VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VIRTUÁLNÍ PROHLÍDKA VE VR

VIRTUAL TOUR IN VR

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. LUKÁŠ PELÁNEK**
AUTHOR

VEDOUCÍ PRÁCE **prof. Ing. ADAM HEROUT, PhD.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2018/2019

Zadání diplomové práce

Student: **Pelánek Lukáš, Bc.**
Program: Informační technologie Obor: Počítačová grafika a multimédia
Název: **Virtuální prohlídka ve VR**
Virtual Tour in VR
Kategorie: Uživatelská rozhraní

Zadání:

1. Seznamte se s problematikou virtuální reality a vývoje pro GearVR.
2. Vyhledejte a analyzujte dostupné aplikace zobrazující všesměrné fotografie a digitalizované světy.
3. Navrhujte a prototypujte dílčí prvky zobrazení všesměrných fotografií a jejich sekvencí na GearVR. Testujte prvky zobrazení a interakce na uživatelích a iterativně je vylepšujte.
4. Navrhněte aplikaci, která umožní na GearVR přirozeným způsobem prohlížet digitalizovaný svět.
5. Demonstrujte aplikaci na vhodných datech, například na digitalizovaném areálu FIT.
6. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- Charles Palmer, John Williamson: Virtual Reality Blueprints: Create compelling VR experiences for mobile and desktop, ISBN: 978-1786462985
- Steve Krug: Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability, ISBN: 978-0321965516
- Steve Krug: Rocket Surgery Made Easy: The Do-It-Yourself Guide to Finding and Fixing Usability, ISBN: 978-0321657299

Při obhajobě semestrální části projektu je požadováno:

- Body 1 a 2, značné rozpracování bodů 3 a 4.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Herout Adam, prof. Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 1. listopadu 2018

Abstrakt

Tato diplomová práce se zabývá tvorbou virtuálních prohlídek ve virtuální realitě. Věnuje se panoramatickým fotografiím a jejich využití při tvorbě virtuálních prohlídek. V rámci této práce byly navrhnuty a implementovány dvě aplikace. První z nich jako webový editor pro vytváření a druhá jako mobilní aplikace pro prohlížení virtuálních prohlídek. Vytvořené řešení poskytuje komplexní nástroj pro práci s virtuálními prohlídkami.

Abstract

This thesis deals with the making of virtual tours in the virtual reality. It deals with panoramic photographs and their use in creating virtual tours. In this thesis, two applications were designed and implemented. The first one as a web editor for creating and the other as a mobile application for viewing virtual tours. The created solution provides a comprehensive tool for working with virtual tours.

Klíčová slova

Virtuální realita, virtuální prohlídka, Unity, Samsung Gear VR, panoramatická fotografie, Laravel, Vue.js, Three.js

Keywords

Virtual reality, virtual tour, Unity, Samsung Gear VR, panoramic photography, Laravel, Vue.js, Three.js

Citace

PELÁNEK, Lukáš. *Virtuální prohlídka ve VR*. Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Adam Herout, PhD.

Virtuální prohlídka ve VR

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana prof. Ing. Adama Herouta, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Lukáš Pelánek
22. května 2019

Poděkování

Chtěl bych poděkovat vedoucímu mé diplomové práce panu prof. Ing. Adamu Heroutovi, Ph.D. za pomoc a cenné rady při konzultacích.

Obsah

Kapitola 1

Úvod

Virtuální realita existuje již poměrně dlouhou dobu, avšak teprve v posledních letech začala dosahovat značného růstu na popularitě. Jedním z možných využití virtuální reality je právě tvorba virtuálních prohlídek, kterými se tato práce zabývá.

Cílem práce je navrhnout a implementovat komplexní řešení pro vytváření a prohlížení virtuálních prohlídek. Práce je členěna do osmi kapitol. Kapitola 2 se věnuje panoramatickým fotografiím a jejich využití při tvorbě virtuálních prohlídek. Dále jsou zde zmíněny některé grafické enginy pro tvorbu aplikací ve virtuální realitě, zařízení pro pořízení 360° panoramat, zařízení pro zobrazení virtuální reality a popis některých existujících aplikací s virtuálními prohlídkami.

Kapitola 3 popisuje grafický engine *Unity*, který byl použit k implementaci aplikace pro prohlížení virtuálních prohlídek. V kapitole je popsán koncept vývoje v *Unity*, práce s *Unity* editorem a prvky, které byly použity při implementaci. V kapitole 4 jsou popsány webové technologie, které byly použity k implementaci aplikace pro vytváření virtuálních prohlídek. V této kapitole jsou popsány frameworky *Laravel* a *Vue.js* a knihovna *Three.js*.

Návrhu řešení přecházela série experimentů, které jsou popsány v kapitole 5. Tyto experimenty se skládaly ze zobrazení snímků v headsetu, uživatelské interakce pro přechod mezi snímky a přechodu mezi snímky pomocí transformace kamery a lineární interpolace. Návrh řešení je popsán v kapitole 6 a je rozdělen na návrh aplikace pro vytváření virtuálních prohlídek a návrh aplikace pro prohlížení virtuálních prohlídek.

V kapitole 7 je popsána implementace a vyhodnocení. Implementace je rozdělena na dvě části a popisuje implementaci aplikace pro vytváření a aplikaci pro prohlížení virtuálních prohlídek. Vyhodnocení zahrnuje porovnání vytvořené prohlídky, uživatelské testování a popisuje pokračování ve vývoji. Závěr se nachází v kapitole 8 a jsou v něm rozebrány dosažené výsledky a zhodnocení práce.

Kapitola 2

Virtuální realita a prohlídky

Virtuální realita je skvělým nástrojem pro vytváření prohlídek, protože uživatel nabývá dojmu, že se na daném místě skutečné nachází. V této kapitole jsou rozebrány panoramatické fotografie a jejich typy, které lze využít k tvorbě virtuálních prohlídek. Dále jsou v této kapitole uvedeny nástroje pro pořízení panoramatických fotografií a zařízení pro zobrazení virtuální reality. V poslední části této kapitoly jsou uvedeny některé grafické enginy, které lze použít pro vývoj aplikací ve virtuální realitě.

2.1 Virtuální prohlídka

Virtuální prohlídku lze charakterizovat jako interaktivní prezentaci prostoru, obvykle složenou ze sekvence videosnímků nebo statických obrázků. Tyto obrázky jsou obvykle pořízeny jako panoramatické fotografie, které jsou následně převedeny do virtuální prohlídky. Na rozdíl od simulace se jedná o prezentaci reálně nasnímaných místností nebo exteriéru.

2.2 Panoramatické fotografie

Panoramatickou fotografii lze chápat jako širokoúhlý snímek (vertikálně či horizontálně), nejčastěji krajiny nebo jakéhokoliv jiného celku, který obvykle vznikne složením či vrstvením několika fotografií, které na sebe jednou stranou navazují. Panoramatický formát fotografie má velkou výhodu v tom, že dokáže zachytit větší šířku záběru, a proto je vhodný pro tvorbu virtuálních prohlídek.

Proces skládání fotografií [25] [20] je velice důležitým krokem při vytváření panoramatických fotografií. Kombinací dvou a více vzájemně se překrývajících fotografií vznikne výsledné panorama. Tohoto kroku je obvykle dosaženo pomocí speciálních nástrojů, které slouží ke skládání fotografií. Mezi tyto nástroje patří například komerční *PTGui*[1] nebo nekomerční *Hugin*[2].

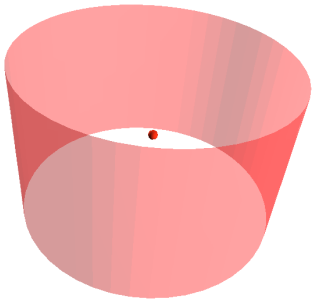
Obrázek 2.1: Několik složených fotografií tvořící panorama [20].

[1] PTGui - https://www.ptgui.com
[2] Hugin - http://hugin.sourceforge.net

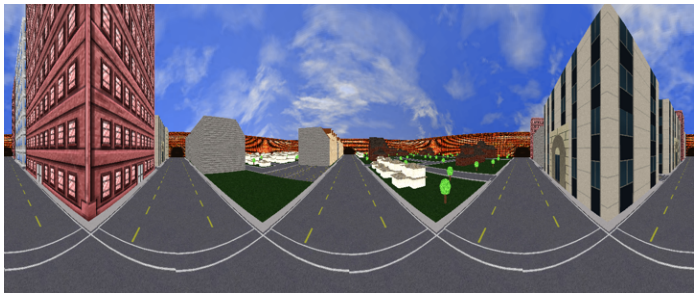
Panoramatické fotografie lze rozdělit na více druhů, které se liší jak způsobem pořízení, tak i způsobem, jakým jsou interpretovány. Druhy panoramatických fotografií, které se nejčastěji používají k tvorbě virtuálních prohlídek, jsou rozebrány v následujících podkapitolách.

2.2.1 Cylindrické panoramatické fotografie

Cylindrickou panoramatickou fotografii [24] [19] lze interpretovat jako panoramatickou fotografii zabalenou do tvaru válce, přičemž pozorovatel se nachází uvnitř válce. Dokáže zachytit celých 360° horizontálně, avšak vertikálně je omezena na 120°. Z toho důvodu není vhodná pro vertikálně orientovaná panoramata, protože vrchní a spodní část zcela chybí, přičemž velikost chybějících částí je závislá na výšce válce.



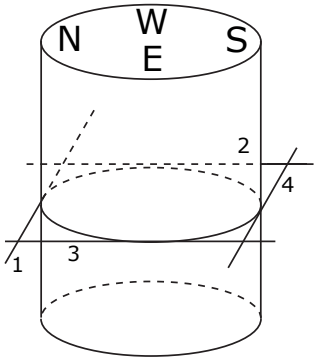
(a) Cylindrická projekce.



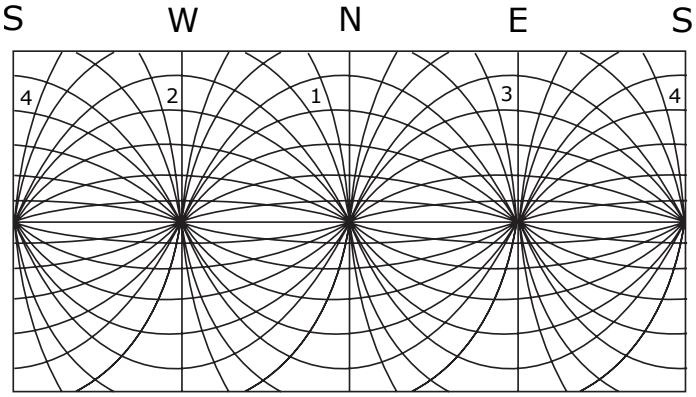
(b) Cylindrický panoramatický obrázek.

Obrázek 2.2: Projekce na plášť válce (a) a interpretace obrázku, který je promítán (b) [19].

Cylindrická projekce nezachovává linearitu objektů, tedy přímka se obecně nezobrazuje na přímku. Zachovává rovnost vertikálních linií, ale dochází k zakřivení linií horizontálních. Speciální případ nastává v momentě, kdy horizontální linie prochází středem pláště válce a je vodorovná. V takovém případě k zakřivení horizontální linie nedochází.



(a) Válec rozdělený na segmenty.

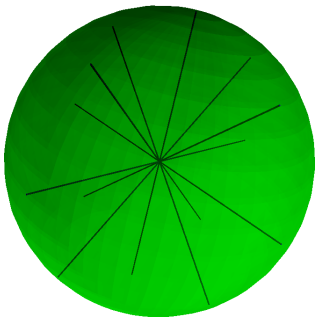


(b) Cylindrická projekce horizontálních segmentů.

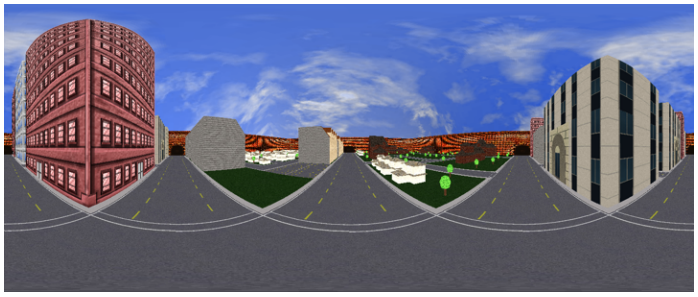
Obrázek 2.3: Ukázka zakřivení horizontálních linií obrázku (b), který je mapován na válec rozdělený na 4 segmenty (a) [24].

2.2.2 Sférické panoramatické fotografie

Projekci sférické panoramatické fotografie [24] [19] si lze představit jako průhlednou kouli, která je umístěna okolo pozorovatele. Všechno, co pozorovatel vidí skrze kouli, je promítáno na vnitřní stranu koule. Koule je následně určitým způsobem zploštěna, čehož výsledkem je plná 360° sférická perspektiva. Stejně jako cylindrická projekce dokáže zachytit 360° horizontálně, ale na rozdíl od ní až 180° vertikálně.



(a) Sférická projekce.



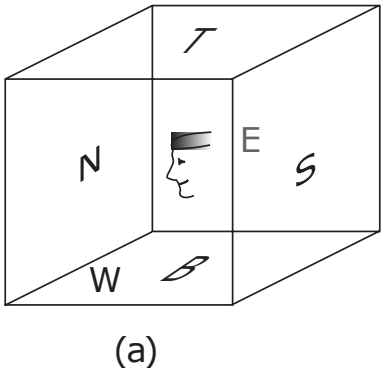
(b) Sférický panoramatický obrázek.

Obrázek 2.4: Projekce obrázku (b) na kouli (a). Linie na povrchu koule (a) značí pásmo, které lze použít jako sloupec obrázku [19].

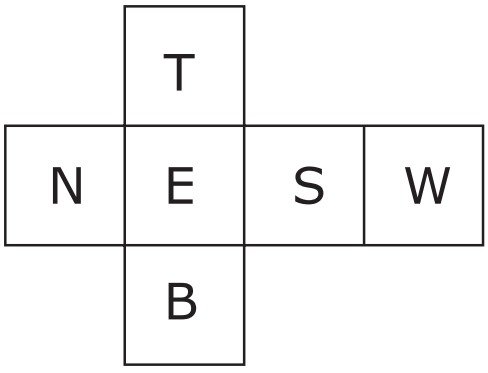
Výhodou sférické projekce je ta, že každý bod je od pozorovatele stejně vzdálen. Dochází k zakřivení horizontálních linií a rovněž k mírnému zakřivení vertikálních linií, obzvláště u podnožníku a nadhlavníku. V těchto částech koule jsou objekty roztaženy a působí detailněji.

2.2.3 Krychlové panoramatické fotografie

Princip krychlové panoramatické projekce je podobný předchozím zmíněným. Pozorovatel je umístěn do středu krychle, skrze kterou vnímá okolní svět. Všechno, co pozorovatel vidí skrze krychli je promítáno na vnitřní stěny krychle, která je následně zploštěna na 6 čtverců, které jsou spojeny. Tím je zajištěno plné 360° panorama.



(a) Krychlová projekce.



(b) Krychlový panoramatický obrázek.

Obrázek 2.5: Projekce obrázku (b) na stěny krychle (a) [24].

Největší výhodou krychlové projekce je ta, že zachovává linearitu objektů. To znamená, že nedochází k zakřivení horizontálních ani vertikálních linií.

2.3 Zařízení pro pořízení 360° panoramat

Existuje více způsobů, jak pořídit 360° panoramatickou fotografii. Jedním z možných způsobů je pořízení fotografií klasickým fotoaparátem s objektivem pomocí stativu a panoramatické hlavy. Poté se vytvořené fotografie složí pomocí speciálního nástroje a vznikne panoramatická fotografie.

Další možností je využití všesměrné kamery pro sférické snímání. Pořízení snímků tímto způsobem je mnohem rychlejší a méně náročné. Příkladem všesměrné kamery je *Samsung Gear 360*. Tato kamera je vybavena dvěma objektivy, kdy každý snímá záběry ve 180° v horizontální i vertikální rovině, čímž vzniká úplné 360° zorné pole.

Obrázek 2.6: Samsung Gear 360 [8].

2.4 Zařízení pro zobrazení virtuální reality

Pro zobrazení virtuální reality se používájí tzv. headsety. Tato zařízení umožní uživateli ponořit se do virtuálního světa nebo sledovat 360° videa. Headsety obvykle vyžadují další zažízení k tomu, aby mohly fungovat. Například smartphone, počítač nebo herní konzoli. V této kapitole jsou popsány některé druhy headsetů, které lze v současnosti najít na trhu.

2.4.1 Headsety nižší třídy

Jedná se o nejdostupnější headsety. Většinou mají jednoduchý design a cena bývá velmi příznivá. Především proto, že bývají vyrobeny pouze z páru čoček a levného materiálu. Nicméně nízká cena je vykoupena tím, že pro použití takového headsetu je nutný smartphone, který podporuje virtuální realitu.

Asi nejvýznamnějším zástupcem této kategorie je *Google Cardboard* [2], který představila společnost *Google* roku 2014. Headsety tohoto typu obvykle v sobě nemají žádný zabudovaný hardware. Některé, jako například právě *Google Cardboard*, mají malé tlačítko, které slouží k emulaci kliknutí uživatele na displej obrazovky.

Obrázek 2.7: Google Cardboard [2].

2.4.2 Headsety střední třídy

Tyto headsety jsou finančně nákladnější než předchozí skupina, ale disponují dalšími funkcemi jako například zabudované ovládací prvky, kolečko pro zaostření obrazu či sledovací senzor. Stejně jako u předchozí skupiny je potřeba smartphone, který podporuje virtuální realitu.

Mezi nejznámější zařízení v této kategorii patří *Gear VR* [9] od společnosti *Samsung*. Výhodou tohoto headsetu je, že se k němu dá připojit ovladač, který rozšiřuje způsoby interakce a tím nabízí nové možnosti využití headsetu. Naopak nevýhodou tohoto zařízení je podpora pouze vybraných modelů od této společnosti.

Obrázek 2.8: Samsung Gear VR [9].

2.4.3 Headsety vyšší třídy

Zařízení z této kategorie nabízí to nejlepší, co se dá v současné době pořídit. Pro jejich fungování není vyžadován smartphone, ale počítač nebo herní konzole. Bývají finančně mnohem náročnější než předchozí uvedené typy, ale na druhou stranu disponují mnohem pokročilejší funkcionalitou jako například sledování pohybu, větší rozlišení obrazovky nebo lepší grafika.

Jako příklad headsetu z této kategorie lze uvést *Oculus Rift* [7], který je výsledkem crowdfundingové kampaně z roku 2012. Následně byl roku 2014 odkoupen společností *Facebook*. Díky spolupráci se společností *Samsung* vznikl roku 2015 již zmíněný *Samsung Gear VR*. Nevýhodou těchto headsetů bývá, že jsou s propojeným zařízením obvykle spojeny pomocí kabelu, čímž je uživatel limitován.

Obrázek 2.9: Oculus Rift [7].

2.5 Vývoj aplikací ve virtuální realitě

Nejefektivnějším způsobem, jak začít s vývojem aplikací ve virtuální realitě, je využít některý z dostupných grafických enginů, které nabízí vývojářům framework pro pohodlnější práci s virtuální realitou. Grafické enginy již často obsahují SDK pro virtuální realitu, jenž dovoluje vývojářům navrhnout, postavit a otestovat své aplikace. Zároveň umožňují vytvořit aplikaci pro různá cílová zařízení, včetně herních konzolí a smartphonů. V této kapitole jsou popsány některé grafické enginy, které lze využít pro tvorbu aplikací ve virtuální realitě.

