel znát všechny dostupné entity a jejich DAO. Také při každé nově přidané entitě a DAO by bylo potřeba upravovat kód, aby s ním komponenty, jako je pravý panel, počítaly.

Abych tomuto problému předešel, navrhl jsem přidat do aplikace objekt, který bude implementovat vzor *Service Locator*. Každé nově přidané DAO se bude u tohoto objektu registrovat a komponenty jako je pravý panel jej budou využívat k tomu, aby dostaly přístup ke všem dostupným entitám. Například pomocí metody `getAll()`. Je nutné zdůraznit, že tento objekt bude používán

jen pro výše uvedené případy a nebude fungovat jako náhražka vzoru vkládání závislostí (*Dependenci Injection* – DI), kde funguje jako návrhový antivzor (angl. *Anti-pattern*).

4.5.6 Import a export

U importu a exportu aplikace jsem uvažoval o třech možných řešeních. Prvním z nich bylo předávat konfiguraci simulátoru jako parametr v URL. To je, už jen kvůli velikosti konfigurace simulátoru, nevhodné řešení. Druhým řešením je poskytovat textový výstup (například JSON), který si může uživatel zkopírovat a uložit na libovolné místo. Při importu by export jen opět kopíroval zpět do vstupního okna. Tento přístup byl užitečný především v minulosti, kdy prohlížeče neměly možnost ukládat soubor na disk. Posledním řešením je již zmíněné ukládání na disk. Protože s ukládáním na disk větší problémy v dnešní době nejsou, zvolil jsem právě toto řešení.

Do souboru se bude ukládat jeden velký JSON objekt, ve kterém budou uloženy jednotlivé konfigurace všech modelů (pole entit) simulace a informace o aktivním modelu. Aby bylo možné ukládat například i stav pohledu komponenty (otevřené záložky, pozice posuvníku atd.), tak akce import a export využijí navrženého distributora událostí. V okamžiku spuštění akce se všem komponentám rozešle událost informující o prováděné akci (import, export). Pokud komponenta bude chtít exportovat stav svého pohledu nebo nějakou další interní konfiguraci, tak jí připojí do generovaného JSON objektu.

4.5.7 Shrnutí

V této sekci (4.5) jsem provedl návrh základních částí programu a vytvořil preskriptivni architekturu. V kapitole implementace bude popsána realizace nového simulátoru a finální verze architektury (deskriptivní architektura). V následující sekci vybírám vhodné nástroje k realizaci tohoto návrhu.

4.6 Výběr nástrojů

4.6.1 Výběr frameworku

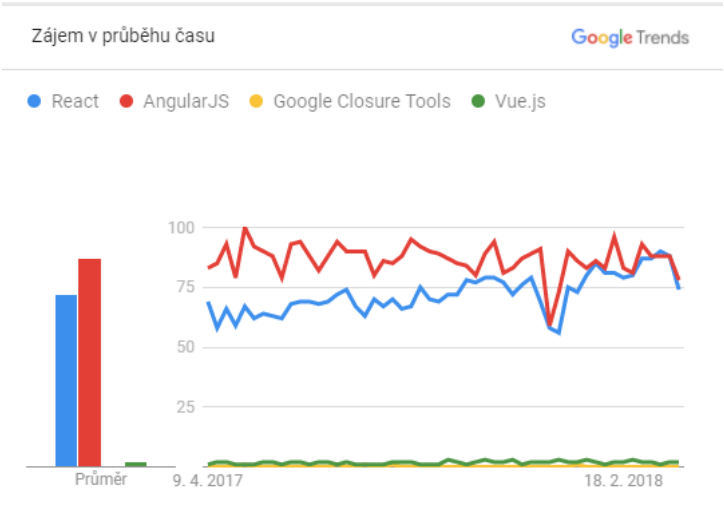
S rostoucím zájmem o vývoj webových aplikací se objevuje i mnoho JavaScriptových frameworků, které jsou různé kvality a s proměnlivou popularitou. V době psaní této práce se mi bohužel nepodařilo najít zdroj, který by obsahoval ucelený a aktuální přehled všech dostupných frameworků,

pomocí kterého by bylo možné výběr provádět. Ale i pokud by takový seznam existoval, tak by bylo detailní porovnání všech frameworků časově velmi náročné, ne-li téměř nemožné, protože dnes jich existují desítky. Jeden takový seznam se vytvářel na Wikipedii [9], ale ten již není aktuální.

Nesnažil jsem se proto provést porovnání všech frameworků a nalézt jeden, který by byl absolutně nejlepší, ale provedl jsem filtraci podle následujících kritérii:

- Aplikaci bude nutné dále rozšiřovat a to s největší pravděpodobností dalšími studenty.
- Vybraný framework musí mít budoucnost[8].
- Musí být možné v něm realizovat návrh ze sekce 4.5.
- Kód by měl být čitelný i bez hlubokých znalostí frameworku.

Nejprve jsem řešil rozšiřitenost frameworku. Když jsem hledal, které frameworky jsou aktuálně nejpopulárnější, nejčastěji jsem narážel hlavně na React, AngularJS a Vue.js. Z předcházejících zkušeností jsem znal ještě knihovnu Closure od firmy Google, kterou lze také použít ke stavbě kostry aplikace. Abych si jejich popularitu ještě částečně ověřit, tak jsem použil nástroj Google Trends a nástroje porovnal (viz následující obrázek).



Obrázek 4.15: Vývoj zájmu v průběhu času o zvolené frameworky.

[8]Některé frameworky jsou pouze pár měsíců staré a je velmi těžké odhadnout jejich budoucnost. Jiné mohou být naopak již velmi staré a zájem může opadat.

Z grafu je vidět, že největší zájem je o AngularJS a React, kde React postupně dohnal, ne-li předehnal, AngularJS. Protože oba nástroje mají repozitář v uložiti GitHub, tak jsem je ještě porovnal zde.

Project	angular.js	react
	angular Jan 6th, 2010 4 hours ago	facebook May 24th, 2013 17 hours ago
Community	★ 58236 ⑂ 28871 430 contributors	★ 92711 ⑂ 17487 454 contributors
Code	8757 commits **JavaScript**: 98% HTML: 1%	9780 commits **JavaScript**: 94% C++: 2% HTML: 1% TypeScript: 1% CoffeeScript: 1%
Development	606 issues 128 open 7578 closed 193 tags	424 issues 79 open 6589 closed 88 tags

Obrázek 4.16: Porovnání nástrojů na GitHub.com (převzato z [21]).

Z tohoto porovnání vyplývá, že React je na tom o něco lépe. Na druhou stranu AngularJS je plnohodnotný framework, který autoři označují jako MVW (viz sekce návrhu) a React je knihovna, někdy také označovaná jako *View-presenter* z MVP, která pracuje především s uživatelským rozhraním. Křivka učení je ale zase u AngularJS mnohem větší, než je u React, ale to i z pochopitelných důvodů [9].

Nakonec jsem rozhodl vybrat React, především kvůli rychlejšímu růstu popularity a snadnější orientaci v kódu, kterou pravděpodobně ocení především budoucí autoři nových verzí simulátoru. Chybějící datovou vrstvu u knihovny React je možné doplnit a většina dostupných funkcí AngularJS by se v tomto projektu ani nevyužila.

4.6.2 Výběr syntaxe

Nyní je ještě potřeba vybrat syntaxi pro psaní aplikace s ohledem na vybraný framework a výběr odůvodnit. I zde je několik možností. První z nich je používání knihovny React se starší verzí ECMAScript 5, která je téměř plně podporovaná všemi dnes dostupnými webovými prohlížeči. Psaní rozsáhlých aplikací (v dnešní době) je však v této verzi zbytečně složité a matoucí.

[9] AngularJS je mnohem komplexnější framework než React

ECMAScript 5 například neobsahuje ani běžnou definici třídy přes klíčové slovo `class`. Je potřeba využívat definici funkce.

Další možností je použít novější definici ECMAScript 6, která již v dnešní době má širokou podporu a je možné jí přímo používat. Také v případě potřeby je možné ECMAScript 6 přeložit (pomocí transpileru Babel) do starší verze. Je také možné jednoduše doplnit statickou typovou kontrolu pomocí nástroje Flow, který je jen dalším produktem autorů knihovny React.

Poslední možností u knihovny React je použití TypeScriptu. TypeScript je nový jazyk, který je ale téměř identický s jazykem ECMAScript. Plus doplňuje některé další syntaktické konstrukce, které známe z vyspělejších jazyků, jako je definice rozhraní třídy. Kód programu se pak opět pomocí transpileru překládá do obyčejného JavaScriptu (ECMAScriptu).

Já jsem se rozhodl vybrat ECMAScript 6 a rozšíření Flow. K tomuto výběru jsem měl dva důvody. Za prvé, nástroj Flow je od stejných autorů a dá se předpokládat, že jeho používání bude (s knihovnou React) naprosto bezproblémové. TypeScript je na druhou stranu od Microsoftu. Druhým důvodem byl fakt, že jsem hledat především podporu statické typové kontroly, která je u obou nástrojů téměř identická (syntaxe).