2.5.1 Unreal Engine

První verze *Unreal Engine* [14] byla uvedena roku 1998 společností *Epic Games*. V roce 2012 byla představena zatím nejnovější verze *Unreal Engine 4*. Za zmínku stojí, že tato společnost stojí za vznikem herních titulů jako je například *Fortnite* či *Unreal Tournament* za použití tohoto enginu. V dnešní době se řadí mezi nejpoužívanější enginy.

Vyvíjení aplikací pomocí tohoto enginu je ve srovnání s konkurenčnímy enginy komplikovanější, ale přináší vývojáři spoustu možností. *Unreal Engine* je vhodný především pro velké projekty a zkušenější vývojáře, protože pro skriptování je nutná znalost programovacího jazyka C++.

Unreal Engine se vyznačuje především svým extrémně flexibilním a mocným Blueprints Visual Scripting systémem. Jde o kompletní skriptovací systém hratelnosti založený na propojování objektů a jejich závislostí uvnitř Unreal Editoru.

2.5.2 Unity

Unity [12] je grafický engine vyvíjený společností *Unity Technologies*. První verze byla vydána roku 2005 a jeho obrovskou výhodou je velká multiplatformní podpora. V *Unity* byly vyvinuty hry jako například *Hearthstone* nebo *Assassin's Creed: Identity*. I přesto, že po technické stránce nedosahuje kvalit jako například *CryEngine* nebo *Unreal Engine*, tak jde o velmi kvalitní a rychle vyvíjející se engine s masivní vývojářskou komunitou.

V současné době společnost *Unity Technologies* na svých stránkách uvádí podporu pro více než 25 platforem. Díky velké komunitě existuje i velké množství tutoriálů, a tak začít s vývojem aplikací pomocí *Unity* lze prakticky okamžitě. Zároveň se na oficiálním fóru

zapojují do diskuse i samotní vývojáři z týmu *Unity*, a tak není problém získat na jakoukoliv otázku relevantní odpověď.

Architektura vývoje v *Unity* je založena především na vytváření objektů z komponent, které implementují určitá relativně modulární chování. To umožňuje vývojářům vytvářet obecné prefabrikované komponenty a sdílet je mezi sebou. Velké množství takových komponent lze najít na *Unity Asset Store*. Skriptovacím jazykem Unity je C#.

2.5.3 CryEngine

Za vznikem grafického enginu *CryEngine* [1] stojí společnost *Crytek*. První vydání proběhlo roku 2002. Z uvedených enginů je nejméně populární, ale byl již využit k vývoji her, které se pyšnily nejpokročilejší grafikou za dob svého vydání. Mezi takové hry patří například *Crysis*, *Far Cry* nebo *Evolve*.

CryEngine je konkurentem populárního *Unreal Engine*. Skriptovací jazyk je rovněž C++, ale v porovnání s *Unreal Engine* není tolik intuitivní a vyžaduje delší čas pro pochopení k efektivnímu využití. Předností *CryEngine* je real-time SandBox Editor, který umožňuje vývojářům modifikovat aplikaci za běhu a tím značně urychlit vývoj.

2.6 Dostupné aplikace s virtuálními prohlídkami

V této kapitole jsou popsány některé dostupné aplikace, které slouží k zobrazování všesměrných fotografií a virtuálních prohlídek. Existuje celá řada aplikací a liší se jak kvalitou provedení, tak i funkcionalitou, kterou nabízí.

2.6.1 Google Street View

Mezi nejznámější webové aplikace zobrazující všesměrné fotografie patří *Street View* [3] od společnosti *Google*, která je součástí služeb *Google Maps* a *Google Earth*. Tato aplikace je volně dostupná a poskytuje interaktivní panoramatické snímky z různých částí světa.

Aplikace obsahuje takzvanou inteligentní navigaci [22], která osvobozuje uživatele od používání obyčejných šipek vpřed a zpět. K vytvoření tohoto prvku využila společnost *Google* při pořizování snímků speciální lasery, které zjišťují vzdálenost od objektů a GPS jednotky pro získání polohy. Na základě těchto informací a vzdálenostech mezi po sobě jdoucími snímky, byli schopni zrekonstruovat geometrii silnic. Výsledkem je navigace mezi jednotlivými snímky pomocí kliknutí na libovolné místo nebo objekt, který chce uživatel vidět.

Mezi další prvky, které aplikace nabízí, patří přiblížení kamery na bod ve scéně a rozhlížení se ve scéně okolo bodu, na kterém se uživatel právě nachází. Dalším zajímavým prvkem je prohlížení si historie snímků, a tak se uživatel může podívat, jak konkrétní místo vypadalo před několika lety. Na některých snímcích lze pozorovat chyby, které vznikly při skládání jednotlivých fotografií.

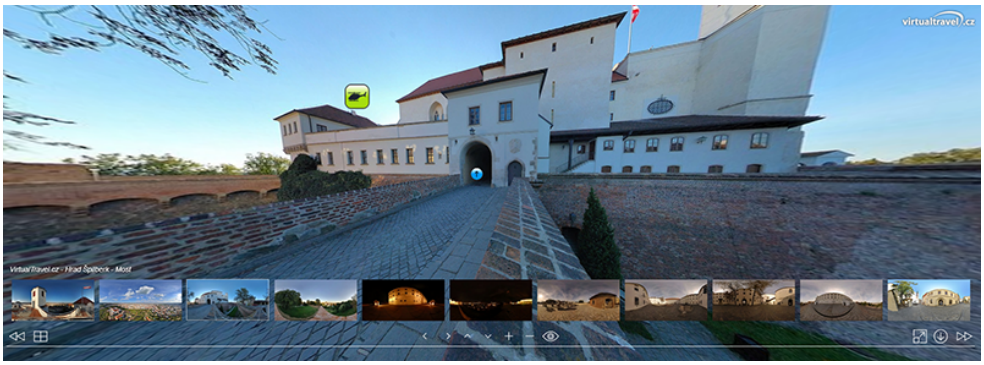
Obrázek 2.10: Pohled na Fakultu informačních technologií VUT v Brně z Google Street View. V levném horním rohu se nachází adresa, na které se uživatel nachází a ovládací prvky zahrnující možnost zobrazení historie aktuálního snímku. V levém dolním rohu je zobrazena aktuální pozice na mapě a v pravém dolním rohu se nachází kompas a tlačítka pro přiblížení scény.

2.6.2 Virtual Travel

Virtual Travel [15] je česká webová aplikace, která zprostředkovává virtuální prohlídky turistických cílů, měst, obcí či firem. Jedná se o komerční aplikaci a společnost, která tuto aplikaci provozuje, vytváří zákazníkům prohlídky za určitou finanční částku na míru.

Aplikace umožňuje uživateli přecházet mezi jednotlivými snímky pomocí jednoduchých šipek. Zároveň některé prohlídky nabízí i letecký pohled na vybrané místo. Uživatel má možnost rozhlížet se ve scéně okolo bodu, na kterém se právě nachází a přiblížit kameru na určitý bod ve scéně. Aplikace dále umožňuje přepnutí do režimu virtuální reality a prohlédnout si prohlídku v headsetu.

Pořízené snímky jsou ve vysoké kvalitě a bezchybně složeny, a tak ve scéně nejsou viditelné chyby, které vznikají při skládání jednotlivých fotografií. Při nečinnosti se kamera samovolně otáčí a tato možnost se nedá vypnout, což může na uživatele působit negativně.



Obrázek 2.11: Pohled na hrad Špilberk v Brně z Virtual Travel [16]. Uprostřed scény se nachází modré tlačítko pro přechod na další snímek a vlevo nahoře od tlačítka pro přechod se nachází zelené tlačítko pro letecký pohled. Ve spodní části se nachází lišta s dalšími prohlídkami a ovládacími prvky včetně tlačítka pro přepnutí do režimu virtuální reality.

2.6.3 Krpano Panorama Viewer

Krpano Panorama Viewer [4] je výkonný a velmi flexibilní prohlížeč pro všechny druhy panoramatických snímků a interaktivních virtuálních prohlídek. Tento prohlížeč je dostupný buď jako Flash nebo HTML5 aplikace. Součástí aplikace je sada nástrojů pro zpracování panoramatických fotografií, ze kterých lze následně vytvořit virtuální prohlídku.

Tento nástroj je vhodný spíše pro pokročilejší vývojáře. Virtuální prohlídky se tvoří pomocí externích konfiguračních souborů. Tyto soubory jsou popsány značkovacím jazykem XML a celá tvorba virtuálních prohlídek je podrobně popsána v dokumentaci. Tento nástroj dokáže pracovat s různými druhy panoramatických fotografií. Zajímavou funkcí jsou fotografie s proměnným rozlišením, které se skládají z více menších fotografií a které jsou následně načteny podle směru kamery.

Téměř vše je přizpůsobitelné a lze modifikovat i chování aplikace pomocí skriptů. Nevýhodou je, že tento nástroj nabízí pouze placenou licenci. Nástroj je možné dále rozšířit pomocí pluginů, které vytvořili další vývojáři.

Obrázek 2.12: Ukázková prohlídka rakouského Kuchlerhaus z Krpano Panorama Viewer [5]. Uprostřed scény se nachází šipka pro přechod na další snímek. Po levé straně se nachází lišta se seznamem snímků v prohlídce.

Kapitola 3

Grafický engine Unity

K vytvoření virtuální prohlídky bude využít grafický engine *Unity*. Především pro přímou podporu vývoje ve virtuální realitě bez nutnosti instalace dodatečných SDK[1] a pro obrovskou aktivní vývojářskou komunitu. V této kapitole jsou popsány důležité části enginu *Unity*, které jsou důležité pro vytvoření virtuální prohlídky.

3.1 Koncept vývoje v Unity

Projekt v *Unity* je vždy tvořen jednou či více scénami. Při přechodu mezi scénami je původní scéna zrušena a nahrazena novou scénou. Každá scéna se skládá z objektů typu `gameObject`. Objekty je možné libovolně hierarchicky uspořádat a každý objekt je tvořen komponentami. Existuje mnoho druhů komponent, ale všechny objekty musí povinně obsahovat komponentu `Transform`. Tato komponenta obsahuje informace o pozici, velikosti a rotaci daného objektu ve scéně.

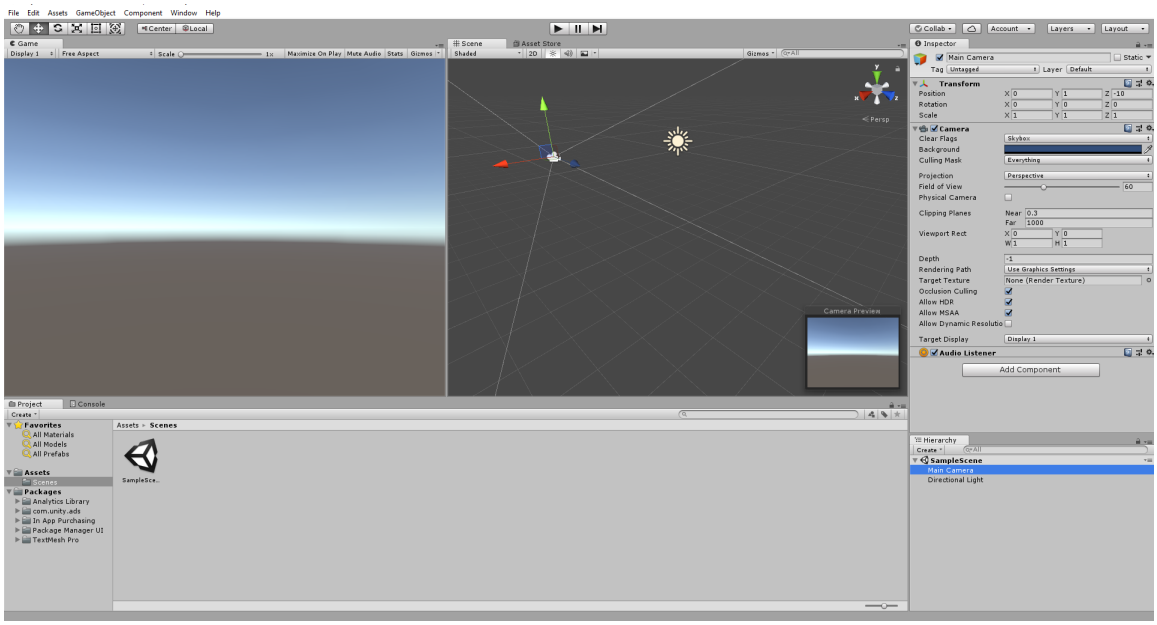
Z objektů je možné vytvořit takzvané prefabrikované objekty, které je možné dále využívat. Prefabrikované objekty lze chápat jako šablonu, ze které jsou vytvářeny nové objekty. Tyto objekty si zachovávají svoji vazbu na původní prefabrikovaný objekt a změnou prefabrikovaného objektu lze ovlivnit i všechny další objekty, které byly z prefabrikovaného objektu vytvořeny.

3.2 Unity editor

Unity editor nabízí jednoduché a uživatelsky přívětivé prostředí, ve kterém se dokáže poměrně rychle zorientovat i začínající vývojář. Vše je přehledně organizováno do několika částí a vývojář má tak pohodlný přístup ke všem vlastnostem vyvíjené aplikace. Díky otevřenému API[2] je možné editor rozšířit a tím zefektivnit práci v něm.

Rozhraní editoru je rozděleno do následujících hlavních částí: okno projektu, okno scény, okno hierarchie, okno inspektoru, panel nástrojů a náhled aplikace. Každé části lze libovolně měnit velikost a pozici v rámci editoru. Rozhraní editoru je zobrazeno na obrázku 3.1.

[1] Software Development Kit
[2] Application Programming Interface



Obrázek 3.1: Rozhraní Unity editoru. Úplně vlevo, kde je vidět horizont, se nachází náhled aplikace. Vedle náhledu aplikace po pravé straně se nachází okno scény. Nad náhledem aplikace a oknem scény je panel nástrojů s ovládacími prvky. Naopak pod náhledem aplikace a oknem scény se nachází okno projektu. V postranním panelu na pravé straně se nachází okno inspektoru a pod ním hierarchie projektu.

3.2.1 Náhled aplikace

Náhled aplikace je renderován z kamer ve scéně a je to finální podoba aplikace, jak bude vypadat po sestavení na cílovém zařízení. Je zapotřebí použit alespoň jednu kameru, aby uživatel viděl vytvořenou scénu. Náhled aplikace lze aktivovat zapnutím herního režimu z panelu nástrojů. Jakákoliv změna provedená v herním režimu bude po ukončení herního režimu vrácena zpět.

3.2.2 Okno scény

Okno scény lze chápat jako interaktivní pohled do světa, který je vytvářen. V tomto okně lze měnit pozici, rotovat či měnit velikost všech objektů ve scéně. Naučit se pracovat s oknem scény je jedna z nezbytných věcí pro práci v *Unity*.

3.2.3 Okno hierarchie

Okno hierarchie je složeno ze seznamu všech objektů typu `GameObject`, které se nachází ve scéně. Současně s tím, jak jsou objekty přidávány nebo odebírány ze scény, tak stejně tak se objevují i v oknu hierarchie. Výchozí řazení objektů je dle pořadí, v jakém byly do scény vloženy. Objekty lze tažením přeuspořádat nebo seskupit.

Unity používá koncept rodičovství. To znamená, že pokud je vytvořena skupina objektů, tak nejvýše postavený objekt je rodičovský a všechny další objekty, které jsou seskupené pod tímto objektem, jsou potomky daného objektu. Okno hierarchie a rodičovství jsou na obrázku 3.2.

Obrázek 3.2: Okno hierarchie a ukázka rodičovství. Objekty `Child` a `Child 2` jsou potomky objektu `Parent`. Objekt `Child 3` je potomkem objektů `Child 2` a `Parent` [13].

3.2.4 Okno projektu

V tomto okně se nachází všechny soubory, ze kterých se projekt skládá, jako například textury, materiály, skripty nebo scény. Na levé straně je umístěn panel, ve kterém se nachází složky, které tvoří strukturu projektu. Pokud je nějaká složka vybrána, její obsah se zobrazí v panelu po pravé straně.

Nad složkami tvořící strukturu projektu v levném panelu se nachází sekce s oblíbenými adresáři. Do této sekce lze vložit často používané adresáře pro rychlejší přístup. Tažením adresáře jej lze zařadit mezi oblíbené. Nad panelem zobrazující obsah adresáře po pravé straně se nachází vyhledávací pole pro rychlejší vyhledávání složek a souborů.

3.2.5 Okno inspektoru

Projekt se skládá z vícero objektů typu `GameObject`, které mají různé vlastnosti. Okno inspektoru poskytuje detailní informace o vybraném objektu a jeho vlastnostech, včetně všech komponent a jejich vlastností, které byly k objektu přiřazeny. V okně inspektoru lze modifikovat funkcionalitu objektů ve scéně.

3.2.6 Panel nástrojů

Panel nástrojů se skládá ze sedmi ovládacích prvků a každý prvek je spojen s odlišnou částí editoru. Transformační nástroje pro manipulaci s objekty v okně scény, transformace okna scény, mediální tlačítka pro náhled aplikace, tlačítko Cloud pro otevření služeb *Unity*, tlačítko uživatelského účtu, tlačítko vrstev ovládající zobrazení objektů v okně scény a tlačítko rozložení hlavních částí editoru.

3.3 Použití textur

Typicky *mesh* geometrie, což je základní grafická primitiva v *Unity*, dává objektu pouze hrubou aproximaci jeho tvaru. Většina jemných detailů je zajištěna pomocí textur. Textura je standardní bitmapový obrázek, který je aplikován na povrch *mesh* geometrie. Texturu si lze představit jako gumovou fólii, na kterou byl vytištěn obrázek a následně napnutím na vhodných místech nanesena na *mesh* geometrii.

3.4 Použití materiálů

Materiály se používají ve spojitosti s renderovanými komponentami a hrají primární roli v tom, jak objekt bude zobrazen. Obecně lze říci, že materiál je kombinací textury a shaderu. Shader je speciální druh grafického programu, který určuje, jak bude kombinace textury a světelné informace použita pro generování pixelů renderovaného objektu na obrazovce.

3.5 Tvorba vlastních skriptů

Na skripty je v *Unity* nahlíženo jako na komponenty a ovlivňují vlastnosti či chování objektů. Vytváření skriptů je nedílnou součástí vývoje v *Unity* a neobejde se bez nich ani ta nejjednodušší aplikace. Pomocí skriptů lze například vytvářet grafické efekty nebo ovládat fyziku objektů.

Primárním jazykem pro tvorbu skriptů v *Unity* je C# a výchozím editorem je *Microsoft Visual Studio*, který je silně doporučován, protože existuje volně dostupný plugin pro integraci s *Uninty*. V případě použití jazyka C# musí každý skript explicitně dědit třídu `MonoBehaviour`, což je základní třída, ze které dědí každý skript v *Unity* a musí povinně implementovat metody `Start` a `Update`.

Skript je po spuštění zahájen metodou `Start`, která je provedena pouze jednou. V této metodě by měla proběhnout inicializace všech proměnných nebo přiřazení komponent objektům. Následně je vyvolána metoda `Update`, která se provádí pravidelně každý snímek. V této části by měla být veškerá logika skriptu.

Třída `MonoBehaviour` zahrnuje velké množství dalších užitečných metod, které lze využít při psaní skriptů. Tyto metody lze najít v oficiální dokumentaci *Unity*.

3.6 Asset Store

Asset Store[3] je obchod s doplňky do *Unity*. Vzhledem k tomu, že *Unity* pro práci s virtuální realitou nabízí minimum předchystaných prefabrikovaných objektů a komponent, tak je toto místo velice vhodným zdrojem. Lze zde najít doplňky, které jsou zdarma, ale i placené.

[3] Asset Store - https://assetstore.unity.com/

Kapitola 4

Webové technologie

Tato kapitola popisuje vybrané webové technologie a nástroje, které byly použity k vývoji aplikace pro vytváření virtuálních prohlídek. Zejména jejich části, které byly potřebné k implementaci.

4.1 Laravel

Laravel [6] je framework pro tvorbu webových aplikací ve skriptovacím jazyce PHP. První verze byla vydána v roce 2011 a neustále se vyvíjí. Tento framework se těší veliké oblibě ve světě a je populární mezi mnoha vývojáři. Je postaven na základech frameworku *Symfony*, avšak jeho znalost není nutná, protože *Laravel* nabízí vlastní rozhraní.

Framework si zakládá na přehledném a jednoduchém kódu. Jeho předností je optimalizace pro reálné webové aplikace. Ta spočívá v již předpřipravených komponentách, například pro registraci a autentizaci uživatelů. Velkou výhodou je velmi kvalitně zpracovaná dokumentace a obrovská vývojářská komunita. Současně existuje webová stránka *Laracasts*[1], na které je umístěno mnoho videotutoriálů, a tak začínající vývojář se dokáže s tímto frameworkem seznámit velmi rychle.

4.1.1 Základní vlastnosti

Laravel využívá architekturu MVC (Model–View–Controller) [23], která rozděluje aplikaci do tří na sobě nezávislých vrstev tak, že změna některé z nich má minimální vliv na ostatní. Výhodou tohoto přístupu je přehlednější a škálovatelnější aplikace.

- **Model** – vrstva obsahující logiku aplikace a data
- **View (pohled)** – vrstva zobrazující výstup, obvykle využívá šablonovací systém
- **Controller (řadič)** – řídící vrstva, která se stará o zpracovávání požadavků, na základě kterých mění stav modelu

Dále využívá jednoduchý, ale účinný šablonovací systém *Blade*, který na rozdíl od ostatních PHP šablonovacích systémů, neomezuje uživatele v používání PHP kódu v pohledech.

[1] Laracasts - https://laracasts.com/

4.1.2 Adresářová struktura

Adresářová struktura frameworku je dobrým výchozím bodem pro velké i malé aplikace, avšak uživatel si může svojí aplikaci libovolně zorganizovat, protože *Laravel* neklade téměř žádná omezení na to, kde má jaká třída být umístěna.

Většina uživatelem definovaného kódu se nachází v adresáři `app`, kde je umístěna logika celé aplikace. Pohledy, kaskádové styly a skripty se nachází v adresáři `resources`.

```
/
├── app ... hlavní adresář s aplikací
│   ├── Console
│   ├── Events
│   ├── Exceptions
│   ├── Handlers
│   ├── Http
│   ├── Listeners
│   └── Providers
├── artisan
├── bootstrap ... spouštěcí soubory jádra aplikace
│   ├── app.php
│   ├── autoload.php
│   └── cache
├── composer.json ... závislostí aplikace
├── config ... konfigurační soubory
├── database ... soubory pro databázi
│   ├── migrations
│   └── seeds
├── package.json
├── public
│   └── index.php ... vstupní bod aplikace
├── resources ... pohledy a nezkompilované soubory
│   ├── assets
│   ├── lang
│   └── views
├── routes ... definice cest v aplikaci
├── server.php
├── storage ... úložiště aplikace
│   ├── app
│   ├── framework
│   └── logs
├── tests ... integrační testy
└── vendor ... závislosti nainstalované pomocí Composeru
```

Obrázek 4.1: Struktura aplikace v Laravelu včetně popisku některých důležitých částí frameworku.

4.1.3 Composer

Composer[2] je nástroj pro správu knihoven a závislostí v PHP. Umoňuje uživateli přidávat funkčnost do aplikace pomocí balíčků a zároveň provádí kontrolu kompatibility v prostředí, ve kterém pracuje. Konfigurace se nachází v souboru **composer.json**, který je umístěn v kořenovém adresáři aplikace a je ve formátu JSON[3]. Dále se zde nachází informace o autorovi aplikace a informace o aplikaci samotné.

Nástroj se ovládá pomocí příkazové řádky a je multiplatformní. Dále používá soubor **composer.lock**, který je rovněž ve formátu JSON a nachází se v něm informace o použitých knihovnách a jejich verzích, které mají být použity.

4.1.4 Šablonovací systém Blade

Výhodou šablonovacího systému *Blade* je používání PHP kódu v pohledech bez žádného omezení. Ve skutečnosti všechny pohledy, které jsou napsány v systému *Blade*, jsou kompilovány do PHP kódu a uloženy v *cache*, dokud nedojde k jejich modifikaci. Soubory *Blade* mají příponu **.blade.php** a jsou uloženy v adresáři `resources/views`.

Hlavní výhody tohoto šablonovacího systému jsou dědičnost šablon a vytváření sekcí. Díky tomu lze vytvářet a spravovat hierarchicky uspořádané pohledy.

4.1.5 Databáze

Laravel používá pro práci s databází objektově relační mapování, které se nazývá *Eloquent ORM*. Díky tomu lze s databází pracovat na úrovni aplikace a není nutné psát žádné SQL dotazy. Zároveň je díky tomu aplikace nezávislá na konkrétním databázovém systému. V současné chvíli podporuje *Laravel* následující databázové systémy:

- MySQL
- PostgreSQL
- SQLite
- SQL Server

Pro vytvoření databázových tabulek včetně jednotlivých sloupců a definici všech cizích klíčů a dalších atributů se používají takzvané migrace. Tyto migrační soubory jsou uloženy v adresáři `database/migrations`. Pro vložení počátečních dat do tabulek se používají takzvané seedy, které se nachází v adresáři `database/seeds`.

4.1.6 Vytváření API

Při tvorbě API je vhodné použít vrstvu mezi modelem a odpovědí ve formátu JSON, která zajistí transformaci dat. Laravel nabízí třídy, které jednoduše umožňují provést transformaci modelů a kolekcí do formátu JSON. Tyto třídy bývají obvykle uloženy v adresáři `app/Http/Resources`.

[2] Composer - https://getcomposer.org/
[3] JavaScript Object Notation

4.2 Vue.js

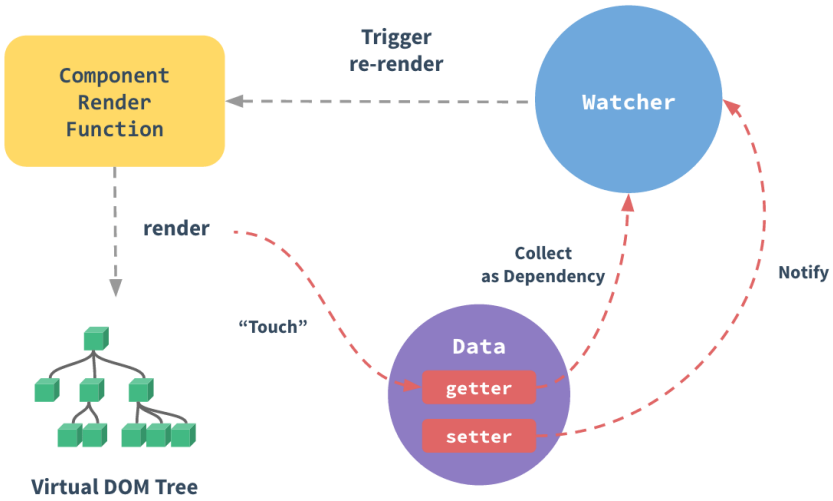
Vue.js [17], často označováno pouze jako *Vue*, je progresivní javascriptový framework, který slouží pro vytváření uživatelských rozhraní a *single-page* aplikací. Vznikl v roce 2014 a patří k rychle vyvíjejícím se frameworkům. Kombinuje technologie, které obsahují frameworky *AngularJS* a *React*. Zaměřuje se na jednoduchost a intuitivnost a lze jej snadno integrovat do již existujících projektů.

4.2.1 Základní vlastnosti

Vue pracuje s konceptem virtuálního DOM[4] a je založeno na systému komponent, které umožňují rozdělit aplikaci do menších částí. Každá komponenta obsahuje vlastní šablonu, která definuje vzhled komponenty a vlastní logiku, která implementuje chováni komponenty. Do komponent lze vkládat i CSS pravidla, která se vztahují lokálně pouze na danou komponentu.

Důležitou vlastností frameworku je jeho reaktivita. Každá komponenta obsahuje vlastní „watcher“, který sleduje atributy dané komponenty. Jakmile dojde ke změně hodnoty některého atributu, dojde k aktualizaci renderované komponenty. Reaktivita frameworku je zachycena na obrázku 4.2.

Vue využívá direktivy, jejich syntaxe začíná prefixem „v-“ a slouží k manipulaci s DOM. Podobně jako *AngularJS*, dokáže i *Vue* používat obousměrnou vazbu dat pomocí direktivy `v-model` a tím zajistit automatickou aktualizaci obsahu.



Obrázek 4.2: Reaktivita frameworku Vue. Při vytvoření komponenty se zahájí sledování dat komponenty. Jakmile je detekována změna na datech, dojde k novému vykreslení komponenty [17].

[4]Document Object Model

4.2.2 Vue Router

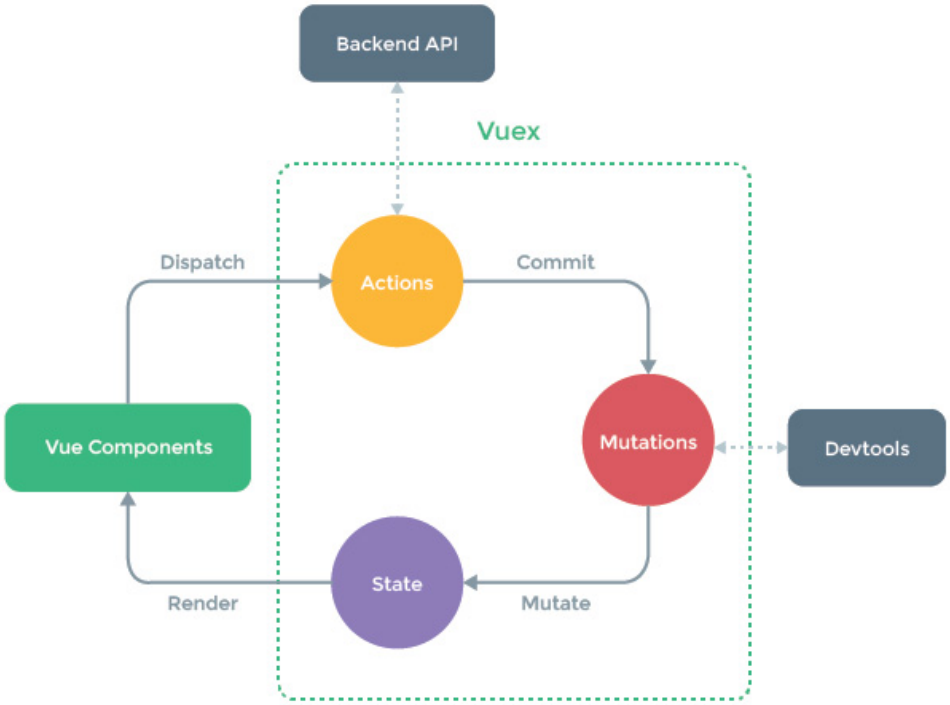
Vue Router[5] je modul pro framework *Vue*, který se stará o navigaci v *single-page* aplikaci. Tento modul používá konfigurační soubor, ve kterém jsou definovány jednotlivé cesty v aplikaci. Ke každé cestě je přiřazena komponenta, která definuje obsah zobrazené stránky. Tento modul rovněž vkládá do aplikace globální objekt `router`, díky kterému lze navigaci ovládat pomocí javascriptového kódu.

4.2.3 Vuex

Vuex [18] je knihovna pro framework *Vue*, která slouží k řízení stavů aplikace. Její úlohou je poskytnout centralizovaný zdroj dat pro všechny komponenty, které se v aplikaci vyskytují. Vzhledem k tomu, že knihovna *Vuex* je napsána pomocí frameworku *Vue*, tak všechna data, která obsahuje, jsou reaktivní. To znamená, že pokud některá komponenta změní stav aplikace, tak ostatní komponenty mohou na tuto změnu reagovat. Model zobrazující aplikaci, která využívá knihovnu *Vuex* je znázorněn na obrázku 4.3.

Všechny sdílené stavy aplikace jsou extrahovány mimo komponenty, kde jsou spravovány pomocí *Vuex*. Tyto stavy lze modifikovat pouze pomocí takzvaných mutací. Mutace jsou podobné událostem ve smyslu, že obsahují vlastní identifikátor a funkci, která se po aktivaci vykoná. Mutace mohou obsahovat pouze synchronní operace.

Obdobou mutací jsou takzvané akce. Rozdílem mezi mutací a akcí je, že akce mohou obsahovat asynchronní operace, například dotaz na API serveru pro získání dat.

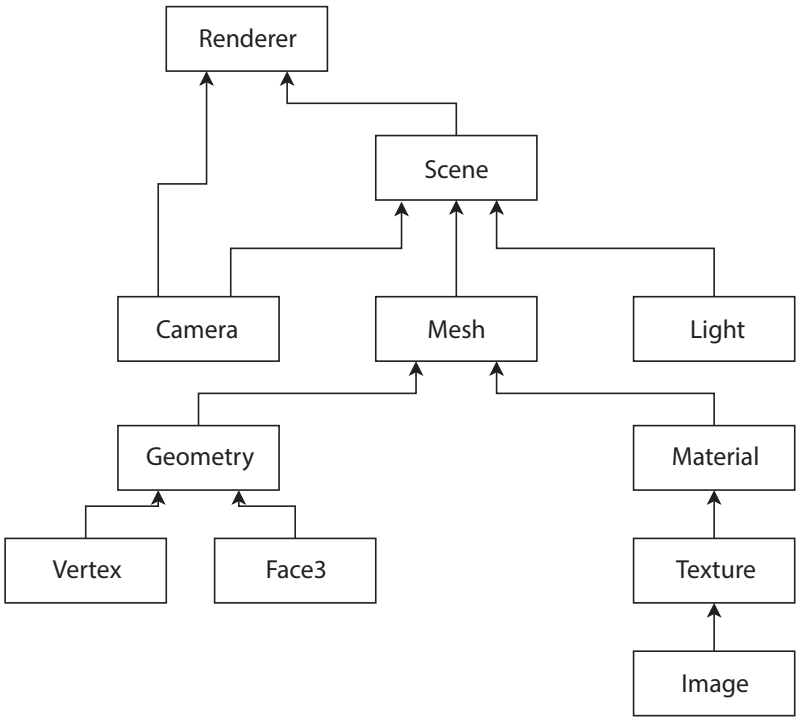


Obrázek 4.3: Model aplikace využívající knihovnu Vuex. Na obrázku je znázorněna modifikace stavu aplikace pomocí mutací a akcí [18].

[5] Vue Router - https://router.vuejs.org/

4.3 Three.js

Three.js [10] [21] je javascriptová knihovna, která byla poprvé uveřejněna roku 2010. Slouží k vytváření a zobrazování 3D animací akcelerovaných přes GPU[6] ve webovém prohlížeči a využívá podporu rozhraní *WebGL*. Aplikace vytvořená pomocí *Three.js* se skládá z objektů, které tvoří stromovou strukturu. Tato struktura je zobrazena na obrázku 4.4. Některé objekty budou popsány v následujících kapitolách.



Obrázek 4.4: Stromová struktura aplikace v Three.js [21].

4.3.1 Renderer

Renderer slouží k zobrazení scény na obrazovce pomocí *WebGL*. Renderer je vytvořen pomocí třídy `WebGLRenderer`.

4.3.2 Scene

Scene představuje samotnou scénu, která je renderována. Ve scéně se nachází všechny objekty, které budou na obrazovce zobrazeny. Scéna je reprezentována třídou `Scene`.

4.3.3 Camera

Tento objekt představuje kameru, pomocí které je na scénu nahlíženo. Kamera může být různého typu. Perspektivní kamera `PerspectiveCamera` se chová jako kamera ve skutečném světě a zachycuje scénu tak, jak by ji člověk vnímal očima. Naopak ortografická kamera `OrthographicCamera` promítá obraz tak, že objekty nemění svoji velikost na základě vzdáleností od kamery.

[6] Graphics Processing Unit

4.3.4 Mesh

Mesh představuje 3D model a většinou se skládá z geometrie a materiálu. Mesh je reprezentován třídou `Mesh`.

4.3.5 Material

Materiál definuje, jak bude konkrétní objekt vypadat. V *Three.js* se nachází více druhů materiálů, jedním z nich je například `MeshBasicMaterial`.

4.3.6 Geometry

Geometrie definuje tvar objektu. Existuje více druhů geometrie a jednou z nich je například `SphereBufferGeometry`.

Kapitola 5

Experimenty předcházející návrhu aplikace

Před samotným návrhem aplikace bylo zapotřebí provést několik experimentů, aby byly ověřeny možnosti, které výsledná aplikace může zahrnovat. Experimenty byly prováděny na zařízení *Samsung Galaxy S6* v kombinaci s headsetem *Samsung Gear VR*. Pro pořízení 360° panoramatických fotografií byla použita digitální kamera *Samsung Gear 360* (2016). Verze *Unity*, ve které byly experimenty provedeny byla 2018.2.16f1.

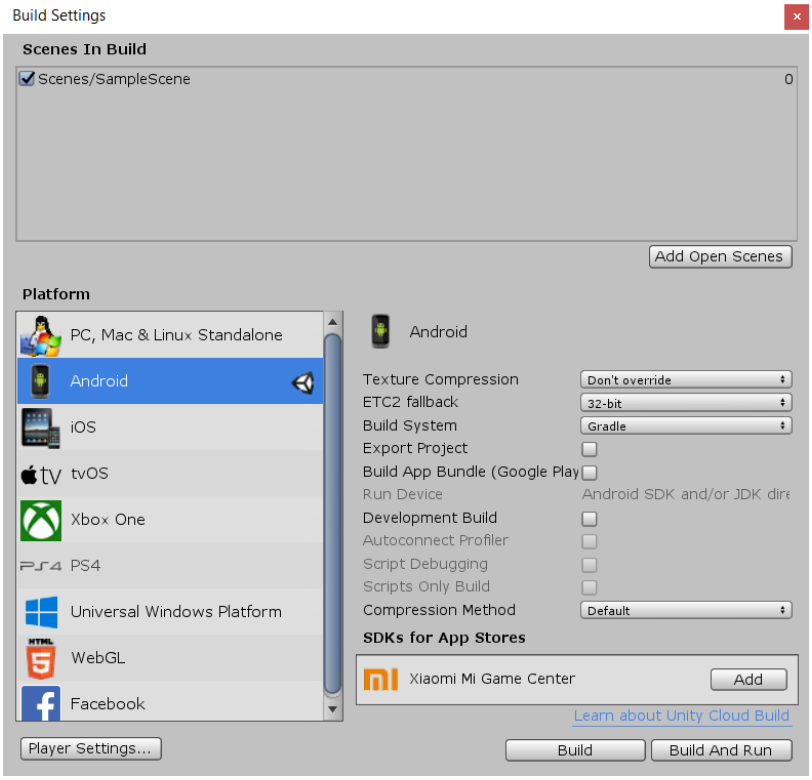
5.1 Konfigurace projektu

Ze všeho nejdříve bylo nutné nainstalovat a správně nastavit *Android* SDK. Toho lze docílit dvěma způsoby, a to buď pomocí příkazové řádky, nebo nainstalovat celé *Android Studio*. Kompletní návod pro instalaci *Android* SDK lze najít v dokumentaci[1] *Unity*. Velmi důležité je mít na paměti, že je vyžadován aktivovaný USB ladící režim na mobilním zařízení.