JSX

Framework React ještě nabízí JSX syntaxi, která je dalším rozšířením JavaScriptu a dovoluje používat XML značky přímo v kódu JavaScriptu, čímž umožňuje přehledně definovat vykreslovanou oblast přímo u logiky, která zpracovává uživatelské události této oblasti. Efektivně tak umožňuje vytvářet volně svázané komponenty uživatelského rozhraní.[10] Alternativním postupem je podporu JSX v Reactu vypnout, používat standardní metody DOM objektů[11] a vykreslovanou oblast z těchto objektů vytvářet. Já jsem pro realizaci práce zvolil řešení právě s podporou JSX, protože to umožňovalo vytvářet daleko lépe čitelný kód.

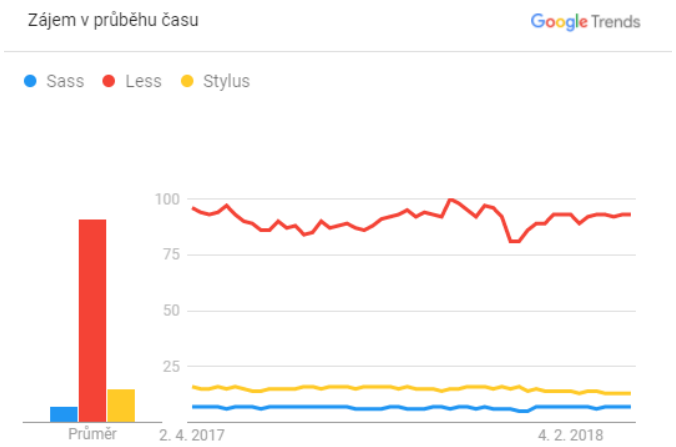
CSS, LESS, Stylus nebo SASS

Nakonec bylo nutné zvážit zda použít nějaké rozšíření (CSS preprocesor) pro CSS, jako je LESS, Stylus nebo SASS, která především přináší modularitu a znovupoužitelnost úseků stylů. To velmi usnadňuje vývoj při velkém množství stylů. Protože ale v mém případě budu vybírat UI knihovnu, která již bude mít předdefinované komponenty a vlastní styly, tak jsem se rozhodl použít pouze čisté CSS. CSS budu používat pouze pro upravování existujících

[10] Autoři Reactu tento přístup ještě dále brání v článku Introducing JSX [24].

[11] Například `document.createElement("BUTTON")`.

komponent a k vytváření základních stylů pro kostru aplikace. Pokud by se v budoucnu uvažovalo o přidání nových komponent nebo předělávání designu aplikace, tak by bylo vhodné výběr CSS procesoru zvážit. Bez hlubšího porovnávání se zdá, že aktuálně nejrozumnější volbou by byl LESS, o který je největší zájem a klíčové vlastnosti mají všechny CSS preprocesory stejné.



Obrázek 4.17: Vývoj zájmu v průběhu času o CSS preprocesory.

4.6.3 Výběr UI knihovny

Dalším krokem je výběr UI knihovny. UI knihovnu jsem vybíral především proto, abych zajistil jednotný vzhled celé aplikace a to i v případě dalšího rozšiřování aplikace. UI knihoven pro webové aplikace je opět velké množství a není možné je v této práci všechny podrobně porovnávat.

Při výběru jsem především hledal UI knihovnu, která by již v základu podporovala React, měla vytvořené komponenty pro jednotlivé kontrolky a elementy UI. Tím bych se vyhnul ručnímu tvoření komponent. Po aplikaci tohoto kritéria jsem nalezl tři často doporučované kandidáty:

- Semantic UI,
- Material UI,
- Blueprint.

Material UI je knihovna, která vytváří UI podle příručky *Material Design*, kterou vytváří firma Google. V době psaní této práce však byla React verze této knihovny stále ve stavu beta a druhou velkou nevýhodou byla skutečnost, že její jednotlivé UI prvky potřebují hodně prostoru. To je pro mou aplikaci

velká nevýhoda a musel bych knihovnu ještě dále upravovat aby vyhovovala mým potřebám. Její výběr se proto nejevil jako rozumná volba.

Naopak knihovna Blueprint, která byla v době výběru ve verzi 1.3 (nyní má již verzi 2), má prvky UI vytvářené právě s ohledem na zobrazování velkého množství dat. Některé UI prvky však nebyly vytvářené zrovna postupem *Mobile-first* a na mobilních zařízeních by byly těžko ovladatelné (například posuvníky). Proto ani tato knihovna nebyla ideální volbou.

Poslední je Semantic UI, která už netrpěla zmíněnými problémy. Malou nevýhodou byl o něco menší výběr UI prvků, při srovnání s Blueprint knihovnou. To však nebyl takový problém. Jediný prvek, který v knihovně chyběl byl posuvník. Pro ten jsem však našel ještě samostatný modul (React komponentu). K realizaci jsem proto vybral právě Semantic UI.

5 Implementace

V této kapitole je postupně popsána finální implementace aplikace vytvořená na základě informací získaných v předcházejících kapitolách a nástroji popsanými v sekci 4.6. Nejprve je popsána adresářová struktura aplikace, poté je představena finální verze architektury aplikace (diagram tříd), popsán tok programu, představeny základy knihovny React, uvedeny možné způsoby rozšiřování simulátoru, důležité části implementace, rozšíření *Progressive Web App*, proces sestavení a podporované verze webových prohlížečů.

Při implementaci nové verze simulátoru se myslelo i na *Best Practices* a aplikace například obsahuje důležité meta-informace, výrazné a hierarchicky uspořádané nadpisy, jednotný design aplikace, popis stránky atd. Používal jsem přitom nástroj *Coach Panel* (rozšíření do webového prohlížeče Chrome), který stránku hodnotí a trestné body aplikace dostala pouze za použití HTTP namísto HTTPS.

5.1 Adresářová struktura

Pro novou implementaci simulátoru jsem navrhl následující adresářovou strukturu. Vycházel jsem přitom ze standardní adresářové struktury React aplikaci, kterou jsem dále upravil podle návrhu (především adresář `src`).

Kořenový adresář aplikace /
V kořenovém adresáři se kromě dalších adresářů nacházejí další podpůrné soubory, jako jsou například `README.md` a `.gitignore`
/build
Tento adresář se vytváří automaticky a obsahuje produkční verzi aplikace. Adresář se vytváří během procesu sestavení (viz sekce 5.12).
/node_modules
Teto adresář je také automaticky generovaný a ukládají se do něj moduly, které se uvedou v závislostech v souboru `package.json`. Každý modul může mít ještě vlastní závislosti, proto počet modulů v adresáři může být vyšší, než je v souboru.
/public
Adresář obsahuje soubory, které se používají přímo v produkční verzi. Jako je `index.html` a `manifest.json`.
/src
Adresář obsahuje veškerenou logiku aplikace. Všechny JavaScriptové soubory

a také styly aplikace (CSS soubory).

/src/css

Jak bylo popsáno v sekci 4.6.2 aplikace obsahuje jen jeden CSS soubor, který obsahuje pouze základní styly aplikace a upravující styly pro komponenty Semantic UI.

/src/services

Do adresáře jsou ukládány třídy poskytující různé služby.

/src/helpers

Do adresáře jsou ukládány další pomocné třídy, které jsou používány službami ze složky `services`.

/src/model

V adresáři se nachází implementace datové vrstvy aplikace.

/src/model/data_collections

Obsahuje různé předdefinované kolekce dat, jak bylo popsáno v sekci 4.5.4.

/src/model/drawable_objects

Obsahuje kolekci kreslitelných objektů.

/src/model/entities

Obsahuje datové entity pro jednotlivé skupiny konfigurace simulátoru. Tyto entity mohou sloužit jako předloha k tvorbě databázových tabulek v budoucích verzích. Jak bylo zmíněno v sekci 4.5.4. Přístup k těmto entitám zařizují odpovídající DAO třídy.

/src/views

Adresář `views` obsahuje React komponenty podle návrhu ze sekce 4.5.2. Z důvodu lepší přehlednosti byly vytvořeny ještě dva následující podadresáře.