Dalším krokem bylo vytvořit a nakonfigurovat projekt v *Unity*. Nejprve bylo potřeba zvolit cílovou platformu. Vzhledem k tomu, že testované zařízení běží na systému *Android*, tak bylo zapotřebí jej zvolit jako cílovou platformu. Po vytvoření projektu bude *Unity* vyžadovat cestu k nainstalovanému *Android* SDK, který byl nainstalován v předchozím kroku. Cílovou platformu lze změnit v okně **Build Settings**, které je přístupné z horní nástrojové lišty. Z tohoto okna po kliknutí na tlačítko **Player Settings** se otevře okno inspektoru pro nastavení hráče. Pod záložkou **XR Settings** je nutné aktivovat podporu virtuální reality a SDK virtuální reality nastavit na **Oculus**.

Dalším nezbytným krokem je získání Oculus Signature (OSIG) souboru, který je nutný pro sestavení aplikace na cílovém zařízení. Ačkoliv bez tohoto souboru lze aplikaci do zařízení úspěšně dostat, tak při spuštění aplikace skončí chybovou hláškou. Soubor musí být umístěn v adresáři `<project>/Assets/Plugins/Android/assets/`, kde `<project>` je cesta k projektu na disku. Kompletní návod jak OSIG soubor vygenerovat lze najít na stránkách Oculusu[2]. Po splnění všech těchto kroků je možné aplikaci sestavit na cílovém zařízení a spustit.

[1] Instalace Android SDK - https://docs.unity3d.com/Manual/android-sdksetup.html
[2] Generování OSIG souboru - https://dashboard.oculus.com/tools/osig-generator/



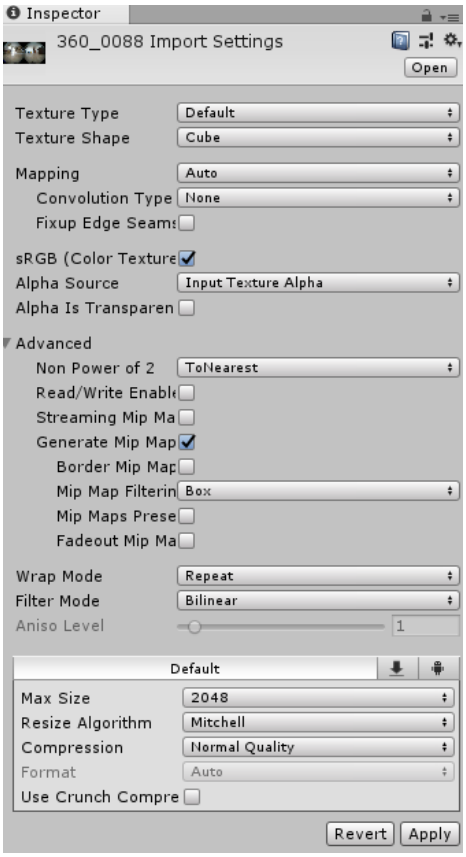
Obrázek 5.1: Okno s konfigurací sestavení aplikace. V horní části je výběr scén, které budou zahrnuty do výsledné aplikace. V levém panelu je zvolena cílová platforma a v panelu po pravé straně se nachází detailní konfigurace sestavení pro cílovou platformu. Výchozí hodnoty jsou dostačující a není třeba je měnit.

5.2 První experiment – pořízení fotografie a zobrazení v headsetu

První experiment se skládal z prostého pořízení statického obrázku a jeho zobrazení. Použitý obrázek je zobrazen na obrázku 5.2.

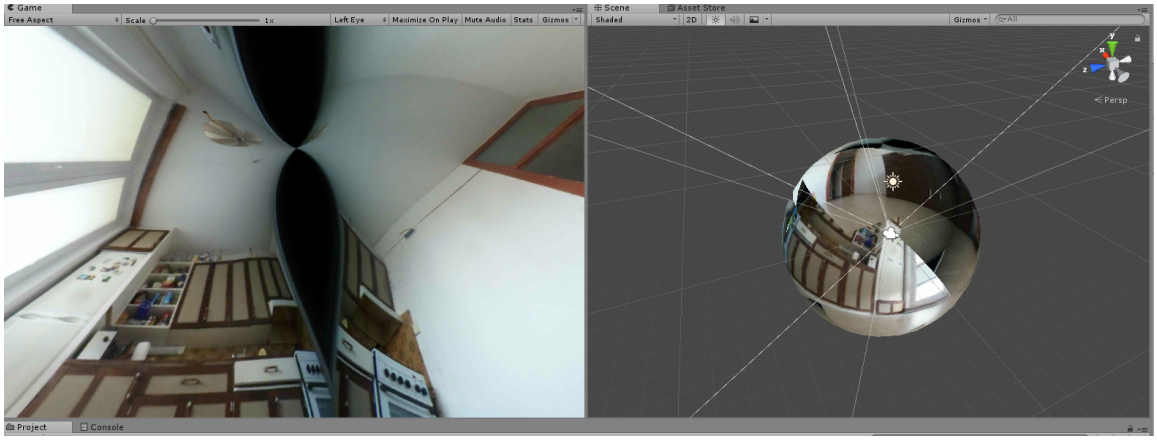
Obrázek 5.2: Snímek pořízený kamerou v kuchyni na zemi. Snímek je rozdělen na dvě poloviny, kde každá polovina zachycuje obraz ve 180° ve vertikální i horizontální rovině.

Snímek byl naimportován do *Unity* jako textura. V inspektoru textury je důležité nastavit její tvar na hodnotu **Cube**. Nastavení textury při importu je na obrázku 5.3.



Obrázek 5.3: Import textury. Důležité bylo nastavení parametru **Texture Shape** na hodnotu **Cube**. Dále ponechat parametr **Mapping** na hodnotě **Auto**. Díky tomu Unity samo detekuje formát fotografie. Dále lze upravovat parametry jako například maximální velikost textury nebo kompresní algoritmus.

Scéna byla tvořena kamerou a koulí ve 3D prostoru. Kamera byla umístěna do středu koule tak, aby byla od všech bodů koule stejně vzdálena. Textury se v *Unity* zpravidla samostatně nepoužívají, a tak bylo nutné vytvořit nový materiál. Při vytvoření materiálu bylo nutné nastavit jeho shader na **Skybox/Cubemap**, protože byl aplikován na *mesh* geometrii koule. Posledním krokem bylo v inspektoru koule u komponenty **Mesh Renderer** nastavit vytvořený materiál s vloženou texturou. Výsledek je zobrazen na obrázku 5.4.



Obrázek 5.4: Aplikace textury na kouli. V levé části obrázku se nachází finální náhled, kde jsou viditelné nedokonalosti. V pravé části obrázku se nachází okno scény, kde lze vidět kameru umístěnou do středu koule, na kterou byla aplikována textura.

Experiment odhalil, že snímek pořízený kamerou nelze přímo aplikovat jako texturu, protože při pohledu do vrchní části scény zde vznikaly mezery a bude vyžadovat další zpracování před vložením do projektu.

5.3 Zpracování fotografie pořízené kamerou

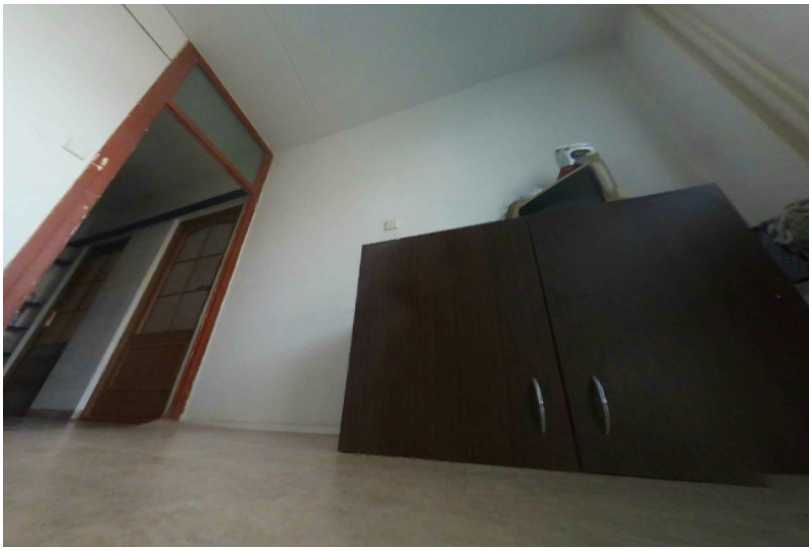
První experiment ukázal, že *Unity* si s pořízenou fotografií nedokáže poradit a bylo nutné fotografii dále zpracovat. Fotografii bylo nutné složit do sférického panoramatického formátu pomocí speciálního nástroje. Některé nástroje, které lze ke složení použít, jsou uvedeny v kapitole 2.2.

Ke složení fotografie byl využit online nástroj *NadirPatch*[3]. Tento nástroj nabízí funkci složení fotografie přímo pro kameru *Samsung Gear 360*. Nevýhodou je, že lze zpracovat pouze 1 fotografii v jednom požadavku. Dávkové zpracování je zpoplatněno, ale pro experimentální účely je to dostačující řešení. Výsledná fotografie je na obrázku 5.5.

Obrázek 5.5: Výsledná fotografie ve sférickém panoramatickém formátu.

[3]NadirPatch - https://nadirpatch.com/

Takto zpracovaná fotografie byla následně vložena jako textura do projektu a nahrazena s ní texturou z předchozího experimentu. Náhled se zpracovanou fotografií je na obrázku 5.6.



Obrázek 5.6: Finální náhled se zpracovanou fotografií. V pravém dolním rohu lze pozorovat drobnou chybu.

Ukázalo se, že nástroj *NadirPatch* dokáže rychle a poměrně dobře zpracovat fotografie z kamery. Z finálního náhledu zmizely prázdná místa, avšak v jednom místě ve finálním náhledu dochází k drobné chybě, která vznikla při skládání fotografií. Protože chyba není na první pohled patrná, tak ve všech následujících experimentech bude vycházeno z tohoto bodu.

5.4 Druhý experiment – uživatelská interakce a změna snímků

Druhý experiment byl zaměřen na způsob jak zajistit interakci mezi uživatelem a aplikací, aby uživatel mohl libovolně přecházet mezi jednotlivými snímky. Pro tento experiment byly zachyceny 3 snímky pokoje z různých bodů a následně byly zpracovány způsobem popsaném v předcházející kapitole.

Scéna se skládala ze 3 koulí, kde každá koule obsahovala jeden z pořízených snímků. Následně do každé koule byly umístěny 2 body, které měly sloužit pro přechod na odlišný snímek. Bohužel prostředky dostupné v základu *Unity* nebyly dostačující, a tak musel být stažen a naimportován balíček z *Asset Store*. Konkrétně balíček *Oculus Integration*, který je zdarma.

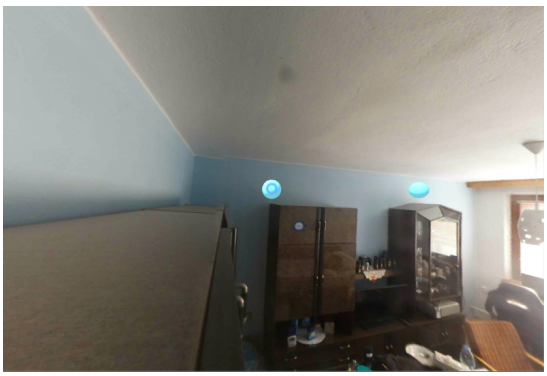
Obyčejná kamera byla nahrazena prefabrikovanou kamerou `OVRCameraRig`. Tato kamera se skládá ze tří tří kamer. `LeftEyeAnchor` kamery pro levé oko, `RightEyeAnchor` kamery pro pravé oko a `CenterEyeAnchor` kamery, která průměruje pozici levého a pravého oka. Dále bylo nutné na prefabrikovanou kameru aplikovat skript `OVRPhysicsRaycaster`. Tento skript zajišťuje, že na místo, kam se uživatel dívá, je promítán pomyslný paprsek, díky kterému lze provádět interakci s objekty ve scéně.

Dále bylo nutné přidat do snímku objekt se skriptem `EventSystem`. Ten zajišťuje, že objektům jsou zasílány zprávy na základě proběhlých událostí, jako například stisk klávesy,

stisk myši a jiné. Primární role event systému v *Unity* jsou: detekce objektu, se kterým je prováděna interakce, detekce vstupního modulu, který je používán, správa pomyslného paprsku, který je vysílán z kamery a aktualizace všech vstupních modulů. Event systém v základu obsahuje standardní input modul, ale ten v kombinaci s virtuální realitou nelze použít, a tak jej bylo nutné nahradit skriptem `OVRInputModule`. Dále bylo nutné tomuto skriptu nastavit parametr **Ray Transform** na jednu z kamer, která je ve scéně a tím určit, jaká kamera bude sloužit jako vstup event systému. Byla použita kamera `CenterEyeAnchor`. Do snímku byl poté vložen prefabrikovaný objekt `GazePointerRing`, který slouží jako ukazatel při interakci s objektem. Bylo nutné mu mu opět nastavit parametr **Ray Transform** na kameru `CenterEyeAnchor`.

V dalším kroku byl vytvořen prázdný objekt `Manager` a do něj vložen vlastní vytvořený skript. Skript obsahoval pouze veřejnou statickou metodu, pomocí které se měnila pozice kamery na základě 3D vektoru, který byl předán jako parametr.

Posledním krokem bylo vytvořit skript pro objekty uvnitř koulí, které sloužily k přechodu mezi jednotlivými snímky. Každému objektu byl tento skript přiřazen samostatně jako komponenta a jako parametr vyžadoval referenci na kouli, která představovala snímek, na který bude uživatel přesunut. Skript naslouchal eventu `OnPointerClick` a měnil pozici kamery pomocí objektu `Manager`, který obsahoval statickou metodu pro změnu pozice kamery. To znamená, že v momentě, kdy uživatel hleděl na daný objekt na snímku a zmáčkl postranní tlačítko na headsetu, byl přesunut do příslušného snímku.



(a) Výchozí snímek.

(b) Snímek po přechodu.

Obrázek 5.7: Zobrazení výchozího snímku (a) a následný přechod po interakci s objektem pro přechod mezi snímky (b). Modré body na snímcích představují objekty, které slouží k přechodu mezi snímky.

Tento experiment ověřil způsob, kterým lze uživateli umožnit přechod mezi snímky.

5.5 Třetí experiment – průlet prostorem pomocí transformace kamery

Tento experiment byl zaměřen na pořízení sekvence snímků a vytvoření plynulého přechodu mezi nimi tak, aby uživatel měl dojem, že se pohybuje prostorem. Původní myšlenka byla taková, že transformací kamery po ose `z`, respektive směrem vpřed z pohledu uživatele, bude možné se dostat do bodu, ve kterém bude možné změnit promítaný snímek a vrátit kameru do počáteční pozice tak, aby se přechod zdál plynulý.

Scéna byla vytvořena stejně jako v prvním experimentu. Byl vytvořen skript, který jako vstupní parametry vyžadoval rychlost transformace po ose **z** a vzdálenost, po které dojde ke změně snímku. Tento skript byl následně aplikován na objekt koule ve scéně. Experiment se skládal ze tří sad testovacích dat:

- Sada 30-ti snímků zachycených po 10-ti centimetrech
- Sada 25-ti snímků zachycených po 30-ti centimetrech
- Sada 15-ti snímků zachycených po 50-ti centimetrech

Experimentováním s parametrem, který určoval vzdálenost, po které dojde ke změně snímku, byla pro každou sadu nalezena optimální hodnota, při které se přechod zdál nejplynulejší. Následně byly prováděny experimenty s rychlostí transformace kamery. Výpočet nové **z** souřadnice je znázorněn rovnicí na obrázku 5.8.

$$z_{n+1} = z_n + \lambda * t$$

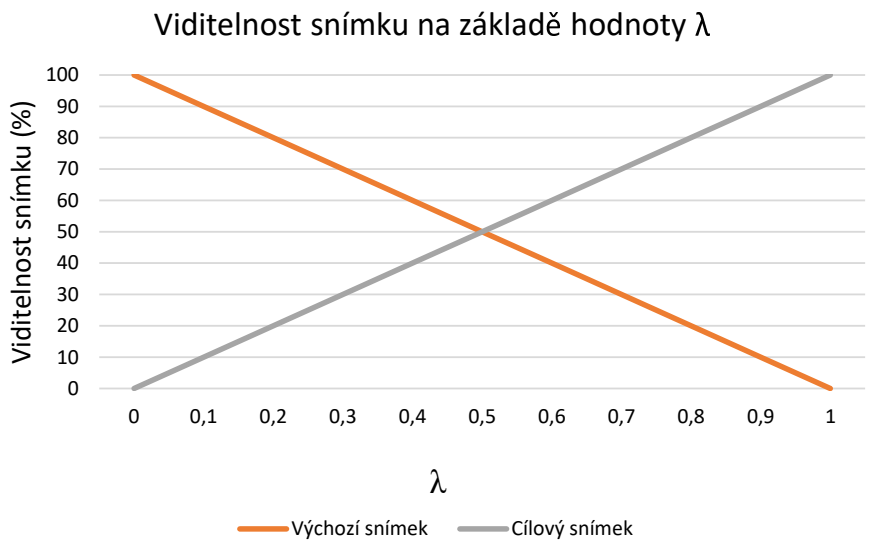
Obrázek 5.8: Rovnice pro výpočet nové **z** souřadnice. Proměnná $\boldsymbol{\lambda}$ je parametr rychlosti transformace kamery a proměnná **t** je čas ve vteřinách, který uběhl od předchozího zpracovaného snímku.

Experimentováním bylo zjištěno, že tento způsob přechodu není ideální, protože přechod je ovlivněn několika faktory. Především chybou nesprávného pořízení snímků. Sebemenší odchylka ve vzdálenosti vedla k viditelnému skoku v přechodu. Dále experiment odhalil, že čím rychlejší přechod byl, tím menší se chyba zdála. Nejplynulejší přechod byl u snímků pořízených po 10-ti centimetrech, nicméně v praxi by toto řešení při prohlídce velkých prostorů bylo velmi neefektivní.

5.6 Čtvrtý experiment – průlet prostorem pomocí lineární interpolace

Vzhledem k tomu, že způsob přechodu mezi snímky z předchozího experimentu nebyl dostačující, bylo potřeba vymyslet nový způsob. Tento experiment byl zaměřen na lineární interpolaci dvou po sobě jdoucích snímků a vytvoření plynulého přechodu.

Bylo nutné vytvořit vlastní shader, kterému byly předány textury dvou po sobě jdoucích snímků a váha, která určovala poměr viditelnosti jednotlivých snímků ve scéně. Tento poměr je vyjádřen grafem na obrázku 5.9. Na základě váhy byla následně provedena lineární interpolace snímků.



Obrázek 5.9: Graf znázorňující viditelnost výchozího a cílového snímku na základě hodnoty $\boldsymbol{\lambda}$, která je definována v intervalu $\boldsymbol{<0,1>}$.

Obrázek 5.10: Lineární interpolace dvou snímků, $\boldsymbol{\lambda = 0,5}$.

Byl vytvořen skript, který jako vstupní parametr vyžadoval rychlost, kterou byla interpolace mezi po sobě jdoucími snímky provedena. Výpočet hodnoty $\boldsymbol{\lambda}$ je znázorněn rovnicí na obrázku 5.11, kde počáteční hodnota $\boldsymbol{\lambda}$ pro každý snímek byla $\mathbf{0}$ a v momentě, kdy hodnota $\boldsymbol{\lambda}$ dosáhla hodnoty $\mathbf{1}$, tak skript nahradil dvojici interpolovaných snímků dalšími následujícími. Testovací sady snímků bylo totožné jako v předcházejícím experimentu.

$$\lambda_{n+1} = \lambda_n + s \ * t$$

Obrázek 5.11: Rovnice pro výpočet $\boldsymbol{\lambda}$. Proměnná **s** je parametr rychlosti interpolace a proměnná **t** je čas ve vteřinách, který uběhl od předchozího zpracovaného snímku.

Experiment odhalil, že přechod pomocí lineární interpolace mezi snímky byl plynulý mezi všemi testovacími sadami.

Kapitola 6

Návrh řešení

Tato kapitola je věnována finálnímu návrhu řešení aplikace. Návrh vychází z experimentů, které jsou popsány v kapitole 5. Při návrhu byl kladen důraz především na jednoduché a uživatelsky přívětivé ovládání. Nejprve bylo potřeba stanovit požadavky na aplikaci, provést jejich analýzu a navrhnout finální podobu aplikace.

6.1 Analýza požadavků

Cílem je vytvořit aplikaci, která umožní uživateli pomocí headsetu *Samsung Gear VR* přirozeným způsobem prohlížet digitalizovaný svět. Hlavní požadavky na aplikaci jsou:

- Zobrazení všesměrných fotografií pomocí *Samsung Gear VR*
- Umožnit uživateli přechod mezi jednotlivými snímky
- Uživatelsky přívětivé ovládání

Dále by bylo vhodné, aby měl uživatel možnost vytvářet si vlastní virtuální prohlídky, a tak aplikace byla více univerzální a nesloužila pouze pro prohlížení předem vytvořených prohlídek.

6.2 Rozdělení na dvě samostatné aplikace

Z analýzy požadavků vyplynulo, že by bylo vhodné, aby měl uživatel možnost vytvářet si virtuální prohlídky sám, a proto došlo k rozhodnutí rozdělit finální řešení na dvě samostatné aplikace. První aplikaci, která bude sloužit pro vytváření virtuálních prohlídek a druhou aplikaci, která bude sloužit k zobrazování virtuálních prohlídek, které byly pomocí první aplikace vytvořeny.

Na základě tohoto rozhodnutí je nutné stanovit požadavky na aplikaci pro vytváření virtuálních prohlídek:

- Vytváření prohlídek
- Nahrávání fotografií od uživatele
- Zpracování uživatelem nahraných fotografií
- Vkládání fotografií do prohlídek
- Vkládání prvků pro interakci s uživatelem

6.3 Návrh aplikace pro vytváření virtuálních prohlídek

V této kapitole je popsán návrh aplikace pro vytváření virtuálních prohlídek. Při návrhu aplikace byly zvažovány následující tři možnosti:

1. Vytvoření rozšíření do Unity editoru
2. Vytvoření samostatné aplikace pomocí Unity
3. Vytvoření samostatné webové aplikace

U první možnosti se nejedná o samostatnou aplikaci, ale pouze o rozšíření do *Unity* editoru. Toto rozšíření by Unity editor rozšířilo o nástroje pro tvorbu virtuálních prohlídek. Velkou nevýhodou tohoto přístupu by byla skutečnost, že uživatel by byl nucen k instalaci celého enginu *Unity* a být dobře seznámen s tímto nástrojem. Naopak toto řešení se jeví jako nejjednodušší z uvedených.

Druhá možnost předpokládá vytvoření samostatného editoru pomocí enginu *Unity*. Taková aplikace by využívala stejné zdroje jako aplikace pro prohlížení prohlídek. Vytvořené prohlídky by byly vyexportovány v podobě balíčku, který by byl následně otevřen pomocí aplikace pro prohlížení prohlídek. Výhodou by bylo použití jediného nástroje pro vytvoření obou aplikací, avšak aplikace by nebyla multiplatformní a bylo by nutné zajistit podporu knihoven, které využívá *Unity* editor.

Poslední možnost spočívá ve vytvoření samostatné webové aplikace. Tato aplikace by současně sloužila jako server a poskytovala uživatelem vytvořené prohlídky. Výhodou tohoto přístupu je, že aplikace by byla multiplatformní, protože ke spuštění by byl vyžadován pouze webový prohlížeč, a tudíž i uživatelsky přívětivější. Současně by uživatel nebyl nucen do aplikace pro prohlížení prohlídek dodávat vytvořené balíčky, protože aplikace by je získala sama ze serveru. Nevýhodou je, že aplikace by vyžadovala použití dalších technologií a celkové řešení by bylo komplexnější a zároveň by aplikace vyžadovala stálé připojení k internetu.

Po zhodnocení všech možností byla vybrána poslední varianta, tedy vytvoření samostatné webové aplikace a to především z důvodu uživatelsky přívětivějšího řešení. Technologie, které budou použity jsou popsány v kapitole 4.

6.3.1 Struktura aplikace

Aplikace bude rozdělena do dvou částí. První část bude seznam všech již vytvořených prohlídek včetně základních informací o každé prohlídce. V této části bude mít uživatel možnost vytvořit novou prohlídku, smazat stávající prohlídku nebo přejít k editaci vybrané prohlídky. Vytváření prohlídek bude řešeno formou vyskakovacího okna, které uživatele vyzve k zadání názvu prohlídky. Po vytvoření prohlídky bude uživatel přesměrován do samotného editoru.

Druhá část aplikace bude samotný editor virtuálních prohlídek, který bude řešen jako *single-page* aplikace. Uživatel bude mít možnost nahrát fotografie, které budou ve sférickém panoramatickém formátu. Z těchto fotografií bude možné dále vytvořit jednotlivé scény v prohlídce. Do každé scény bude možné umístit objekt reprezentující informaci ve scéně nebo objekt pro přechod na další snímek. Editor by měl být přehledný a uživatelský co nejpříjemnější k používání.

6.3.2 Návrh databáze

Jako databázová služba byla zvolena databáze *MySQL*[1], kterou podporuje framework *Laravel*. Rozhodnutí bylo učiněno zejména na základě předchozích zkušeností s touto databází. Také kvůli bezplatné licenci a velké podpoře ze strany komunity.

Dále je nutné stanovit entity v aplikaci. Vzhledem k tomu, že *Laravel* využívá objektově relační mapování pro práci s databází, tak každá entita bude mít odpovídající tabulku v databázi. Schéma databáze je na obrázku 6.1.

Entita VirtualTour

Tato entita představuje samotnou virtuální prohlídku. Mezi její atributy patří název prohlídky a reference na entitu scény, která je vybrána jako výchozí při otevření prohlídky. V databázi je reprezentována tabulkou `virtual_tours`.

Entita Scene

Každá prohlídka se skládá z entit scény, které byly do prohlídky vloženy. Scéna představuje místo v prohlídce, do kterého se může uživatel dostat. Každá scéna obsahuje referenci na prohlídku, do které patří, referenci na fotografii, která tvoří scénu a hodnoty rotace všech tří os. Scéna je reprezentována tabulkou `scenes`.

Entita Image

Uživatelem nahrané fotografie jsou interpretovány entitou `Image`. Každá fotografie zahrnuje referenci na virtuální prohlídku, ke které se váže, název fotografie, cestu k fotografii a velikost fotografie. Tato entita je reprezentována tabulkou `images`.

Entita Draggable

Tato entita není v databázi uložena. Jedná se o abstraktní entitu, která je předkem entit `Hotspot` a `Information`. Tyto entity mají společné atributy, a proto je vhodné použít abstraktní entitu.

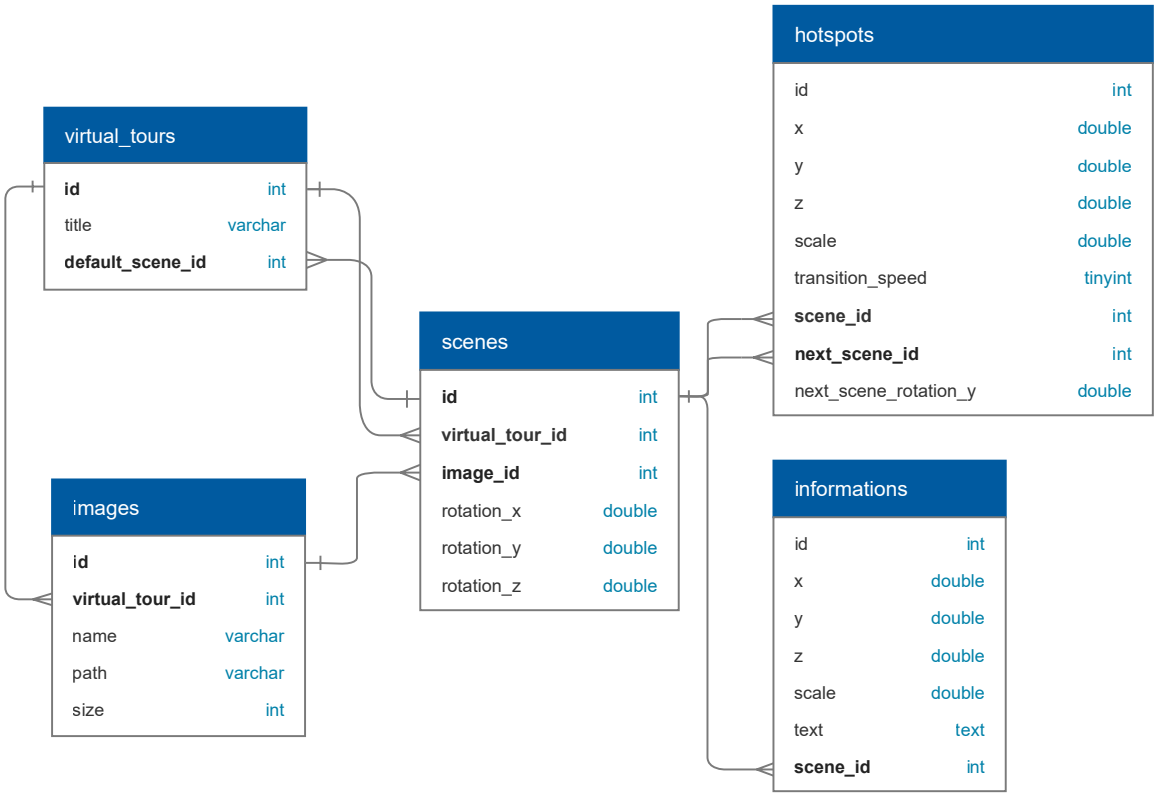
Entita Hotspot

K přechodu mezi jednotlivými snímky slouží entita `Hotspot`, která je reprezentována tabulkou `hotspots`. Tato entita obsahuje referenci na scénu, do které patří, referenci na scénu, na kterou bude přecházeno, souřadnice, měřítko, rychlost přechodu a rotaci osy `y`, která se váže k snímku, na který bude přecházeno.

Entita Information

Tato entita slouží k zobrazování uživatelem vložených informací ve scéně. Informace obsahuje text s obsahem informace, referenci na scénu, do které patří, souřadnice a měřítko. Tato entita je reprezentována tabulkou `informations`.

[1] MySQL - https://www.mysql.com



Obrázek 6.1: Schéma databáze aplikace pro vytváření virtuálních prohlídek znázorňující vazby mezi jednotlivými entitami.

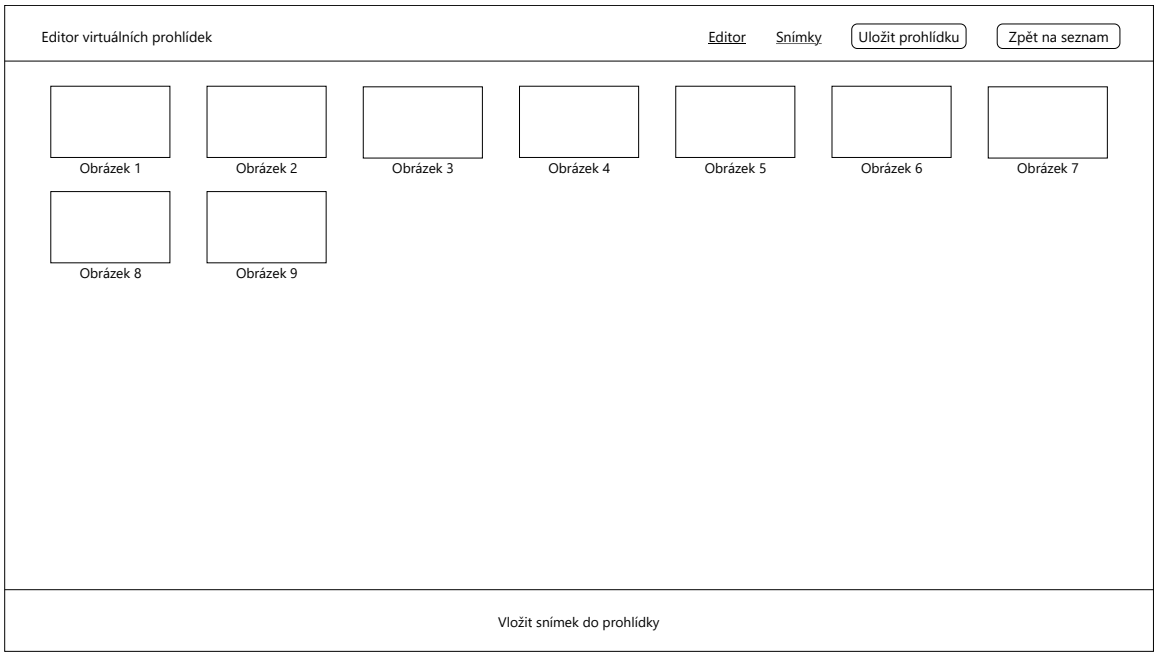
6.3.3 Návrh uživatelského rozhraní

V této kapitole je popsán návrh uživatelského rozhraní. Při návrhu byl kladen důraz především na přehlednost a snadné ovládání editoru. Editor by měl být intuitivní a uživatel by měl zvládnout se v editoru rychle zorientovat. Největší plochu aplikace by mělo zabírat okno s náhledem do editované scény, aby uživatel měl přehled, jak bude výsledná scéna vypadat.

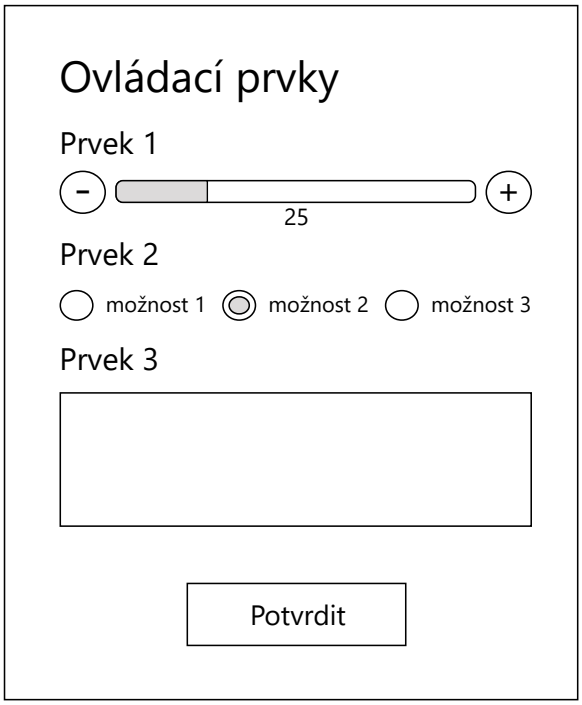
Dále by editor měl zahrnovat možnost pro výběr editované scény, vkládání nové scény do prohlídky a ovládací nástroje pro práci v editoru. Vkládání objektů s informací a přechodem na další snímek bude zajištěno pomocí kontextového menu, které bude aktivováno kliknutím do okna se scénou. Po výběru typu objektu bude na místo, kde uživatel kliknul, zvolený objekt vložen.

Vzhledem k povaze aplikace nebude při návrhu zvažován responzivní design, protože na zařízeních s malou plochou by bylo vytváření prohlídek velmi problematické. Na následujících obrázcích je znázorněno rozložení všech dílčích částí editoru.

Obrázek 6.2: Návrh editoru virtuálních prohlídek. V horní části se nachází menu s prvky pro navigaci a uložení prohlídky. Pod menu v levé části je hlavní okno se scénou, kde je zobrazena právě editovaná scéna. Na pravé straně je umístěn ovládací panel s ovládacími prvky. Ve spodní části aplikace se nachází panel se seznamem scén pro výběr editované scény a tlačítko pro přidání scény do prohlídky na levé straně.



Obrázek 6.3: Návrh správce snímků. V horní části se nachází menu. Ve spodní části se nachází oblast pro nahrávání nových snímků do prohlídky a v prostřední části se nachází seznam všech snímků, které byly do prohlídky nahrány.



Obrázek 6.4: Návrh ovládacího panelu s ovládacími prvky.

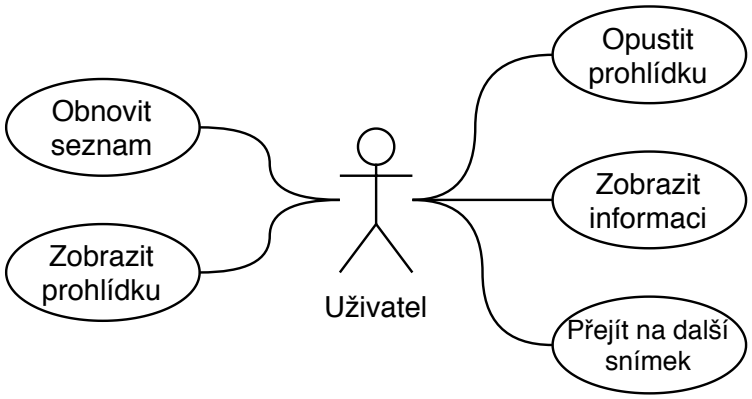
6.4 Návrh aplikace pro prohlížení virtuálních prohlídek

Tato kapitola popisuje návrh aplikace pro prohlížení virtuálních prohlídek. Aplikace bude realizována jako mobilní aplikace pomocí enginu *Uniy*, který je popsán v kapitole 2.5.2. Návrh vychází z experimentů, které byly popsány v kapitole 5 a také z návrhu aplikace pro vytváření virtuálních prohlídek, protože je nutné, aby aplikace vzájemně spolupracovaly.

6.4.1 Popis aplikace

Cílem aplikace je umožnit uživateli pomocí headsetu *Samsung Gear VR* přirozeným způsobem prohlížet digitalizovaný svět. Aplikace bude fungovat jako klient pro zobrazování virtuálních prohlídek, které byly vytvořeny pomocí aplikace pro vytváření virtuálních prohlídek. Aplikace se bude skládat ze dvou částí. První částí bude scéna s menu, které bude umístěno v prostoru. Menu bude obsahovat základní informace o aplikaci a návod k ovládání, aby se uživatel v aplikaci dokázal rychle zorientovat. Dále bude menu obsahovat seznam prohlídek, které byly vytvořeny. Výběrem prohlídky v seznamu dojde k jejímu zobrazení. Funkcionalita aplikace z pohledu uživatele je znázorněna diagramem případů užití na obrázku 6.5.

Druhá část aplikace bude samotná prohlídka, která se bude skládat z variabilního počtu scén. Každá scéna bude tvořena sférickou panoramatickou fotografií a bude obsahovat předem definované objekty pro přechod na další snímek a objekty pro zobrazení informace.



Obrázek 6.5: Diagram případů užití.