/src/views/left_panel_modules

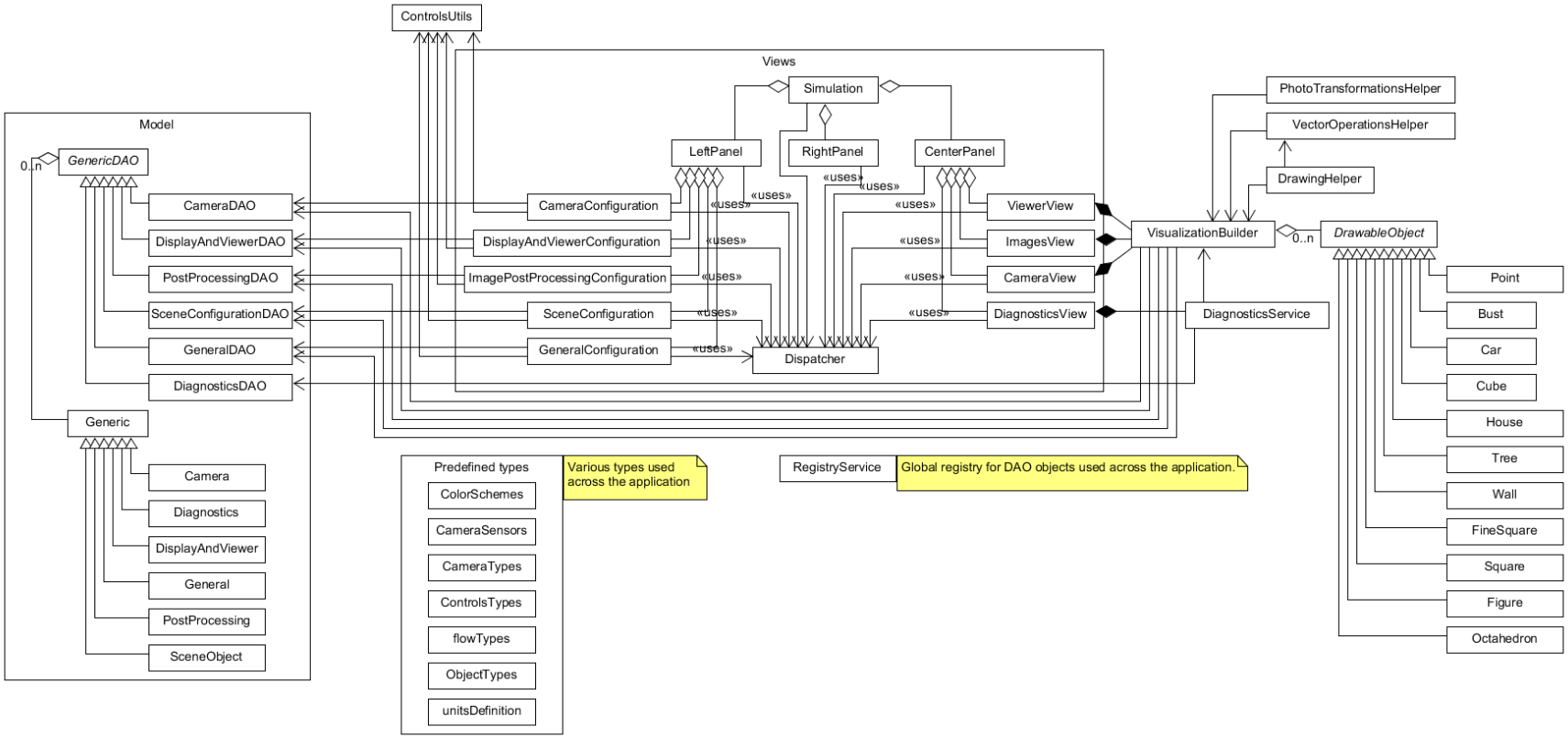
Do tohoto adresáře byly přesunuty komponenty, které vykreslují jednotlivé skupiny kontrolek konfigurace simulace.

/src/views/simulation_views

Adresář obsahuje komponenty jednotlivých pohledů simulace.

5.2 Struktura aplikace

Následující diagram tříd ukazuje strukturu finální implementace aplikace.



Obrázek 5.1: Diagramu tříd nové implementace v1.01

5.3 Použití knihovny React

Než se budu moci pustit do popisu toku programu (který následuje v další sekci), tak musím alespoň částečně představit základy knihovny React, bez kterých by nebylo možné další popis provést. Další informace o knihovně React je možné najít v dokumentaci projektu [22].

Základním stavebním kamenem knihovny React je komponenta. Tato komponenta se definuje jako třída, která rozšiřuje třídu `React.Component`. Každá komponenta si definuje a spravuje část webové stránky (DOM). Celý obsah webové stránky je rozdělen do několika komponent. Každá nová třída (komponenta) musí implementovat metodu `render`. V této metodě si třída definuje obsah oblasti (vykreslované oblasti) a vrací jej jako výsledek. Jak jsem zmiňoval v kapitole návrhu, obsah vykreslované oblasti definuji pomocí JSX syntaxe. Pro lepší představu ještě přikládám velmi jednoduchou ukázku React komponenty.

```
// definice komponenty
class HelloWorld extends React.Component {
  render() {
    return (<div>Hello World</div>);
  }
}
// vykresleni komponenty HelloWorld do DOM elementu mountPoint
ReactDOM.render(<HelloWorld/>, mountPoint);
```

Výpis 5.1: Velmi jednoduchý příklad React komponenty.

Každá komponenta má svůj životní cyklus a stav. Dále má několik metod, které volá při přechodech mezi stavy. Tyto metody je možné použít i k implementaci vlastní logiky. Nebudu zde popisovat všechny dostupné metody, protože v aplikaci se používají pouze tyto tři `componentDidMount`, `componentWillUnmount` a `componentDidUpdate`. Metoda `componentDidMount` je volána v okamžiku, kdy je komponenta přidána do webové stránky (DOM). Například při volání `ReactDOM.render` nebo v metodě `render` jiné komponenty. Metoda `componentWillUnmount` je naopak volána v okamžiku, kdy je komponenta odebírána ze stránky. Metoda `componentDidUpdate` je volána pokaždé, když dojde k nějaké aktualizaci vykreslované oblasti komponenty. Metoda `componentDidMount` je u komponent aplikace používána především k registraci odběru událostí a metoda `componentWillUnmount` naopak ke zrušení tohoto odběru (viz sekce 5.7 a 5.4.1).

U každé komponenty je také možné definovat vlastní proměnné, které budou ovlivňovat stav komponenty. Proměnné se definují v konstruktoru třídy. K vyvolání změny stavu komponenty přes vlastní proměnou je potřeba

použít metodu `setState`. Parametrem metody je nová hodnota nastavované proměnné. Po zavolání metody `setState` knihovna React vyhodnotí, jestli je potřeba provést aktualizaci vykreslované oblasti. Pokud je nutné aktualizaci provést, tak knihovna aktualizuje jen tu část vykreslované oblasti (DOM element), které se změna týká. Při rozhodování, kterou část aktualizovat, používá heuristický algoritmus (viz zdroj [23]). Po provedení změny ja také volána metoda `componentDidUpdate`. Pro lepší představu je opět přiložen kód ukazující jednoduchou práci se stavovou proměnou.

```
// definice stavove promenne
constructor(props: Props) {
    this.state = {
        drawCameraGrid: false
    }
}
// ukazka aktualizace promenne
this.setState({
  drawCameraGrid: true
});
```

Výpis 5.2: Ukázka definice proměnné ovlivňující stav.

5.4 Tok programu

V této sekci je velmi zjednodušeně popsáno, jak probíhá spuštění programu a způsob jakým se zpracovávají události v programu. Spuštění aplikace probíhá v souboru `index.js`. Nejprve se zkontroluje, zda jsou dostupné pasivní verze posluchačů událostí DOM objektů[1]. Provede se spuštění služby *Service Worker* (viz dále sekce 5.11), pokud je služba dostupná, a zavolá se metoda `ReactDOM.render`. Ta spustí generování simulace do předem zvolené HTML značky s ID `simulation-wrapper`. HTML značka je umístěná v souboru `index.html`. Prvním parametrem metody `ReactDOM.render` je jméno vykreslované komponenty, která vykreslí svůj obsah do zvolené značky. V tomto případě jde o komponentu `Simulation`, která je základní komponentou celé aplikace. U komponenty je volána metoda `render`, ve které je pomocí JSX syntaxe nadefinovaný obsah komponenty. V tomto případě celé aplikace. JSX dovoluje kromě běžného HTML vkládat i další komponenty, takže výsledný obsah komponenty může

[1]Ty umožňují o něco rychlejší zpracování uživatelských událostí, jako je například kliknutí, protože zakazují používání (ignorují) některých funkcí uvnitř metod, které událost zpracovávají.

být skládán i z několika dalších komponent. A právě podle návrhu hierarchie komponent 4.14 byl upraven i obsah `render` metod jednotlivých komponent.

Každá komponenta si v metodě `render` ještě (kromě obsahu) zaregistruje události, které může uživatel spouštět (např. kliknutí na různé části obsahu). Ke každé takové události si ve své třídě definuje metodu, která bude událost zpracovávat. Tyto metody mají prefix `handle` a definují vstupní body aplikace. Každá komponenta jich má několik, podle svého obsahu. Komponenty mezi sebou musejí také často interně komunikovat a v následující sekci je popsána implementace této komunikace.

5.4.1 Generované události

Jak bylo zmíněno již během návrhu aplikace, komponenty jsou hierarchicky uspořádané a komunikace může přímo probíhat pouze mezi rodičem a potomkem. Proto když chtějí informovat ostatní komponenty, které nejsou jejich potomkem, nebo rodičem, tak musejí rozeslat událost přes distributora událostí (viz sekce 5.8). V následujícím textu je ještě uveden výčet těchto událostí a odpovídající popis.

- **resize** – Rozesílaná při změně velikosti obrazovky (okna prohlížeče).
- **modelSwitch** – Rozesílaná při přepínání mezi dostupnými scénami. Událost používají například komponenty levého panelu, aby aktualizovaly svůj obsah.
- **configurationUpdate** – Rozesílaná po aktualizaci některého z panelů konfigurace. Například při změně ohniskové vzdálenosti. Tuto událost využívají komponenty pohledů a komponenta `Diagnostics`, aby mohly aktualizovat svůj obsah.
- **exporting** – Rozesílá se během exportu konfigurace simulace. Komponenty do předávaného objektu ukládají svou konfiguraci. Využívají komponenty, které chtějí obnovovat svůj stav, jako jsou například komponenty pohledů (pozice posuvníků, nastavení zoomu, atd.).
- **importing** – Rozesílá se během importu nové konfigurace. Komponentám se předává objekt, ve kterém se nachází jejich dříve uložená konfigurace.
- **paste** – Rozesílaná po vložení nové konfigurace do aktivního modelu. Tuto událost odebírá každá komponenta, u které je možné ukládat její stav do schránky. Komponentám se předává objekt, který obsahuje vkládaná nastavení.