6.4.2 Uživatelská interakce

Uživatelská interakce byla součástí experimentu 5.4, který předpokládal interakci pomocí tlačítka na headsetu. Po otestování tohoto způsobu interakce několika respondenty, kdy někteří tento způsob hodnotili jako uživatelsky nepřívětivý, bylo rozhodnuto o změně způsobu interakce. Ten spočívá v pevně umístěném bodu ve středu kamery, který bude sloužit jako kurzor a bude směrovat na místo, na které se uživatel právě dívá. Jakmile dojde k interakci s některým prvkem, který umožňuje uživatelskou interakci, tak okolo kurzoru se objeví nekompletní kruh, který se začne plnit po směru hodinových ručiček. Jakmile dojde k vykreslení kompletního kruhu, dojde k aktivaci prvku, na který uživatel hleděl. Tento způsob uživatelské interakce implementuje například aplikace *Virtual Travel* v režimu virtuální reality. Tato aplikace je popsána v kapitole 2.6.2. Ukázka uživatelské interakce pomocí plnícího se kruhu je na obrázku 6.6.

Obrázek 6.6: Ukázka uživatelské interakce pomocí plnícího se kruhu [15].

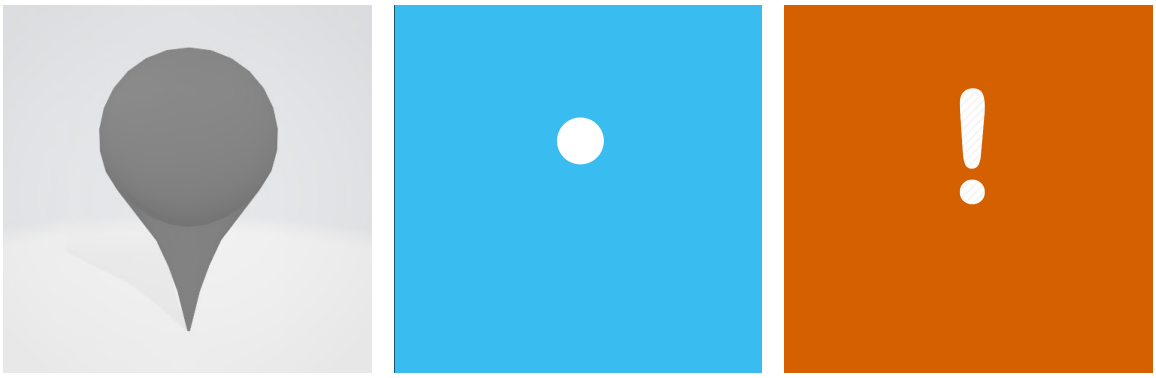
6.4.3 Objekty pro uživatelskou interakci

Kromě hlavního menu se budou v aplikaci vyskytovat dva typy objektů pro uživatelskou interakci. Tyto objekty bude reprezentovat 3D model špendlíku, na který bude aplikována textura. Model špendlíku je vidět na obrázku 6.7a. Tento model byl vytvořen modelovacím nástrojem *Blender*[2].

První z objektů bude fungovat jako přechod na další snímek. Způsob přechodu na další snímek byl ověřen experimentem, který je popsán v kapitole 5.6. Aby byla prohlídka více

[2]Blender - https://www.blender.org/

interaktivní, tak bude obsahovat i objekty s informacemi, které se po uživatelské interakci přemění na plátno s textem.



(a) 3D model špendlíku.

(b) Textura pro objekt s přechodem na další snímek.

(c) Textura pro objekt s informací.

Obrázek 6.7: 3D model špendlíku a textury objektů pro uživatelskou interakci.

6.5 Komunikace mezi aplikacemi

Server, na kterém bude běžet aplikace pro vytváření virtuálních prohlídek bude obsahovat koncové body, na které se bude aplikace pro prohlížení prohlídek dotazovat. První z nich bude poskytovat seznam všech prohlídek, které byly dosud vytvořeny. Každá prohlídka bude obsahovat pouze nezbytnou část dat, která budou potřebná pro zobrazení v seznamu prohlídek. Součástí každé prohlídky bude url adresa, na které se nachází kompletní data prohlídky. Po zvolení konkrétní prohlídky v menu dojde k dotazu na tuto url adresu a aplikace tak získá veškerá data o prohlídce, stáhne všechny fotografie a zobrazí prohlídku uživateli v headsetu.

Kapitola 7

Implementace a vyhodnocení

Tato kapitola obsahuje popis implementace a vyhodnocení finálního řešení. Implementace vychází z návrhu, který byl popsán v předchozí kapitole.

7.1 Implementace aplikace pro vytváření prohlídek

V této kapitole je popsána implementace aplikace pro vytváření virtuálních prohlídek. Aplikace je rozdělena na klientskou a serverovou část. Při implementaci se vycházelo z návrhu, který byl popsán v předchozí kapitole.

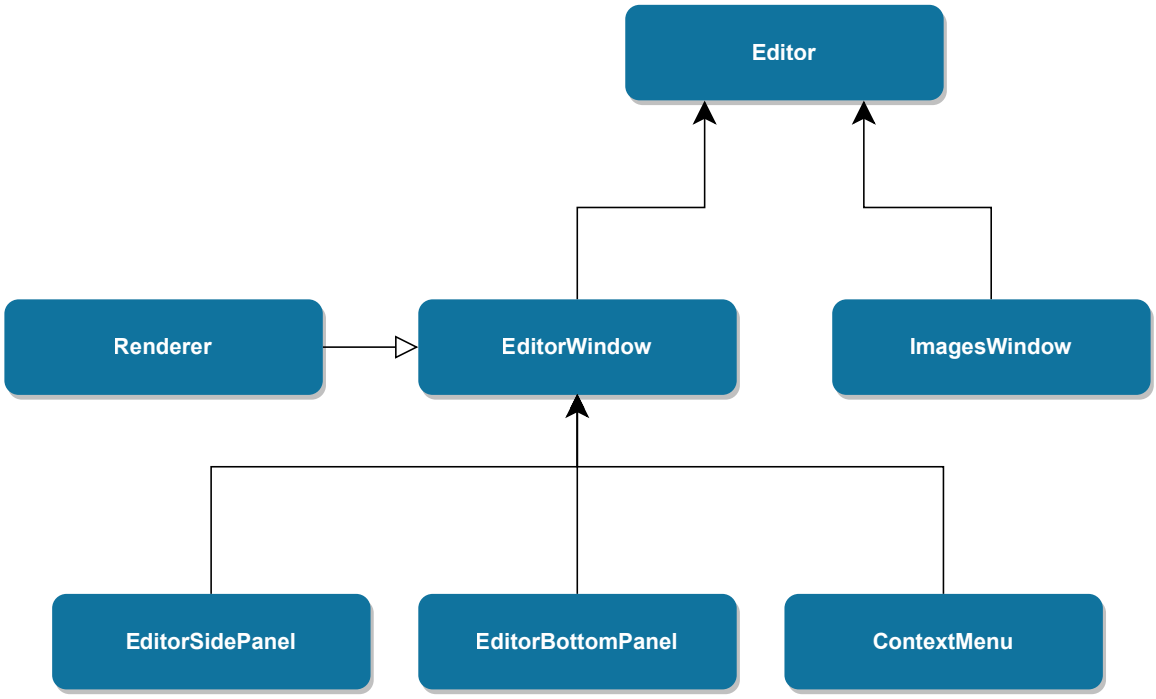
7.1.1 Klientská část aplikace

Klientská část aplikace je implementována jako *single-page* aplikace. Na rozdíl od běžných webových aplikací, poskytují *single-page* aplikace uživatelsky přívětivější prostředí, avšak vyžadují použití moderních webových prohlížečů. Celá aplikace komunikuje se serverovou částí pomocí AJAX[1] požadavků. Pro zasílání těchto požadavků byla použita javascriptová knihovna *axios*[2]. Všechny požadavky, které jsou mezi klientskou a serverovou částí zasílány, jsou ve formátu JSON.

Pro implementaci klientské části byl použit javascriptový framework *Vue*, který je rozebrán v kapitole 4.2 v kombinaci s knihovnou *Bootstrap*[3]. Všechny ikony, které jsou v aplikaci použity, pochází z balíčku *Font Awesome*[4]. Celá aplikace je tvořena stromovou strukturou několika komponent, která je zobrazena na obrázku 7.1. Každá komponenta představuje reálnou část aplikace. Při implementaci a rozdělení do jednotlivých komponent se vycházelo z návrhu uživatelského rozhraní, které je popsáno v kapitole 6.3.3.

V aplikaci jsou dále zaregistrovány globální komponenty, které lze použít na libovolném místě v aplikaci. První z nich je komponenta `Flash`[5], která je aktivována pomocí globální funkce `flash` a slouží k zobrazování notifikačních zpráv. Další globální komponentou je komponenta `VModal`[6]. Tato komponenta slouží k zobrazování vyskakovacích oken s libovolným obsahem. Poslední globální komponentou je `VueSweetalert2`[7], která je používána jako vyskakovací potvrzovací okno, například při mazání objektů z prohlídky.

[1] Asynchronous JavaScript and XML
[2] axios - https://github.com/axios/axios
[3] Bootstrap - https://getbootstrap.com/
[4] Font Awesome - https://fontawesome.com/
[5] Vue Flash - https://github.com/andyleach/vue-flash
[6] Vue.js modal - https://github.com/euvl/vue-js-modal
[7] vue-sweetalert2 - https://github.com/avil13/vue-sweetalert2



Obrázek 7.1: Stromová struktura komponent tvořící aplikaci.

Aplikace používá centralizovaný zdroj dat *Vuex*. Data, která jsou sdílena napříč komponentami, jsou definována následující strukturou:

```
state: {
    // aktualni scena
    currentScene: null,

    // aktualne vybrany objekt ve scene
    currentDraggableObject: null,

    // neulozeny seznam vsech scen v~prohlidce
    currentScenes: [],

    // seznam vsech scen v~podobe, v~jake jsou ulozeny v~databazi
    savedScenes: [],

    // seznam vsech fotografii, ktere byly nahrany do prohlidky
    uploadedImages: []
}
```

Položka `currentScene` obsahuje právě zobrazenou scénu, která je instancí třídy `Scene` a obsahuje také seznam všech objektů, které do dané scény patří. Další položka v centralizovaném zdroji dat `currentDraggableObject` reprezentuje zvolený objekt ve scéně, který je právě editován a je instancí třídy `Hotspot` nebo `Information`, které jsou potomky třídy `DraggableObject`. Aktuální seznam všech scén, které patří do prohlídky je uložen v `currentScenes`. V položce `savedScenes` je naopak uložen stav prohlídky, který kore-

sponduje se stavem v databázi. Všechny fotografie, které byly do prohlídky nahrány, jsou uloženy v `uploadedImages`.

Do centralizovaného zdroje dat *Vuex* byl vytvořen a vložen modul `sceneRenderer`, který využívá převážně komponenta `Renderer` a obsahuje data a funkce pro práci s knihovnou *Three.js*.

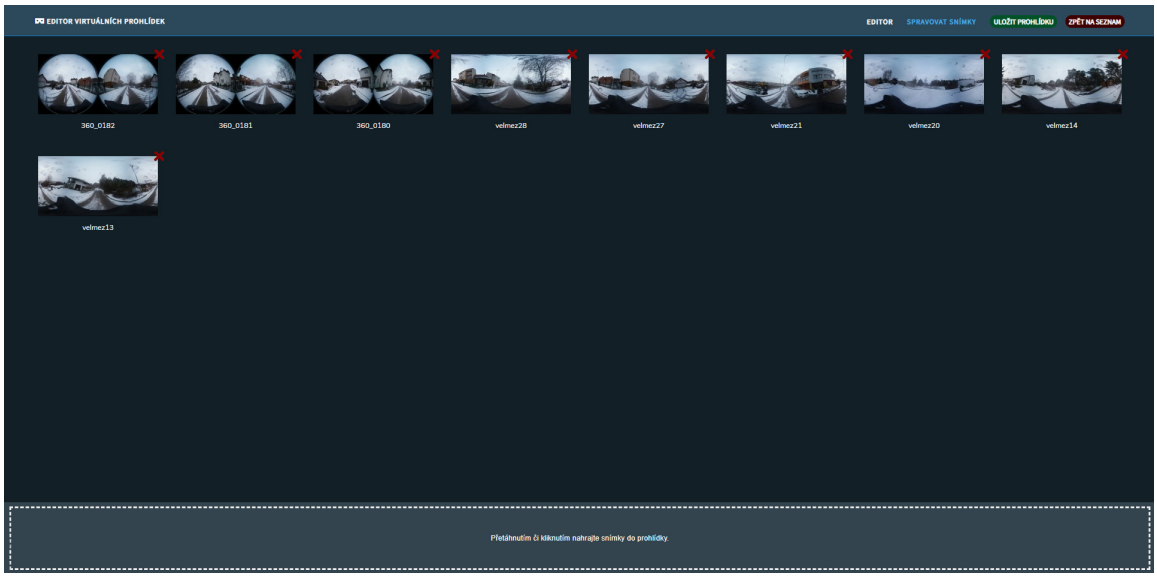
Komponenta Editor

Komponenta `Editor` je jádrem celé aplikace. Do této komponenty je vložen centralizovaný zdroj dat *Vuex*, a díky tomu všechny subkomponenty mají přístup k těmto datům. Dále obsahuje *Vue Router*, díky kterému lze přepínat mezi podstránkami, které jsou reprezentovány komponentami `ImagesWindow` a `EditorWindow`. Při implementaci došlo ke zjištění, že v souvislosti s knihovnou *Three.js* docházelo při přepínání mezi podstránkami k únikům paměti, protože knihovna neuvolňovala paměť korektně. Z toho důvodu byla komponenta `router-view`, která renderuje aktivní podstránku, obalena elementem `keep-alive`. Každá již inicializovaná komponenta, která je obalena tímto elementem, je uložena do paměti *cache*, a tak nedochází k opětovné inicializaci komponenty při přepínání podstránek.

Komponenta `Editor` při vytvoření provede dotaz na serverovou část a získá seznam všech uživatelem nahraných fotografií, které následně vloží do centralizovaného zdroje dat.

Komponenta ImagesWindow

Tato komponenta slouží jako podstránka pro zobrazení a nahrávání fotografií. Stránka obsahuje seznam všech nahraných fotografií do prohlídky včetně tlačítka pro odstranění fotografie. Náhled na stránku s fotografiemi je zobrazen na obrázku 7.2. Ve spodní části stránky se nachází oblast pro nahrávání fotografií, která je řešena pomocí komponenty `Vue2-Dropzone`[8] a umožňuje dělené nahrávání fotografií. Velikost chunku byla zvolena na 1 Mb. Podporované formáty fotografií jsou jpeg a png. Náhled na nahrávací oblast fotografií je na obrázku 7.3.



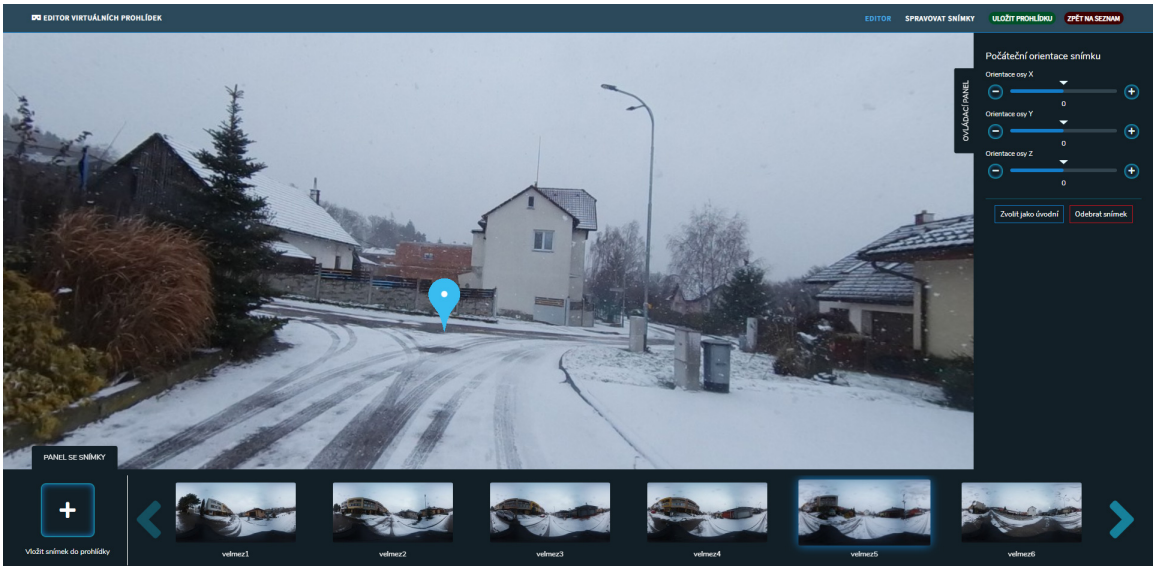
Obrázek 7.2: Náhled na stránku s fotografiemi.

[8] vue-dropzone - https://github.com/rowanwins/vue-dropzone

Obrázek 7.3: Náhled na část stránky s nahráváním fotografií.

Komponenta EditorWindow

Komponenta `EditorWindow` reprezentuje podstránku se samotným editorem. Rozšiřuje komponentu `Renderer`, ve které je umístěna veškerá funkcionalita pro renderování scény. Obsahuje subkomponenty `EditorSidePanel`, `EditorBottomPanel` a `ContextMenu`. V této komponentě je umístěno okno se scénou, které je renderováno pomocí *Three.js*. Dále obsahuje funkce pro vkládání objektů do scény, které jsou volány při kliknutí na položku v kontextovém menu. Náhled na editor virtuálních prohlídek je na obrázku 7.4.



Obrázek 7.4: Náhled na editor virtuálních prohlídek.

Komponenta Renderer

V této komponentě je uložena veškerá funkcionalita pro zobrazení scény pomocí knihovny *Three.js*. Po inicializaci komponenty dojde k inicializaci knihovny *Three.js*. Inicializace spočívá ve vytvoření rendereru `WebGLRenderer`, kterému je předána velikost plátna, ve kterém bude scéna renderována. Následně je vytvořena kamera `PerspectiveCamera` a `Raycaster`, který bude sloužit k detekci kolizí s objekty ve scéně. Následně je renderer vložen do elementu, který slouží jako plátno.

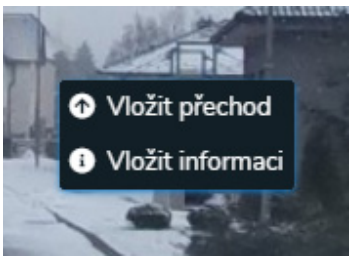
Posledním krokem inicializace je registrace událostí pro stisknutí tlačítka myši, uvolnění tlačítka myši, pohyb myši a změna velikosti okna. Každá z těchto událostí má svůj vlastní *event listener*. Při změně velikosti okna dojde k přepočítání nové velikosti plátna. Zbylé události jsou důležité pro uživatelskou interakci. Při kliknutí do scény se pomocí prvku `Raycaster` a metody `intersectObjects` detekuje, zda uživatel kliknul na objekt ve scéně. Pokud byl objekt detekován, tak je vložen do centralizovaného zdroje dat jako aktuálně

vybraný objekt ve scéně. V opačném případě pokud uživatel kliknul mimo objekt, tak dojde ke zrušení výběru nebo k otevření kontextového menu, případně zavření kontextového menu.

Při kliknutí do scény si komponenta uloží počáteční souřadnice, které jsou následně použity při pohybu myši pro výpočet nových souřadnic kamery při otáčení se ve scéně nebo při přemisťování objektů ve scéně.

Komponenta ContextMenu

Tato komponenta reprezentuje kontextové menu pomocí kterého se vkládají objekty do scény. Využívá javascriptovou knihovnu *Popper.js*[9]. Kontextové menu je aktivováno kliknutím do scény a nabízí uživateli vložení objektu pro přechod na další snímek nebo objekt s informací. Náhled na kontextové menu je na obrázku 7.5.



Obrázek 7.5: Náhled na kontextové menu.

Vybraný objekt je vložen na místo, kde uživatel při otevření kontextového menu kliknul. Bylo nutné zajistit převod z 2D souřadnic obrazovky, na kterých bylo kliknuto do 3D světa, ve kterém bude objekt umístěn. Zdrojový kód pro převod souřadnic je znázorněn na výpisu 7.1.

```
this.click3DCoords.set(
    (event.clientX / this.canvasWidth) * 2 - 1,
    - ((event.clientY - this.headerOffset) / this.canvasHeight) * 2 + 1,
    -1
);
this.click3DCoords.unproject(this.camera);
this.click3DCoords.sub(this.camera.position).normalize();
```

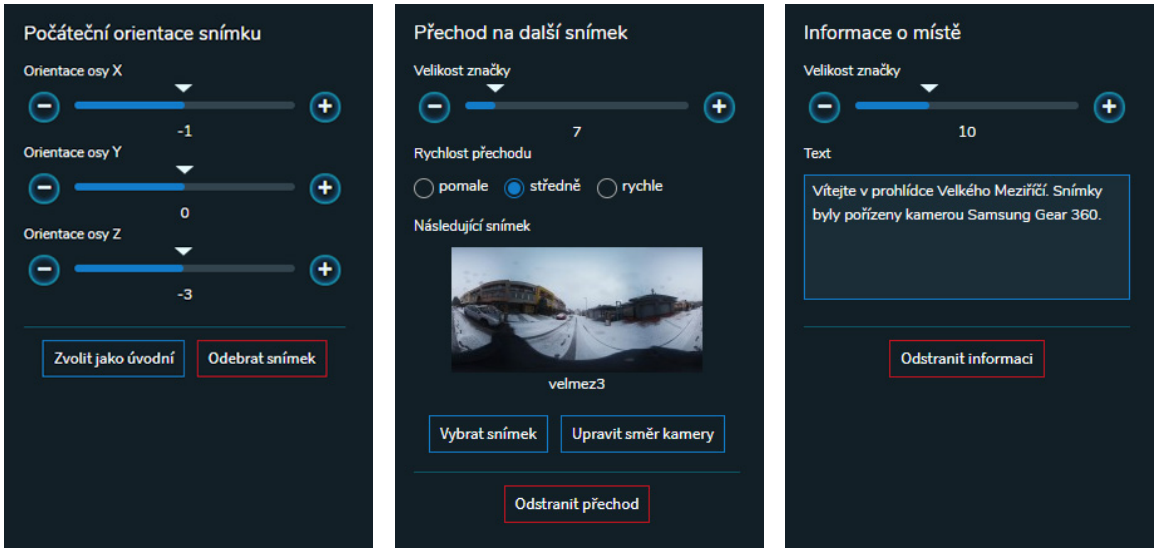
Výpis 7.1: Převod 2D souřadnic obrazovky do 3D světa [11].

Objekty ve scéně jsou reprezentovány 3D modelem, který je shodný s 3D modelem, který je použit v aplikaci pro prohlížení prohlídek. Tyto objekty jsou popsány v kapitole 6.4.3. Objekty jsou do prohlídky vloženy pomocí třídy `OBJLoader`.

Komponenta EditorSidePanel

Komponenta `EditorSidePanel` představuje ovládací panel v pravé části editoru. Jejím úkolem je nabídnout uživateli možnost měnit nastavení scény a objektů ve scéně. Jedním z ovládacích prvků je posuvník s tlačítky, jehož základ tvoří `vue-slider-component`[10]. Obsah panelu se dynamicky mění na základě toho, zda je právě vybrán některý objekt ve scéně a o jaký typ objektu se jedná. Náhled na ovládací panel je na obrázku 7.6.

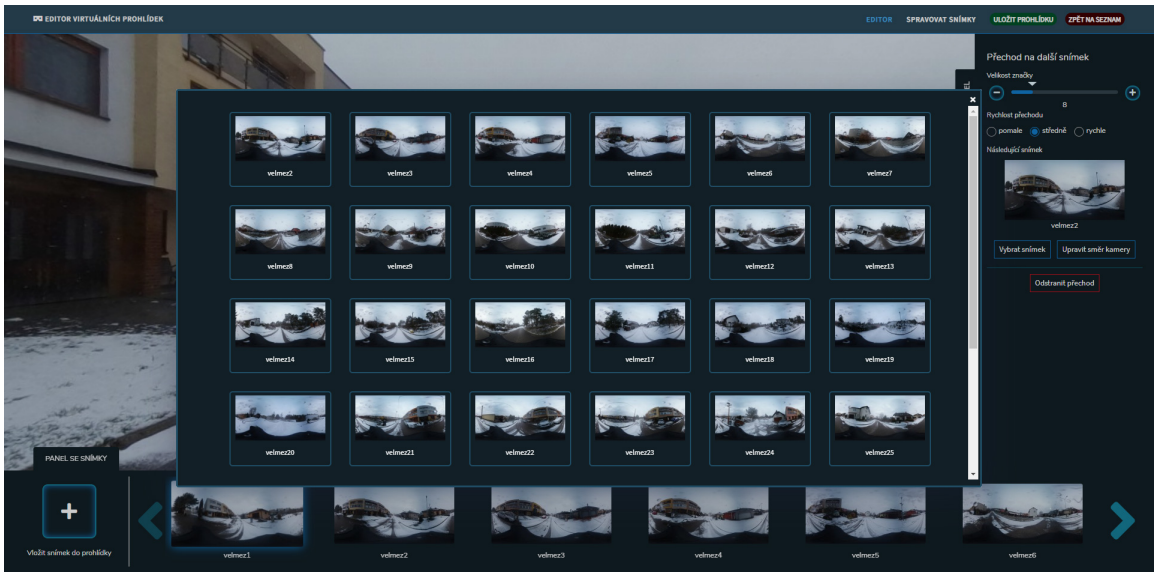
[9] Popper.js - https://github.com/FezVrasta/popper.js
[10] vue-slider-component - https://github.com/NightCatSama/vue-slider-component



(a) Nastavení scény. (b) Nastavení přechodu. (c) Nastavení informace.

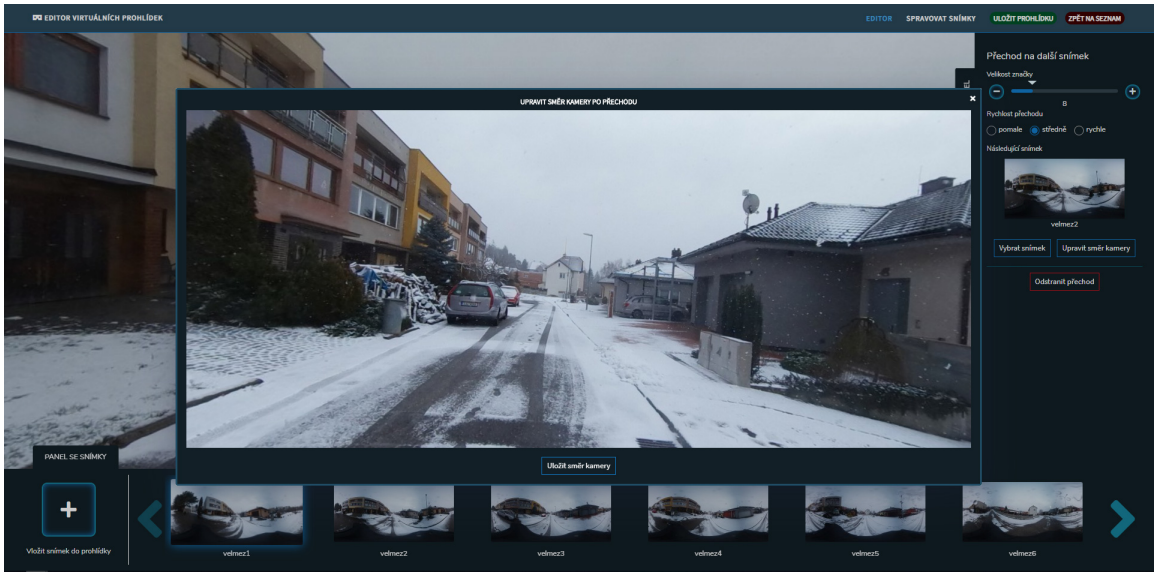
Obrázek 7.6: Obsah ovládacího panelu. Na levé straně se nachází obrázek s nastavením scény. Uživatel může upravit rotaci snímku, nastavit scénu jako výchozí a odstranit scénu z prohlídky. Uprostřed se nachází nastavení přechodu na další snímek. Lze měnit velikost objektu ve scéně, který slouží k přechodu, nastavit rychlost přechodu na další snímek, zvolit cílovou scénu a upravit směr kamery, kterým bude kamera otočena po přechodu. Vpravo se nachází nastavení objektu s informací, kterému lze změnit velikost a přiřadit text, který bude zobrazován.

U objektu pro přechod na další snímek má uživatel možnost zvolit cílovou scénu a směr kamery, kterým bude kamera otočena po přechodu. Po kliknutí na příslušné tlačítko se uživateli zobrazí vyskakovací okno pomocí komponenty `VModal`, kde si zvolí scénu nebo upraví směr kamery. Náhled na vyskakovací okno s výběrem scény je na obrázku 7.7.



Obrázek 7.7: Náhled na vyskakovací okno s výběrem scény.

Vyskakovací okno pro změnu směru kamery umožňuje uživateli pohyb pouze po horizontální rovině. Okno obsahuje vlastní instanci *Three.js*, která je inicializována podobným způsobem jako u komponenty `Renderer`. V tomto případě nedochází ke změně pohledu kamery, ale dochází k rotaci celé scény po ose `y`. Rotace os `x` a `z` korespondují s nastavením, které přísluší cílové scéně, aby náhled na scénu byl shodný. Po uložení směru kamery je rotace osy `y` uložena do interního stavu objektu pro přechod. Náhled na vyskakovací okno s nastavením směru kamery je na obrázku 7.7.



Obrázek 7.8: Náhled na vyskakovací okno s nastavením směru kamery.

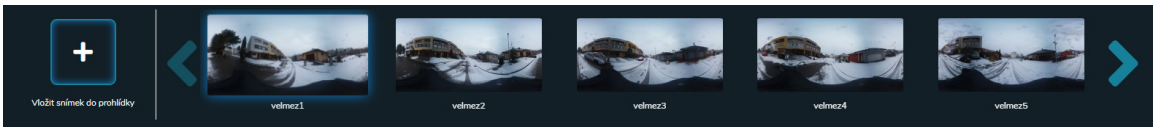
Komponenta EditorBottomPanel

Tato komponenta v aplikaci figuruje jako panel se snímky, který se nachází ve spodní části pod oknem se scénou. Obsahuje seznam všech scén, které byly do prohlídky vloženy. Seznam scén je zobrazen pomocí komponenty `Vue Carousel`[11]. Kliknutím na scénu dojde ke změně aktivní scény v centralizovaném zdroji dat a vykreslení příslušné scény včetně všech objektů, které zvolená scéna obsahuje. Vykreslení nové scény spočívá ve zrušení předchozí scény, která byla vykreslena knihovnou *Three.js* a uvolnění paměti. Následně je fotografie, která patří ke zvolené scéně, načtena pomocí třídy `TextureLoader` jako textura. Z této textury je následně vytvořen materiál, který je reprezentován třídou `MeshBasicMaterial`. Dále je vytvořena geometrie koule `SphereBufferGeometry`, která má invertovanou osu `x`, aby plocha koule směrovala směrem dovnitř. Kombinací geometrie koule a materiálu je následně třídou `Mesh` vytvořena *mesh* geometrie. Tato geometrie je následně umístěna ve scéně tak, aby se kamera nacházela uprostřed této geometrie. Protože byla invertována osa `x` na geometrii koule, tak fotografie je vykreslena na vnitřní stranu této koule a je vidět pomocí kamery.

V levé části panelu se nachází tlačítko pro vložení nové scény do prohlídky. Po kliknutí na tlačítko se uživateli zobrazí vyskakovací okno se seznamem všech fotografií, které uživatel do editoru nahrál. Každá fotografie může být k vytvoření nové scény použita pouze jednou, a proto fotografie, které již byly použity k vytvoření scény, se v tomto seznamu nezobrazují.

[11] Vue Carousel - https://github.com/SSENSE/vue-carousel

Vzhled vyskakovacího okna je totožný s vyskakovacím oknem, které slouží k výběru scény pro přechod na další snímek, které je zobrazeno na obrázku 7.7.



Obrázek 7.9: Náhled na panel se snímky.

7.1.2 Serverová část aplikace

Vzhledem k tomu, že editor je implementován jako *single-page* aplikace, tak serverová část aplikace slouží především jako API pro práci s databází. Logika serverové části aplikace se nachází ve třídě `VirtualTourController`. API pro klientskou část editoru zahrnuje vkládání prohlídek a fotografií, načítání prohlídek a fotografií a mazání prohlídek a fotografií. Dále API aplikace pro zobrazování virtuálních prohlídek zahrnuje načítání seznamu prohlídek a načítání dat pro konkrétní prohlídku.

Nahrávání fotografií

Pro nahrávání fotografií do prohlídky je použita knihovna *Laravel chunked upload*[12], která umožňuje dělené nahrávání souborů. Po úspěšném nahrání fotografie na server je následně změněna její velikost na 4096x2048 pixelů knihovnou *Intervention Image*[13]. Tato velikost je dostačující a větší fotografie by kladly vysoké nároky na objem přenesených dat po síti. Současně je pro fotografii vytvořena i náhledová fotografie o velikosti 200x100 pixelů. Tyto náhledové fotografie jsou využity jako miniatury v editoru, ale následně i v aplikaci pro prohlížení prohlídek.

7.2 Implementace aplikace pro prohlížení virtuálních prohlídek

Tato kapitola popisuje implementaci aplikace pro prohlížení virtuálních prohlídek. Pro implementaci aplikace byl použit engine *Unity*. Konfigurace projektu byla shodná jako v kapitole 5.1.

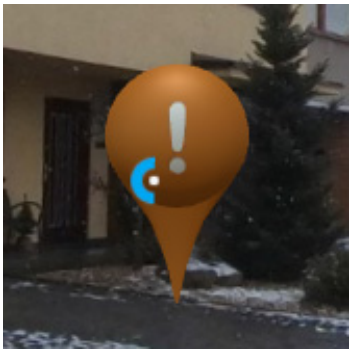
Uživatelské rozhraní

Z návrhu vyplynulo, že uživatelská interakce by měla být implementována formou plnícího se kruhu při interakci s objekty. Pro tyto účely byl použit balíček *VR UIKit*[14], který tuto formu uživatelské interakce umožňuje. Součástí balíčku jsou také prefabrikované objekty pro vytváření uživatelského menu. Tento balíček využívá pouze nativní podporu virtuální reality, kterou Unity nabízí a nevyžaduje tak žádné závislosti, jako tomu bylo u experimentu 5.4, který využíval balíček *Oculus Integration*. Čas potřebný pro vyplnění kruhu a následnou

[12] Laravel chunked upload - https://github.com/pionl/laravel-chunk-upload
[13] Intervention Image - https://github.com/Intervention/image
[14] VR UIKit - https://assetstore.unity.com/packages/tools/gui/vr-uikit-bootstrap-your-vr-app-with-ease-128236

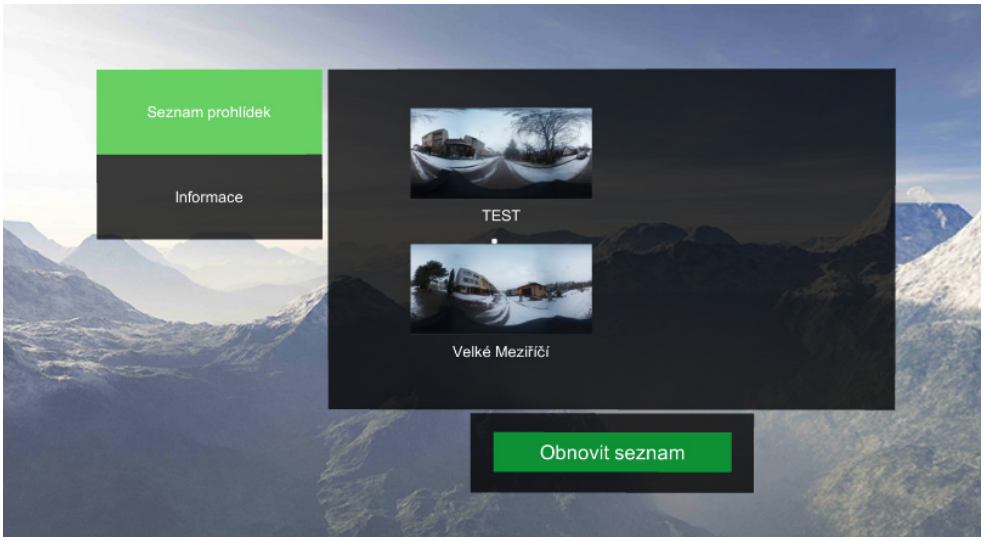
aktivaci objektu byl po zpětné vazbě od několika respondentů zvolen na jednu vteřinu jako nejpřijatelnější.

Pro všechny objekty, které umožňují uživatelskou interakci, byla vytvořena nová vrstva `Gazable`. Tato vytvořená vrstva byla následně nastavena jako maska pro události komponentě `PhysicsRaycaster` v hlavní kameře. Díky tomu lze provádět interakci pouze s takto označenými objekty. Uživatelská interakce je zachycena na obrázku 7.10.



Obrázek 7.10: Uživatelská interakce pomocí plnícího se kruhu.

Hlavní menu se skládá za dvou záložek. První záložkou je seznam všech prohlídek. Každá položka v seznamu obsahuje název prohlídky a náhledový obrázek. Ve spodní části pod seznamem se nachází tlačítko pro obnovení seznamu, které po aktivaci provede dotaz na server a aktualizuje seznam. Druhá záložka obsahuje informace o prohlídce včetně stručného návodu k ovládání aplikace. Hlavní menu bylo vytvořeno pomocí balíčku *VR UIKit*, který nabízí prefabrikované objekty pro tvorbu menu. Hlavní menu je zobrazeno na obrázku 7.11. Použité pozadí pochází z balíčku *Cope! Free Skybox Pack*[15].



Obrázek 7.11: Hlavní menu aplikace. V levé části se nachází záložky. Uprostřed se nachází seznam prohlídek a pod seznamem se nachází tlačítko pro obnovení seznamu.

[15]Cope! Free Skybox Pack - https://assetstore.unity.com/packages/2d/textures-materials/sky/cope-free-skybox-pack-22252

Zobrazení seznamu prohlídek

Celá aplikace je řízena pomocí komponenty `TourManager`. Při startu aplikace a aktivaci této komponenty dojde k zavolání metody `LoadVirtualTours`. Tato metoda využívá komponentu `TourLoader`, která zodpovídá za komunikaci se serverem a stahování prohlídek. Je zavolána metoda `FetchVirtualToursList`, které je předána url adresa koncového bodu serveru se seznamem prohlídek. Návratová hodnota této metody je `IEnumerator`, a proto může být volání této metody předáno jako parametr metodě `StartCoroutine`, kterou implementuje Unity. Tato metoda umožňuje pozastavit vykonávání skriptu a pokračovat ve chvíli, kdy byla operace dokončena a tím neblokovala vykonávání dalších skriptů v aplikaci.