- **sceneObjectSelected** – Rozesílá se po vybrání objektu ve scéně (zobrazení kontrolního rámečku objektu). Událost rozesílá komponenta `CameraView` a odebírá komponenta `SceneConfiguration`. V komponentě `SceneConfiguration` se využívá k otevírání panelu nastavení vybraného objektu.
- **sceneObjectDeleted** – Rozesílá komponenta `SceneConfiguration` poté, co uživatel klikne na tlačítko *Delete* u vybraného objektu scény. Odebírá komponenta `CameraView`. Ta schová otevřený kontrolní rámeček, pokud byl otevřený nad mazaným objektem.
- **modelDelete** – Informuje odběratele, že došlo ke smazání modelu (scény) s ID x. Využívají komponenty, které si udržují svůj stav (jako jsou například pohledy), aby také provedly odstranění uložených nastavení pro daný model.
- **getCenterPanelSettings** – Slouží ke sběru nastavení komponent v centrálním panelu (především jednotlivých pohledů). Událost používá funkce *Copy views*, která získaná nastavení uloží do schránky.
- **sceneConfigurationUpdate** – Tato událost je speciálním případem události `configurationUpdate`. Používá se k aktualizaci pozice objektu scény v levém panelu. Generuje se například při přímé manipulaci s objektem přes výběrový rámec (viz obrázek 4.2).

5.5 Import a export

Import a export je realizovaný podle návrhu ze sekce 4.5.6. Tyto dvě akce má na starosti komponenta pravého panelu (metody `handleSimulationExport` a `handleSimulationImport`). Během exportu se nejprve přes realizovaný *Service Locator* získají konfigurace všech modelů a ID právě aktivního modelu. Druhým krokem je rozesílání události o exportu. Při volání *Callback* funkcí se předává objekt, do kterého mohou komponenty přidávat svou konfiguraci. Naplněný objekt se pak přidá do generovaného výstupu. Celý výstup se pomocí metody `JSON.stringify` převede na text a uloží do souboru. Import probíhá stejným způsobem, ale v opačném pořadí. Konfigurace exportovaná do souboru má 4 KB.

K ukládání exportu do souboru jsem využil modul `FileSaverJS` [20], díky kterému jsem mohl zaručit širokou podporu webových prohlížečů.

5.6 Rozšiřitelnost simulátoru

Při realizaci simulátoru se myslelo i na budoucí rozšiřitelnost simulátoru a v této sekci popíši, jak je možné jednotlivé části simulátoru rozšiřovat.

5.6.1 Přidání nového objektu scény

Prvním jsou objekty (postava, dům, strom atd.), které je možné přidávat do scény. Ty se nacházejí v adresáři `drawable_objects`. Každý nově přidaný objekt dědí od třídy `DrawableObject`, která obsahuje základní metody pro práci s objekty. V definici třídy je potřeba, podobně jako v původní verzi, určit body objektu a hrany mezi body. Pozice bodů (x,y,z) je vhodné udávat pouze v rozsahu od -0.5 do 0.5. V opačném případě by body například přetékaly výběrový rámec (viz obrázek 4.2).

```
export default class House extends DrawableObject {
    vertices: vertices = [
        // bottom
        [0.45, -0.5, -0.45],
        ...
    ];
    edges: Array<[number, number]> = [
        [0, 1], [1, 2], [2, 3], [3, 0],
        ...
    ];
}
```

Výpis 5.3: Ukázka objektu scény.

Poté co je nový objekt založený, je potřeba jej přidat ještě do kolekce dostupných objektů v souboru `ObjectTypes.js` a do metody `createDrawableObject` ve třídě `DrawingHelper.js`.

5.6.2 Přidání nového konfiguračního panelu

Pod pojmem konfigurační panel jsou myšleny jednotlivé komponenty, které se přidávají do komponenty levý panel, jako je například komponenta s nastavením kamery (`CameraConfiguration`). Pro každý konfigurační panel je nejprve nutné vytvořit novou datovou entitu ve složce `model/entities`, která bude obsahovat hodnoty komponentou přidané konfigurace (jako je například ohnisková vzdálenost u komponenty `CameraConfiguration`). Nově přidané entity musí dědit od třídy `Generic`, která opět obsahuje základní metody pro práci s entitami. Kromě metod `get` a `set` pro jednotlivé položky entity, je potřeba ještě do entity doplnit metodu `getCopy`, která bude

vytvářet kopii entity. Metoda `getCopy` se používá například při kopírování konfigurace mezi modely.

Pro položky entity, u kterých se vytváří také GUI kontrolka (není to například jen pomocná proměnná používaná při výpočtech), je potřeba ještě doplnit základní informace o vykreslované položce, jako je například u čísla minimální a maximální dovolená hodnota. K tomu se vytváří nová položka entity, která se definuje pouze jako statická hodnota. Jméno této položky je nutné vytvořit podle jména položky, pro kterou se vytváří, a nakonec přidat sufix `Control`. Obsah této položky tvoří pole o šesti hodnotách (číslech) a každá hodnota má svůj předem daný význam a místo. Pro přístup k jednotlivým hodnotám se používá pomocný objekt `ctlSetLoc`, který dovoluje psát čitelnější kód (nahrazuje používání obyčejného indexu do pole). Pro lepší pochopení je přiložena následující ukázka.

```
// priklad jak ziskat hodnotu polozky entity
getFocalLength(ctlSetLoc.step)
// priklad jak ziskat informace o minimalni
// a maximalni dovolene hodnote polozky entity
getFocalLengthControl(ctlSetLoc.min)
getFocalLengthControl(ctlSetLoc.max)
```

Výpis 5.4: Informace o kontrolce se získávají přes následující objekt.

Dále je potřeba založit nové DAO pro vytvořenou entitu. Tato třída se umisťuje do složky `model`. Každé DAO musí dědit od třídy `GenericDAO` a implementovat tři základní metody: `toJSON`, `fromJSON` a `addNewRecord`. Metody `toJSON` a `fromJSON` se používají během importu a exportu simulace. Metoda `addNewRecord` se používá při zakládání nového modelu. Třídy DAO používají vzor Singleton. Na konci definice každé třídy je nutné vytvořit instanci této třídy, která se používá při importech, a registrovat jí do instance třídy `RegistryService` (viz sekce 5.7). Tím se zajistí, že komponenty, jako je pravý panel, budou mít přístup ke všem entitám simulace.

Nakonec je potřeba přidat novou React komponentu a provést její registraci. Komponentu je potřeba vložit do složky `left_panel_modules`. Základem je vytvořit obsah vykreslované oblasti komponenty v metodě `render`, pomocí syntaxe JSX, a metody zpracovávající uživatelské akce v dané oblasti. Metody zpracovávající uživatelské akce by měly mít prefix `handle` (viz implementace existujících komponent). V metodě `componentDidMount` je nutné komponentu registrovat k odběru událostí `modelSwitch` a `paste` (viz sekce 5.4.1) a vytvořit pro ně odpovídají *Callback* funkce. Při aktualizacích konfigurace komponenty musí komponenta naopak rozesílat událost `configurationUpdate`. Vytvořenou komponentu je také nutné přidat do seznamu `PANEL_COMPONENTS`, který se nachází ve třídě

`LeftPanel`.

5.6.3 Přidání nového pohledu

Pohledy se nacházejí ve složce `simulation_views` a jsou vykreslované komponentou `CenterPanel`. Třída pohledu je opět React komponenta. Komponenta musí reagovat na události `modelSwitch`, `configurationUpdate` a `paste`. Základní funkce, které by měl každý pohled implementovat, jsou zoom, translace, pravítka a změna velikosti pohledu. Další popis už je pouze doporučený, protože nové pohledy mohou mít různé funkce a obsah. Stávající pohledy používají třídu `VizualizationBuilder`, ve které má každý pohled vlastní metodu `render...Visualization`. Tuto metodu volají vždy, když dochází k aktualizaci vizualizace, předávají jí ukazatel na své plátno (HTML element Canvas) a metoda vizualizaci na plátno vykreslí.

5.7 Implementace distribuce událostí

Podle návrhu ze sekce 4.5.3, jsem hledal vhodné řešení k implementaci distribuce událostí. Protože jsem k realizaci použil knihovnu React, tak jsem nejprve uvažoval o použití třídy `Dispatcher` z projektu Flux [25]. Projekt Flux je přidruženým projektem knihovny React. Třída `Dispatcher` je variantou návrhového vzoru Observer. Třída funguje jako prostředník, u kterého se odběratelé (v angl. *Subscribers*) registrují a vydavatelé (v angl. *Publishers*) rozesílají.

Jejich implementaci distribuce událostí však měla z mého pohledu jeden problém. Když chce třída obdržet nějakou událost, musí se u třídy `Dispatcher` nejprve zaregistrovat (předat *Callback* funkci) a poté může začít zpracovávat události. Problém však je, že *Callback* funkce třídy bude dostávat všechny události, bez ohledu na to, jestli je chce přijímat nebo ne. Až vlastním kódem musí rozhodovat, které události bude zpracovávat. V současné verzi v aplikaci mnoho událostí není. Pokud by ale počet událostí začal růst, mohlo by se stát, že aplikace začne mít výkonnostní problémy.

Realizoval jsem proto vlastní verzi třídy `Dispatcher`, u které registrující třída vždy udává, při jaké události se má registrovaná *Callback* funkce volat. Jeho API se skládá ze tří metod: `register`, `unregister` a `dispatch`. Metody `register` a `unregister` používají React komponenty ve svých metodách `mount` a `unmount`.

Metodu `dispatch` mohou komponenty využívat k distribuci události. Metodě se kromě identifikátoru akce ještě předává proměnná (nejčastěji jde o objekt), která je předávána *Callback* funkcím posluchačů. V případě akce

export je proměnná použita jako akumulátor a posluchači do ní předávají konfiguraci určenou k exportu.

5.8 Implementace vzoru Service Locator

Jak bylo v sekci 4.5.5 navrženo, přidal jsem do aplikace třídu `RegistryService`, která implementuje vzor *Service Locator*. Každé DAO se při svém vzniku do třídy registruje pomocí metody `register` a komponenty, jako je pravý panel, mohou využívat metodu `getAll` k získání všech DAO.

5.9 Nové datové typy

Jak bylo zmíněno v kapitole návrhu 4.6.2, v aplikaci je použit nástroj Flow, který provádí statickou typovou kontrolu. Aby nástroj mohl kontrolu provádět, tak bylo potřeba do kódu doplnit potřebné typy. Typ je vždy možné přidávat za proměnou následujícím způsobem: `proměnná: typ`. Flow má nadefinováno mnoho základních typů, které je možné používat. Protože jsem ale chtěl kontrolovat také některé složitější struktury a nechtěl jsem neustále používat (u každé proměnné) složité zápisy, kde jsem musel typ vytvářet z několika primitivních, tak jsem si dostupné typy rozšířil. V souboru `flowTypes.js` se nacházejí všechny nové typy. Nové typy obsahují například definici vektoru s předdefinovanou velikostí (viz výpis 5.5) a typ kamery s předdefinovanou strukturou (viz výpis 5.6). Tyto typy se pak používají stejně jako základní typy od Flow.

```
type vec3 = [number, number, number];
```

Výpis 5.5: Definice vektoru o třech prvcích.

```
type cameraType = {
    location: vec3;
    direction: vec3;
    right: vec3;
    up: vec3;
    aspect: number;
    color: vec3
};
```

Výpis 5.6: Definice typu pro objekt kamery.

5.10 Implementace vykreslování vizualizací

Základní implementace vykreslování byla převzata z původní verze simulátoru a rozdělena do několika souborů (služba a pomocné třídy). Jak bylo výše naznačeno, vykreslování má na starosti třída `VisualizationBuilder`. Ta k vykreslování používá (k vůli lepší orientaci a znovupoužitelnosti) ještě tři pomocné třídy `PhotoTransformationsHelper`, `DrawingHelper` a `VectorOperationsHelper`. Do třídy `VectorOperationsHelper` byly přesunuty všechny metody, které provádějí různé operace s vektory. Ve třídě `PhotoTransformationsHelper` jsou umístěny metody implementující jednotlivé operace postprodukce stereoskopických snímků. Třída `DrawingHelper` obsahuje metody, které zařizují vykreslování jednotlivých elementů simulace (pravítka, objekty, kamery atd.).

Třída `VisualizationBuilder` vykonává tři hlavní úkoly. Vytváří pomocné objekty, které se používají při sestavování vizualizace (např. objekt reprezentující oči). Vykresluje vizualizace na plátna přes `renderX` metody a zjišťuje zda došlo k vybrání objektu v metodě `getSelectedObject`.

5.11 Progressive Web Apps

V této sekci popisuji implementované rozšíření, které se označuje jako *Progressive Web Apps* (PWA). PWA je nový termín, který se objevil někdy okolo roku 2015 a používá se k odlišení webových stránek, které se mohou do určité úrovně chovat jako nativní aplikace. Vývojáři z firmy Google mají podrobný seznam kritérií [15], které by webová stránka měla splňovat, aby jí bylo možné považovat za PWA.

Autoři knihovny React mají pro implementaci PWA také připravené řešení, které jsem využil a upravil pro svou aplikaci. Toto řešení má dvě hlavní komponenty:

1. Kód který zařizuje, že webová stránka používá *Service Worker*.[2]
2. Konfigurační soubor *Manifest*.[3]

I když toto řešení a má realizace nesplňuje všechna kritéria, která si nadefinovala firma Google, a nelze jí, alespoň podle jejich popisu, nazývat PWA, tak už jen díky tomuto rozšíření je možné aplikaci používat jako „nativní“. Například v prohlížeči Chrome je možnost přidat stránku na

[2]Jde o proxy server, který stojí mezi aplikací a internetovým připojením aplikace. Zpracovává HTTP dotazy aplikace a používá *Cache first* stategii.

[3]Manifest je konfigurační, který poskytuje další metadata o aplikaci.

plochu. To aplikaci uloží do počítače a stránku bude možné spouštět i bez připojení k internetu.

Samotný *Service Worker* je implementovaný na straně webového prohlížeče. Kód, který zařizuje jeho aktivaci se nachází v souboru `registerServiceWorker.js` a jeho spouštění probíhá v souboru `index.js`. Jeho úspěšné spuštění můžeme ověřit například tak, že zamezíme přístup k internetu a provedeme opětovně požadavek na webovou stránku. Stránka by se měla opět načíst. Detailní informace lze získat přes vývojové nástroje některých moderních prohlížečů (například Chrome).

Důležitým požadavkem služby *Service Worker* je aplikaci poskytovat pomocí HTTPS. V opačném případě bude *Service Worker* z bezpečnostních důvodů (viz zdroj [16]) vypnutý. V současné době také není podpora služby *Service Worker* příliš rozšířená a v případě nedostupnosti bude služba opět vypnutá.

Zmíněný *Manifest* je potřeba přidat do hlavičky webové stránky (viz kód 5.7).

```
<link rel="manifest" href="...../manifest.json">
```

Výpis 5.7: Připojení souboru manifest.json

Do tohoto souboru (*Manifest*) je možné například uvádět informace o autorovi aplikace, jméno aplikace, krátký popis a další. Důležitým atributem je seznam ikon. V tomto seznamu se nachází ikony různých velikostí pro různá zařízení. V okamžiku, kdy se uživatel mobilního zařízení rozhodne si webovou stránku uložit (například na plochu), je použita ikona právě z tohoto seznamu.

S tím souvisí také další dva důležité atributy a to `orientation` a `display`. Tyto dva atributy se používají právě po uložení aplikace do zařízení. První udává v jaké poloze se má aplikace zobrazit po spuštění. Druhý atribut, při zvolení hodnoty `"standalone"`, otevře aplikaci bez webového prohlížeče. Tedy i bez jeho extra rámečku a aplikace bude opravdu působit jako nativní.

Do souboru je možné zadávat ještě mnoho dalších atributů. Přehledný seznam je možné nalézt například na stránkách Mozilla Developer Center [17]. Navíc je velmi pravděpodobné, že se tento seznam bude do budoucna rozrůstat.

5.12 Sestavení

Protože jsem k vývoji používal knihovnu React, tak jsem i k sestavování aplikace používal nástroj, který vytvořili autoři knihovny. Tento nástroj (balíček) se jmenuje *react-scripts*. Nástroj poskytuje čtyři základní příkazy

(více viz Programátorská dokumentace). Během sestavování aplikace se používá příkaz `react-scripts build`. Ten je uložený v souboru `package.json`, takže je možné v příkazové řádce přímo používat příkaz `npm build`[4].

Po spuštění se provádí slučování, minifikace a obfuskace zdrojových souborů aplikace (z adresáře `src`) a vytváří se jeden soubor `main.js`, který se používá v produkční verzi. Tento soubor má zanedbatelnou velikost, v porovnání se zdrojovými soubory. Při procesu sestavení se přitom používají dva důležité nástroje. Prvním je transpiler Babel, který zdrojové soubory překládá do prohlížečem použitelné verze JavaScriptu. Zdrojové soubory totiž obsahují některé nestandardní syntaktické konstrukce a informace o typech, které je nutné z produkční verze odstranit. Druhým nástrojem je Webpack, který zařizuje složení všech souborů do jednoho výstupního souboru `main.js`.

5.13 Podpora webových prohlížečů

Funkčnost webové aplikace byla ověřována ve všech široce rozšířených webových prohlížečích (Microsoft Edge, Chrome, Firefox, Opera, Safari). Prohlížeč Internet Explorer 11 sice ještě používá nezanedbatelné procento uživatelů, ale už i samotný Microsoft jej nahradil prohlížečem Edge a jeho další vývoj byl ukončen. Proto ani aplikace tento prohlížeč nepodporuje.

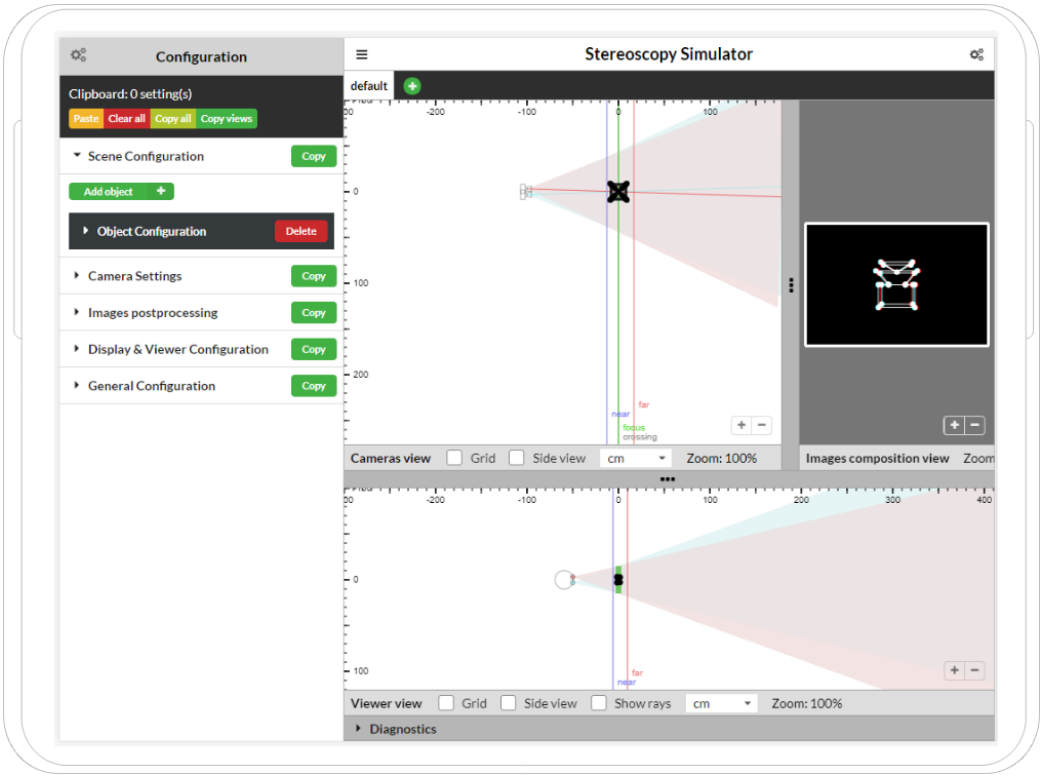
5.13.1 Výkonnostní rozdíly prohlížečů

Samotná simulace neobsahuje žádné časově náročné výpočty, které by mohly vykreslování nějak zpomalovat, a to ani na mobilních zařízeních. Kde se ale projevují výkonnostní problémy, jsou aktualizace webové stránky (DOM). Tento problém je spojený s webovým prohlížečem a způsobem jakým s DOM pracuje. Nejhůře je na tom prohlížeč Edge od Microsoftu, u kterého na tento problém bylo několikrát upozorňováno (viz zdroj [26]). Nejlépe je na tom (podle mého testování) prohlížeč Chrome. Zbylé prohlížeče jsou na tom výkonem podobně jako Chrome (výrazně lépe než Edge). Samotnou funkčnost simulátoru tento problém nijak neovlivňuje, ale u dvou akcí (používání posuvníků pohledů a přímá manipulace objektu scény) používání simulátoru znepříjemňuje. Rozdíl je možné porovnat osobně, například použitím zmíněných posuvníků. V prohlížeči Chrome dochází k naprosto plynulé aktualizaci, kdežto u prohlížeče Edge je posun často trhavý.

[4]NPM je balíčkovací systém, používaný během vývoje.

5.14 Podpora mobilních zařízení

Aplikace byla vyvíjena i s ohledem na mobilní zařízení (tablety a telefony), přesto v tomto případě nebyl volen přístup *Mobile-first*. Důvodem byla primární funkce simulátoru. Tím je vzdělávací aplikace. U aplikace bylo proto žádoucí, aby byly vždy vidět všechny tři pohledy simulace. Což je především na velmi malých displejích problém. Proto jsem si stanovil minimální úhlopříčku zařízení na 7 palců a rozlišení displeje alespoň 1280 x 720 (HD Ready). U displejů s nižším rozlišením nebo úhlopříčkou (především telefony) je teoreticky možné aplikaci také používat, ale práce se simulátorem by mohla být přinejmenším frustrující.



Obrázek 5.2: Ukázka zobrazení aplikace na tabletu s rozlišením 1024x768.

U mobilních zařízeních je také defaultně zavřený panel Diagnostics, aby pro pohledy simulace zbylo více místa. Diagnostiky je ale stále možné normálně otevírat, stejně jako u desktopových prohlížečů. Také, jak bylo zmíněno v sekci 5.11, je možné aplikaci uložit na plochu a spouštět jako „nativní“ aplikaci.

6 Praktické otestování

Posledním bodem práce bylo praktické otestování programu, jehož cílem bylo ověřit použitelnost nové verze programu. I přesto, že navržené a realizované změny v programu se odůvodňovaly a je možné předpokládat, že uživatelům skutečně pomohou, je vhodné vždy uživatelské testování provádět. Jak v článku Usability Test, Even When You Know the Answer [8] autoři uvádějí, testujte i pokud jste si jisti výsledkem (zjednodušeně řečeno).

Při určování vhodného počtu uživatelů, se kterými jsem prováděl testování, jsem vycházel z článku Why You Only Need to Test with 5 Users [7], kde vysvětlují, že není potřeba mít spoustu uživatelů a že je vhodnější vytvořit více menších testů a testovat nanejvýše s 5 uživateli. Protože už jsem neměl k dispozici studenty, kteří absolvují předmět, ve kterém se simulátor používá, tak jsem našel 5 dobrovolníků, kteří se mnou test prováděli. Mezi dobrovolníky byli dva, kteří již měli nějaké zkušenosti s fotografováním, rozuměli základům 3D a byli technicky zdatní. Jeden byl pouze technicky zdatný a zbylí dva měli zkušenosti pouze s fotografováním ale nebyli to žádní technici.

Při sestavování testů jsem vycházel ze zjištěných případů užití a realizovaných úprav. Navrhl jsem 3 úkoly/testy, aby testování pro dobrovolníky nebylo příliš dlouhé. Ani jeden z testů jsem nenavrhoval jako striktní úkol typu krok za krokem, protože jsem chtěl především zapojit kreativitu dobrovolníka a zjistit, jak se bude při používání aplikace skutečně chovat. S každým dobrovolníkem jsem tedy dělal úkoly individuálně. Při vykonávání úkolu jsem je sledoval a zaznamenával jsem jejich chování pro pozdější analýzu. Když uživatel s úkoly skončil, tak jsem jim ještě pokládal tři jednoduché otázky:

1. Chybělo Vám něco v simulátoru?
2. Odstranil byste něco ze simulátoru?
3. Chtěl byste něco v simulátoru změnit?

Cílem otázek bylo především získat nějakou zpětnou vazbu v případě, kdy dobrovolník během testování programu nebyl příliš aktivní nebo nebylo jednoduché odhadnout jeho reakce. Odpovědi jsem si opět zaznamenával a v závěru provedu jejich zhodnocení.

První úkol byl cílený na první dojem z programu. Dobrovolníka jsem podle jeho zkušeností nejprve seznámil s úplným základy 3D fotografování. Samotný simulátor jsem přitom vůbec nezmiňoval. Po seznámení jsem před nimi simulátor otevřel a řekl jim, aby se zkusili v programu zorientovat.

Nechal jsem je si se simulátorem experimentovat a po chvíli jsem je zastavil a zeptal se, jestli by mi dokázali říci, co v simulátoru vidí a jestli by mi jej dokázali nějak popsat. Nestanovoval jsem žádný striktní čas, po kterém bych dobrovolníka zastavil, protože jsem především chtěl vidět, o co se budou postupně zajímat, a každý má trochu jiné tempo. Uživatele jsem pouze případně usměrnil, když se zdálo, že nad drobností tráví mnoho času, aby testování nebylo příliš dlouhé. Snažil jsem se však takové zásahy minimalizovat. U tohoto úkolu jsem si dobrovolníky rozdělil na dvě skupiny. S první skupinou (3 dobrovolníci) jsem úkol vykonával ve staré verzi simulátoru a s druhou skupinou (2 dobrovolníci) v nové verzi. Abych si mohl verze porovnat. Další dva úkoly už jsem prováděl pouze v nové verzi a zmíněným třem dobrovolníkům jsem představil i novou verzi simulátoru, aby u druhého úkolu neměli nevýhodu.

U druhého úkolu už jsem měl konkrétnější požadavek a chtěl jsem, aby si dobrovolníci vyzkoušeli vytvořit snímek čtverce, u kterého vznikne perspektivní zkreslení, a následně jej pomocí dostupných možností postprodukce opravili. V tomto případě už jsem tedy chtěl, aby dobrovolník aktivně pracoval se simulátorem, abych objevil případné problémy, které by mohly vznikat i u jednoduchých činností. Úkol také sloužil jako příprava na poslední úkol.

U posledního úkolu bylo cílem především zapojit nové funkce simulátoru. Zadání bylo opět jednoduché. Vytvořte si scénu, ve které budete mít 3 objekty, různých velikostí, typů a na různých pozicích, tak aby se dal 3D obraz příjemně pozorovat. Poté si založte ještě další scénu, do ní si dle potřeby zkopírujte nastavení z první scény a zkuste vytvořit jiné rozložení, které se bude opět příjemně pozorovat. Hotové nastavení si uložte pro budoucí použití.

6.1 Výsledky testování

Jak jsem zmiňoval, dobrovolníky jsem při práci sledoval a poznamenával si co se simulátorem provádí. Na základě těchto informací jsem sestavil návrh nových funkcí, které by bylo vhodné do další verze programu přidat.

1. Všichni dobrovolníci se snažili různě klikat do jednotlivých pohledů a to především na ilustrace kamer a hlavy. Do další verze programu by tedy bylo vhodné přidat přímou manipulaci s kamerou a hlavou, podobně jako je tomu v nové verzi u objektů.
2. Pohled Viewer dobrovolníkům stále dělal trochu problémy i v nové verzi a dobrovolníci měli problém rozpoznat promítací plochu. Bylo

by pravděpodobně vhodné přidat i jednoduchou ilustraci monitoru (promítacího zařízení), podobně jako se přidala ilustrace hlavy.

3. Dále by bylo vhodné doplnit nějakou výraznou informaci o nastaveném pohledu (top view, side view). Dobrovolníkům působilo trochu problém rozpoznat, že se jedná o 3D scénu, na kterou se dá dívat i z boku.
4. Nepřidávat objekty na stejné počáteční místo. Protože se v nové verzi nové objekty přidávají na stejné počáteční místo, dobrovolníky to zmátlo a nebyli si jistí, zda se nový objekt skutečně přidal.
5. Při stisku tlačítka copy u skupiny nastavení nějak zvýraznit, že došlo k vložení do schrány.

Pozorování uživatelů přineslo i několik zajímavých faktů, na které je dobré při budoucím rozšiřování nezapomenout.

1. Je vhodné v počátečním stavu ve scéně zobrazovat pouze jeden objekt (nejlépe z reálného světa). Některé dobrovolníky ve staré verzi mátl druhý objekt obdélníku, o kterém si mysleli, že ohraničuje jakýsi konec scény.
2. Při pozorování dobrovolníků, kteří měli za úkol se zorientovat v nové verzi programu, se ukázalo, že zavřené záložky jednotlivých karet nastavení jim pomohly se rychleji zorientovat v programu oproti dobrovolníkům, kteří používali starou verzi. V nové verzi se totiž nejprve soustředili na nadpisy jednotlivých skupin a následně si karty začali otevírat, u původní verze se rovnou pustili do jednotlivých kontrolek a pak měli problém zjistit co se kde bude měnit.
3. Ilustrace kamery a obličeje v pohledech skutečně dobrovolníkům pomáhaly rychleji identifikovat účel pohledu.

Kromě vlastních poznámek jsem nasbíral také informace od dobrovolníků při kladení výše zmíněných otázek. Také tyto informace vedly k návrhu nových funkcí a úprav programu. Seznam návrhů přikládám jako přílohu. Kromě zmíněných dobrovolníků, si program prohlížel i odborník, který také vyučuje 3D fotografování a i jeho zpětnou vazbu přidávám do tohoto seznamu.

Zhodnocení

Jak je z předcházejícího textu vidět, praktické testování programu přineslo mnoho návrhů na další rozšiřování programu, které mohou sloužit jako inspirace pro budoucí verze programu. Realizované změny programu se

zdají plnit svůj účel, alespoň na základně provedeného testování. Žádné kritické problémy objeveny nebyly a reakce dobrovolníků byly velmi pozitivní. Dobrovolníků, kteří měli možnosti si vyzkoušet starou i novou verzi programu, jsem se ještě doptával, zda se jim v nové verzi pracovalo lépe. Všichni se jednoznačně shodli, že ano. Po provedeném testování lze tedy prohlásit, že realizace nové verze se zdá být úspěšná a dobrovolníci nová rozšíření přivítali.

7 Zhodnocení a závěr práce

Cílem práce bylo vytvořit novou verzi výukového programu, který by byl především uživatelsky přívětivější. Na základě identifikovaných slabin jsem proto navrhl a realizoval sadu zlepšení. Novou verzi programu jsem následně testoval se skupinou dobrovolníků, abych ověřil skutečnou použitelnost nové verze a na základě zpětné vazby dobrovolníků jsem vytvořil seznam dalších zlepšení, které by bylo vhodné realizovat v dalších verzích programu.

K realizaci nové verze programu jsem vybral nástroje, které se na základě provedené analýzy, ukázaly být vhodnou volbou pro tento typ projektu a měly by být podporovány i v budoucnosti. Během návrhu architektury nové verze a implementace programu jsem také důsledně dbal na budoucí rozšiřování programu. Navržená architektura by měla poskytovat dostatečný prostor pro různá budoucí rozšíření a také orientace v kódu programu by neměla činit větší potíže. Některé možnosti rozšiřování, které budou pravděpodobně využity, jsou popsány i v textu práce.

Věřím, že se povedlo splnit hlavní cíle práce a vznikla nová lepší verze výukového programu, která je uživatelsky přívětivější a při studiu 3D fotografování bude užitečným pomocníkem. Jak už jsem naznačil, samotný program má velký potenciál pro další rozšiřování a i zpětná vazba dobrovolníků naznačuje, že by bylo vhodné simulátor dále rozšiřovat a vytvářet nové verze programu. Také u některých nových rozšíření programu bylo nutné realizovat pouze kompromisní řešení, aby realizace nezabrala příliš mnoho času a byl čas také na jiná rozšíření (viz například možnosti práce s pohledy simulátoru). I v těchto případech by proto bylo vhodné znovu zvážit další rozšiřování.

Literatura

[1] ŽÁRA, Jiří. *Moderní počítačová grafika.* 2. vyd. BENEŠ, Bedřich, SOCHOR, Jiří, FELKEL, Petr. Brno: Computer Press, 2004. ISBN: 80-251-0454-0.

[2] Sanders, Chris. *About Face 3.* 2. vyd. Wiley Computer Publishing, 2007. ISBN: 978-0-470-08411-3.

[3] LOBAZ, Petr. *Mechanismy rozpoznání hloubky.* [online]. Západočeská univerzita v Plzni, Fakulta aplikovaných věd. Poslední změna 2017 [cit. 3.3.2018]. Dostupné z: kiv.zcu.cz/ lobaz/uf3d/web2016/01_uvod/text02_mechanismy

[4] LOBAZ, Petr. *Úvod do 3D fotografování.* [online]. Západočeská univerzita v Plzni, Fakulta aplikovaných věd. Poslední změna 2017 [cit. 3.3.2018]. Dostupné z: kiv.zcu.cz/ lobaz/uf3d/web2016/01_uvod/text01_uvod

[5] LOBAZ, Petr. *Kreativní 3D snímání.* [online]. Západočeská univerzita v Plzni, Fakulta aplikovaných věd. Poslední změna 2018 [cit. 3.3.2018]. Dostupné z: kiv.zcu.cz/ lobaz/uf3d/web2016/02_kreativni3d/text05_kreativni

[6] HARLEY, Aurora. Icon Usability. In: *nngroup.com* [online]. 12.10.2014 [cit. 21.1.2018]. Dostupné z: nngroup.com/articles/icon-usability/

[7] NIELSEN, Jakob. Why You Only Need to Test with 5 Users. In: *nngroup.com* [online]. 19.3.2000 [cit. 21.4.2018]. Dostupné z: nngroup.com/articles/why-you-only-need-to-test-with-5-users/

[8] LORANGER, Hoa. Usability Test, Even When You Know the Answer. In: *nngroup.com* [online]. 28.1.2018 [cit. 21.4.2018]. Dostupné z: nngroup.com/articles/test-when-you-know-answer/

[9] *Comparison of JavaScript frameworks.* [online]. Poslední změna 27.12.2017 [cit. 3.3.2018], Wikipedie. Dostupné z: wikipedia.org/wiki/Comparison_of_JavaScript_frameworks

[10] *Depth of field.* [online]. Poslední změna 7.4.2018 [cit. 28.4.2018], Wikipedie.
Dostupné z: wikipedia.org/wiki/Depth_of_field

[11] *Model–view–controller*. [online]. Poslední změna 9.4.2018 [cit. 28.3.2018], Wikipedie.
Dostupné z: wikipedia.org/wiki/Model–view–controller

[12] *Stereoskopie*. [online]. Poslední změna 18.10.2014 [cit. 28.3.2018], Wikipedie.
Dostupné z: wikipedia.org/wiki/Stereoskopie

[13] *Model–view–presenter*. [online]. Poslední změna 23.4.2018 [cit. 28.3.2018], Wikipedie.
Dostupné z: wikipedia.org/wiki/Model-view-presenter

[14] *Model–view–viewmodel*. [online]. Poslední změna 3.3.2018 [cit. 28.3.2018], Wikipedie.
Dostupné z: wikipedia.org/wiki/Model–view–viewmodel

[15] *Progressive Web App Checklist.* [online]. Google Developers. Poslední změna 17.11.2018 [cit. 28.3.2018].
Dostupné z: developers.google.com/web/progressive-web-apps/checklist

[16] *Service Worker API.* [online]. Mozilla Developer Network and individual contributors. Poslední změna 27.3.2018 [cit. 28.3.2018]. Dostupné z: mozilla.org/en-US/docs/Web/API/Service_Worker_API

[17] *Web App Manifest.* [online]. Mozilla Developer Network and individual contributors. Poslední změna 28.1.2018 [cit. 2.4.2018]. Dostupné z: mozilla.org/en-US/docs/Web/Manifest

[18] Igor Minar MVC vs MVVM vs MVP. In: *plus.google.com* [online]. 19. 7. 2012 [cit. 28.3.2018]. Dostupné z: plus.google.com/+AngularJS/posts/aZNVhj355G2

[19] Andrew Rota *Component Based UI Architectures for the Web.* [online]. SlideShare, 2017 [cit. 28.3.2018]. Dostupné z: slideshare.net/andrewrota/component-based-ui-architectures-for-the-web

[20] Eli Grey. *FileSaver.js.* [online]. GitHub Inc. Poslední změna 26.10.2018 [cit. 3.3.2018]. Dostupné z: github.com/eligrey/FileSaver.js

[21] Brian Payne. *Github Compare.* [online]. Brian Payne ©2014 [cit. 3.3.2018]. Dostupné z: github.io/github-compare

[22] *React – Dokumentace.* [online]. Facebook Developers. Facebook Inc. Poslední změna 13.4.2018 [cit. 3.3.2018]. Dostupné z: reactjs.org/

[23] *React – Optimizing Performance.* [online]. Facebook Developers. Facebook Inc. Poslední změna 16.1.2018 [cit. 3.3.2018]. Dostupné z: reactjs.org/docs/optimizing-performance

[24] *React – Introducing JSX.* [online]. Facebook Developers. Facebook Inc. Poslední změna 17.2.2018 [cit. 28.3.2018]. Dostupné z: reactjs.org/docs/introducing-jsx

[25] *Flux – Dokumentace.* [online]. Facebook Developers. Facebook Inc. Poslední změna 26.1.2018 [cit. 28.3.2018]. Dostupné z: facebook.github.io/flux/

[26] Edge performance with DOM fragments is 7x slower than Chrome. In: *developer.microsoft.com.* [online]. Poslední změna 12.9.2015 [cit. 3.3.2018]. Dostupné z: developer.microsoft.com/en-us/microsoft-edge/platform/issues/4561410/

A Programátorská dokumentace

V této příloze jsou popsány některé časté úkoly, které je nutné během vývoje vykonávat. Během vývoje byl používaný balíčkovací systém NPM ve verzi 5.6. I u vyšších verzí by však neměl být problém. Alternativou je použít Yarn. Další informace je také možné najít v souboru `README.md`, který se nachází v kořenovém adresáři aplikace.

A.1 Proces vývoje

Prvním krokem je instalace závislostí projektu. To lze provést následujícím příkazem (ukázky jsou z operačního systému Windows).

```
> cd stereoscopy-simulator
> npm install
```

Výpis A.1: Instalace závislostí programu.

Po úspěšné instalaci se všechny závislosti uloží do adresáře `node_modules`. Přehled závislostí programu se nachází v souboru `package.json` (viz A.2). Protože vypsané závislosti mohou mít také své závislosti, tak adresář `node_modules` typicky obsahuje více modulů, než je uvedeno v `package.json`.

```
{
  "name": "stereoscopy-simulator",
  "version": "1.0.3",
  "private": true,
  "dependencies": {
    "file-saver": "1.3.3",
    "flow-bin": "0.68.0",
    "react": "16.2.0",
    "react-dom": "16.2.0",
    "react-rangeslider": "2.2.0",
    "react-scripts": "1.1.1",
    "semantic-ui-css": "2.2.14",
    "semantic-ui-react": "0.78.2"
  },
  "homepage": "/stereoscopy-simulator/build",
  "scripts": {"flow": "flow",
    "start": "react-scripts start",
    "build": "react-scripts build"
  }
}
```

Výpis A.2: Konfigurační soubor package.json.

Po instalaci potřebných závislostí je možné spustit vývojovou verzi aplikace následujícím příkazem.

```
> npm start
```

Výpis A.3: Spuštění vývojové verze.

V případě úspěšného spuštění se v konzoli objeví podobný výpis informující o úspěšném spuštění a portu na kterém aplikace běží.

```
"C:\Program Files\JetBrains\WebStorm 2017.3.4\bin\runnerw.exe"
"C:\Program Files\nodejs\node.exe"
"C:\Program Files\nodejs\node_modules\npm\bin\npm-cli.js"
run start --scripts-prepend-node-path=auto

> stereoscopy-simulator@1.0.3 start D:\stereoscopy-simulator
> react-scripts start

Starting the development server...

Compiled successfully!

You can now view stereoscopy-simulator in the browser.

  Local:            http://localhost:3000/
  On Your Network:  http://192.168.0.80:3000/

Note that the development build is not optimized.
To create a production build, use npm run build.
```

Výpis A.4: Úspěšné spuštění programu.

Vývojová verze aplikace je výhodná především, pokud používáme nějaké chytré vývojové prostředí (jako je WebStrom), protože je možné mnoho věcí ještě dále zjednodušit (například samotné zastavení/spouštění aplikace). U vývojové verze se také monitorují změny v souborech aplikace. Když provedeme uložení změn, tak se tyto změny automaticky projeví ve spuštěné instanci aplikace. Do konzole se také vypisují varovná hlášení knihovny React. Protože byla do aplikace přidána statická typová Flow, tak je možné používat následující příkaz ke spuštění kontroly. Nalezené chyby se pak vypisují přímo do konzole.

```
> npm flow
```

Výpis A.5: Spouštění typové kontroly.

A.2 Proces nasazení

Pokud je připravena nová verze aplikace, tak je vhodné nejprve aktualizovat verzi v souborech `RightPanel.js` a `package.json`. Následně nastavit správnou relativní cestu k souborům (viz výpis A.6). Například, pokud bude aplikace umístěna v kořenovém adresáři hostingu, tak stačí nahradit `"/stereoscopy-simulator/build"` za `"/"`, nebo parametr ze souboru úplně smazat.

```
...
"homepage": "/stereoscopy-simulator/build",
...
```

Výpis A.6: Změna relativní cesty v package.json.

Nakonec je nutné spustit příkaz A.7 v kořenovém adresáři projektu. Tento příkaz spouští proces popsaný v sekci 5.12.

```
> npm build
```

Výpis A.7: Spouštění procesu sestavení produkční verze.

Produkční verze aplikace se vygeneruje do adresáře `build`. Odtud je možné aplikaci rovnou nasadit.

B Seznam návrhů dalších úprav programu

Tento seznam obsahuje návrhy úprav simulátoru, které byly nasbírány od dobrovolníků během praktického testování.

1. Několik účastníků zmínilo, že by chtěli pro zoom používat kolečko myši.
2. U objektu by chtěli místo nastavování poměru rozměrů, klasickou možnost nastavit výšku, šířku a hloubku.
3. Chtěli by mít možnost zobrazit snímek z Images composition view v reálné velikosti, podle nastavených vlastností displeje.
4. Mít možnost si například uložit nastavení kamer do nějakého textového dokumentu, který by si mohli vytisknout a použít ho během nastavování skutečné sestavy kamer.
5. Mít možnost si přeskládat pohledy dle potřeby.
6. Mít možnost vybraný pohled dočasně zavřít.
7. Při práci s více scénami mít možnost si například dvě scény zobrazovat vedle sebe.
8. Mít možnost měnit pořadí vytvořených scén (přesouvat jako záložky v prohlížečích).
9. Dovolit kopírovat mezi scénami vybrané skupiny parametrů (např. vybrat 5 parametrů), nejen po skupinách, jak je tomu teď.
10. Mít možnost vidět co ve schránce je a dle potřeby také jednotlivé položky vyhazovat.
11. Rotovat piktogramy kamer, aby znázorňovaly směr pohledu kamery.
12. Přidat možnost mezi scénami sdílet, některé části konfigurace aby nebylo nutné je opakovaně nastavovat (či kopírovat).
13. Po otevření záložky objektu v levém panelu také označit patřičný objekt v pohledu scény.
14. Realizovat nějaký systém zvýrazňování změn v simulátoru. Například, když se hýbe s objektem, tak zvýrazňovat vše co se kde mění.
15. Zlepšit práci s kontrolním rámečkem objektu. Například přidat možnost pohybovat s objektem pouze po osách.
16. Barevně odlišit jednotlivé kategorie nastavení pro rychlejší orientaci.

17. Mít možnost si v pohledu Images composition view zobrazit pouze pohled levého či pravého oka.

C Použité zkratky

1. MVC – Model–view–controller.
2. MIT – Massachusetts Institute of Technology
3. AJAX – Asynchronous JavaScript and XML
4. SPA – Single-page Application
5. HMVC – Hierarchical Model–view–controller
6. MVVM – Model–view–viewmodel
7. MVP – Model–view–controller
8. SoC – Separation of Concern
9. MVW – Model-view-whatever
10. HTML – Hypertext Markup Language
11. DOM – Document Object Model
12. CBA – Component-based Architecture
13. API – Application Programming Interface
14. ORM – Object-relational Mapping
15. DAO – data access object
16. DI – Dependency Injection
17. URL – Uniform Resource Locator
18. JSON – JavaScript Object Notation
19. JSX – JavaScript Syntax eXtension
20. XML – Extensible Markup Language
21. CSS – Cascading Style Sheets
22. UI – User Interface
23. UX – User Experience

24. HTTP(S) – Hypertext Transfer Protocol (Secure)
25. PWA – Progressive Web Apps