Data jsou ze serveru stáhnutá pomocí třídy `UnityWebRequest`. Odpověď serveru je zachycena na obrázku 7.12. *Unity* používá pro práci s formátem JSON takzvanou serializaci. To znamená, že data ve formátu JSON jsou konvertována na objekty. Byla vytvořena třída `TourListResource`, která má shodné atributy jako odpověď od serveru. Odpověď od serveru je následně konvertována do instance této třídy. Následně je z konvertované odpovědi vytvořen seznam prohlídek, kde prohlídka je reprezentována třídou `VirtualTourListItem`. Pro každou prohlídku v seznamu bylo nutné stáhnout její náhledový obrázek pomocí třídy `UnityWebRequestTexture` a uložit jej do paměti.

Obrázek 7.12: Odpověď serveru obsahující seznam prohlídek ve formátu JSON.

Jakmile je seznam prohlídek úspěšně stažen a uložen do paměti, dojde k jeho vložení do seznamu v hlavním menu. Seznam prohlídek v hlavním menu má na starosti komponenta `CardListManager`. Pro každou prohlídku je vytvořena instance třídy `Card`, která představuje položku v seznamu. Každé položce je přiřazen titulek, náhledový obrázek a url adresa, která odkazuje na koncový bod serveru s kompletními daty prohlídky. Následně je každé kartě přiřazen *event handler* `CardClickHandler`, který je aktivován po kliknutí na kartu a zahájí načítání vybrané prohlídky. Pod seznamem s prohlídkami se nachází tlačítko pro obnovení seznamu. Po jeho aktivaci dojde k odstranění seznamu prohlídek z paměti a je znovu zavolána metoda `LoadVirtualTours`.

Zobrazení prohlídky

Po výběru prohlídky v menu dojde k jejímu stažení ze serveru. Stejně jako u zobrazení seznamu prohlídek je ke stažení prohlídky použita komponenta `TourLoader`. Pro získání dat prohlídky je zavolána metoda `FetchVirtualTour`, které je jako parametr předána url adresa koncového bodu serveru, na kterém se nachází data prohlídky. Dále je nutné data

opět konvertovat na objekt. Pro tato data byla vytvořena třída `TourResource`. Odpověď od serveru a obdržená data jsou zobrazeny na obrázku 7.13. Nejprve dojde k vytvoření samotné prohlídky, která je reprezentována třídou `VirtualTour`. Následně jsou inicializovány všechny scény `Scene`, které prohlídka obsahuje a všechny objekty s informací `Information`. Současně dojde ke stažení fotografie pro každou scénu a jejímu uložení do paměti. Jakmile jsou všechny scény inicializovány, dojde k inicializaci objektů pro přechod `Hotspot` pro každou scénu a nastavení reference na následující scénu. Uživatel je v průběhu informován o stavu načítání prohlídky pomocí skriptu `TourProgressScript`, který zobrazuje průběh stahování v hlavním menu.


```
id:                          2
title:                       "TEST"
scenes:
  0:
      id:                    34
    image:
        id:                  39
        name:                "velmez28"
      url:                   "https://tour.sdev.cz/storage/tours/images/2/img/velmez28b69fd137e22078e489f9d273212dbf65.jpg"
      prev_url:              "https://tour.sdev.cz/storage/tours/images/2/prev/velmez28b69fd137e22078e489f9d273212dbf65.jpg"
    rotationX:               5
    rotationY:               0
    rotationZ:               -4
    hotspots:
      0:
          id:                726
          x:                 -74.76
          y:                 -3.99
          z:                 -4.46
          scale:             6
          transition_speed:  2
          next_scene_id:     36
          next_scene_rotation_y: 58.7
    informations:
      0:
          id:                4
          x:                 7.03
          y:                 10.53
          z:                 73.92
          scale:             10
          text:              "Zde se nachazi informace"
```


Obrázek 7.13: Odpověď serveru obsahující část vybrané prohlídky ve formátu JSON.

Po úspěšném stažení prohlídky dojde k zavolání metody `LoadScene`, kterou obsahuje komponenta `SceneManager` a které je předána jako parametr výchozí scéna prohlídky. Tato komponenta slouží k řízení prohlídek. Scéna v *Unity* je vytvořena stejným způsobem jako v experimentu 5.2 s tím rozdílem, že bylo nutné modifikovat shader. Modifikace spočívala v upravení shaderu tak, aby bylo možné staženou fotografii aplikovat přímo na povrch koule bez předchozího importu. Další modifikací bylo přidání možnosti rotace textury po všech třech osách.

Úlohou metody `LoadScene` je aplikace fotografie jako textury na povrch koule, která tvoří scénu a nastavení rotace všech tří os. Následně dojde k instanciaci všech objektů ve scéně pro přechod na další snímek a objektů pro zobrazení informace včetně nastavení souřadnic a velikosti. Všechny objekty jsou umístěny do společného rodičovského objektu `SceneObjectsContainer`. Vzhledem k tomu, že *Unity* pracuje se souřadným systémem tro-

chu odlišně než knihovna *Three.js*, bylo nutné modifikovat souřadnice, na které byly objekty umístěny. Modifikace spočívala v záměně a invertování některých hodnot obdržených souřadnic. Nová souřadnice osy `x` byla nastavena jako invertovaná hodnota osy `z`, souřadnice osy `z` jako invertovaná hodnota osy `x` a hodnota osy `y` zůstala zachována.

Přechod na další snímek

Pro přechod na další snímek slouží objekty ve scéně, které byly vytvořeny při načítání scény. Tyto objekty jsou vytvářeny instanciací prefabrikovaného objektu `HotspotPrefab`, který se skládá z 3D modelu na který je aplikována textura a skriptu `HotspotScript`. Tento skript slouží k ovládání objektu a implementuje metody `OnPointerEnter`, `OnPointerExit` a `OnPointerClick`, které jsou volány při interakci s objektem. Objekt mění při interakci svoji velikost pomocí animačního enginu *DOTween*[16]. Objekt pro přechod na další snímek je znázorněn na obrázku 7.14.

Obrázek 7.14: Objekt pro přechod na další snímek.

Po aktivaci objektu je zavolána metoda `LoadNextScene` na komponentě `SceneManager`, které je předána reference na další scénu, směr kamery po přechodu, který je reprezentován rotací osy `y` a rychlost přechodu. Přechod na další snímek využívá lineární interpolaci jako v experimentu 5.6 s modifikací shaderu, která spočívala v přidání možnosti rotace cílového snímku ve všech třech osách. Před zahájením přechodu jsou ze scény odstraněny všechny objekty pro přechod a objekty pro zobrazení informace. Poté dojde k nastavení textury cílového snímku pro interpolaci a nastavení rotace os `x` a `z` cílového snímku. Při nastavení rotace osy `y` je nutné vzít v potaz aktuální směr kamery, aby po přechodu uživatel hleděl zvoleným směrem. To spočívá v odečtení aktuální rotace kamery v ose `y` od zvolené rotace osy `y` cílového snímku. Jakmile jsou všechny parametry nastaveny, dojde k prolnutí snímků pomocí lineární interpolace a plynulému přechodu na další snímek. Po dokončení přechodu dojde k načtení všech objektů pro aktuální scénu. Posledním krokem je nastavení rotace osy `y` objektu `SceneOjectsConrainer`, ve kterém se nachází všechny objekty ve scéně, aby byly umístěny na správném místě. Tato rotace je vypočítána odečtením nově spočítané rotace aktuální scény od rotace scény, která byla zvolena v editoru virtuálních prohlídek.

Zobrazení informace

K zobrazení informace slouží objekty ve scéně, které jsou vytvořeny instanciací prefabrikovaného objektu `InformationPrefab`. Tento objekt se skládá z 3D modelu na který

[16] DOTween - http://dotween.demigiant.com/

je aplikována textura a dále z plátna, na kterém bude zobrazena informace. Objekt je ovládán pomocí skriptu `InformationScript` a implementuje metody `OnPointerEnter`, `OnPointerExit` a `OnPointerClick`, které jsou volány při interakci s objektem. Stejně jako u objektu pro přechod na další snímek dochází ke změně jeho velikosti při interakci. Objekt s informací je zobrazen na obrázku 7.15.

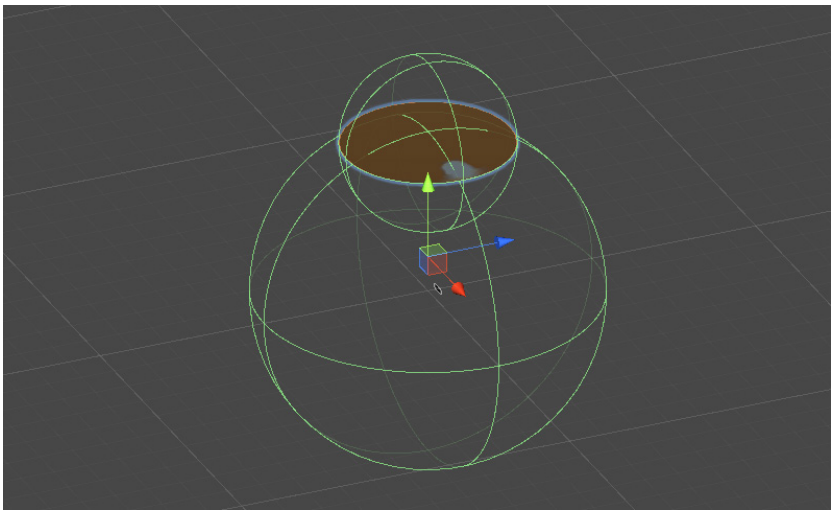
Po aktivaci objektu dojde ke skrytí objektu a na jeho místě se objeví plátno s informací, které zůstane aktivní dokud na něj bude uživatel mířit kurzorem. Po opuštění plátna dojde k jeho skrytí a na místě se znovu objeví objekt s informací.

(a) Objekt s informací. (b) Plátno s informací.

Obrázek 7.15: Na levé straně se nachází objekt s informací a na pravé straně plátno s informací po aktivaci objektu.

Návrat do menu

Bylo zapotřebí vymyslet způsob, kterým se uživatel vrátí z prohlídky do hlavního menu. Toho bylo docíleno tím, že na vrchol scény byl umístěn neviditelný objekt ve tvaru kruhu. Tento kruh se nachází ve vrstvě `Gazable`, aby s ním mohl uživatel interagovat. Po aktivaci tohoto objektu je zavolána metoda `ExitScene` na třídě `SceneManager` a tím dojde k návratu do hlavního menu a uvolnění prohlídky z paměti. Náhled na okno scény s objektem pro návrat do menu je na obrázku 7.16.



Obrázek 7.16: Náhled na okno scény. Objekt pro návrat do menu se nachází na vrcholu objektu, ve kterém je zobrazena aktuální scéna prohlídky a je barevně odlišen.

7.3 Vyhodnocení

Tato kapitola se zabývá vyhodnocením výsledného řešení. V první části se nachází porovnání vytvořené prohlídky v editoru s prohlídkou zobrazenou v aplikaci pro prohlížení prohlídek. Druhá část kapitoly se zabývá uživatelským testováním a v poslední části je popsán návrh na pokračování ve vývoji.

7.3.1 Porovnání vytvořené prohlídky

Ze všeho nejdůležitější bezesporu bylo, aby vytvořená prohlídka v editoru se shodovala s prohlídkou zobrazenou pomocí aplikace v headsetu. To spočívalo především v tom, aby všechny objekty ve scéně byly umístěny na správném místě a byly stejně velké. Dále všechny objekty pro přechod na další snímek odkazovaly na správný snímek a směr kamery po přechodu odpovídal. Byla vytvořena prohlídka, která se skládala z 33 fotografií. Následně byla vytvořená prohlídka porovnána se zobrazenou prohlídkou v headsetu. Porovnání jedné scény prohlídky se nachází na obrázku 7.17.

(a) Vytvořená scéna v editoru.

(b) Zobrazená scéna v headsetu.

Obrázek 7.17: Porovnání scény prohlídky, která byla vytvořena v editoru a zobrazené scény v headsetu. Pozice objektů včetně velikostí se shodují.

Prvním krokem bylo otestování, zda scéna, která byla zvolena jako výchozí se po načtení prohlídky skutečně zobrazila jako úvodní. Dále ověřit nastavenou rotaci pro jednotlivé scény a v nich umístěné objekty. Na závěr byly otestovány objekt s informací, zda správně zobrazují text a objekty pro přechod na další snímek, zda kamera je otočena zvoleným směrem po přechodu. Z porovnání vyplynulo, že všechny prvky v prohlídce fungují správně. Jedinou zjištěnou odchylkou bylo rozdílné zobrazení 3D modelů, které byly vkládány do scény i přes to, že byly použity shodné 3D modely i textury.

7.3.2 Uživatelské testování

Testování se zúčastnili 4 uživatelé. Uživatelé obdrželi sadu 20-ti fotografií a dostávali úkoly, které měli splnit. Byla pozorována jejich práce s aplikací a zpětná vazba, kterou poskytovali.

Prvním úkolem bylo vytvořit novou prohlídku a nahrát fotografie do prohlídky. Vytvořit prohlídku zvládli všichni uživatelé bez problému, ale jeden uživatel měl zprvu problém s nahráním fotografií, protože se snažil fotografie vložit pomocí tlačítka pro přidání scény.

Avšak nakonec se mu podařilo fotografie nahrát bez nutné asistence. Uživatelé si chválili možnost nahrávání více fotografií naráz a ukazatel průběhu nahrávání fotografie.

Další úkol spočíval ve vytvoření scén z libovolných fotografií, které byly nahrány v předchozím úkolu a vytvoření přechodu mezi nimi. Vytvořit scény zvládli všichni uživatelé bez zaváhání. Dva uživatelé měli problém s vytvořením přechodu mezi snímky. Hledali tlačítko pro vložení přechodu v ovládacím panelu. Jeden uživatel musel být naveden a bylo mu ukázáno vkládání objektů pomocí kontextového menu. Nastavení směru kamery po přechodu bylo bez problémů.

Posledním úkolem bylo vložení libovolné informace do prohlídky. Vzhledem k tomu, že informace se vkládají do prohlídky stejným způsobem jako přechod na další snímek, tak tento úkol zvládli všichni uživatelé bez zaváhání.

Uživatelé si chválili přehlednost a plynulou práci s editorem. Jedinou výtkou bylo, že při vložení více scén do prohlídky se v seznamu scén hůře orientuje. Kromě vkládání objektů pomocí kontextového menu uživatelé neměli s používáním aplikace žádné větší potíže.

Následně byla uživatelům předána k testování aplikace pro zobrazování prohlídek. Jediným úkolem uživatelů bylo, aby si otevřeli vytvořenou prohlídku a vyzkoušeli si interakci se všemi objekty. Hned po spuštění aplikace se uživatelům zobrazí stránka s informacemi, kde se vyskytuje i popis ovládání aplikace, a proto všichni uživatelé neměli s ovládáním aplikace žádný problém. Celkově na ně působila aplikace dobrým dojmem a interakce pomocí plnícího kruhu se jim líbila. Jeden uživatel měl výtku ke způsobu navrácení do hlavního menu, avšak nedokázal navrhnout lepší řešení.

7.3.3 Pokračování ve vývoji

Implementované řešení je zaměřeno pouze na samotné vytváření a prohlížení virtuálních prohlídek. Nabízí se rozšíření webové aplikace pro vytváření virtuálních prohlídek o uživatelské účty. Každý uživatel by měl svůj vlastní účet a mohl spravovat pouze vlastní prohlídky. Dále by aplikace mohla být rozšířena o další objekty, které se mohou v prohlídce vyskytovat. Například vkládání fotografií nebo videozáznamů do prohlídky. Dalším užitečným rozšířením by byla implementace více druhů přechodů mezi snímky, a tak by si uživatel mohl vybrat, jaký efekt chce při přechodu na další snímek použít. Uživatelské testování odhalilo, že při více scénách v prohlídce je seznam nepřehledný, a tak by se aplikace dala rozšířit o možnost seskupování scén do skupin.

Kapitola 8

Závěr

Cílem této diplomové práce bylo seznámit se s problematikou virtuální reality a vývojem aplikací pro headset *Samsung Gear VR*. Na základě získaných znalostí pak bylo cílem navrhnout a implementovat aplikaci, která umožní přirozeným způsobem prohlížet digitalizovaný svět. Výsledné řešení bylo navíc rozšířeno o aplikaci pro vytváření virtuálních prohlídek.

V textu jsou popsány druhy panoramatických fotografií, které lze využít k tvorbě virtuálních prohlídek. Dále text popisuje některé grafické enginy pro tvorbu aplikací ve virtuální realitě, zařízení pro pořízení 360° panoramat, zařízení pro zobrazení virtuální reality a popis některých existujících aplikací s virtuálními prohlídkami.

Výsledné řešení se skládá ze dvou aplikací. První aplikací je webový editor virtuálních prohlídek, který je řešen jako *single-page* aplikace pomocí frameworku *Vue.js* a knihovny *Three.js*. Serverová část aplikace je implementována pomocí frameworku *Laravel* a poskytuje API pro druhou aplikaci, která slouží k zobrazování vytvořených prohlídek v editoru. Aplikace pro zobrazování virtuálních prohlídek funguje jako klient a je implementována pomocí enginu *Unity*.

Na závěr došlo k porovnání vytvořené prohlídky pomocí editoru v aplikaci pro zobrazování prohlídek a otestování několika uživateli. Bylo zjištěno, že výsledné řešení lze použít jako plnohodnotný nástroj pro vytváření a zobrazování virtuálních prohlídek. Zadání práce bylo tedy splněno. Aplikaci by bylo vhodné do budoucna rozšířit o uživatelské účty a možnost vkládání více objektů do prohlídky.

Literatura

[1] *CryEngine*. [Online; navštíveno 06.01.2019].
URL https://www.cryengine.com/

[2] *Google Cardboard*. [Online; navštíveno 06.01.2019].
URL https://vr.google.com/cardboard/

[3] *Google Street View*. [Online; navštíveno 04.05.2019].
URL https://www.google.com/streetview/

[4] *Krpano Panorama Viewer*. [Online; navštíveno 04.05.2019].
URL https://krpano.com/

[5] *Krpano Panorama Viewer - Kuchlerhaus*. [Online; navštíveno 04.05.2019].
URL https://krpano.com/tours/kuchlerhaus/

[6] *Laravel*. [Online; navštíveno 07.05.2019].
URL https://laravel.com/

[7] *Oculus*. [Online; navštíveno 06.01.2019].
URL https://www.oculus.com/

[8] *Samsung Gear 360*. [Online; navštíveno 06.01.2019].
URL https://www.samsung.com/global/galaxy/gear-360/

[9] *Samsung Gear VR*. [Online; navštíveno 06.01.2019].
URL https://www.samsung.com/global/galaxy/gear-vr/

[10] *Three.js*. [Online; navštíveno 10.05.2019].
URL https://threejs.org/

[11] *Three.js projecting mouse clicks to a 3D scene - how to do it and how it works*. [Online; navštíveno 10.05.2019].
URL http://barkofthebyte.azurewebsites.net/post/2014/05/05/three-js-projecting-mouse-clicks-to-a-3d-scene-how-to-do-it-and-how-it-works

[12] *Unity*. [Online; navštíveno 06.01.2019].
URL https://unity3d.com/

[13] *Unity Manual*. [Online; navštíveno 06.01.2019].
URL https://docs.unity3d.com/

[14] *Unreal Engine*. [Online; navštíveno 06.01.2019].
URL https://www.unrealengine.com/

[15] *Virtual Travel*. [Online; navštíveno 04.05.2019].
URL https://www.virtualtravel.cz/

[16] *Virtual Travel - Hrad Špilberk*. [Online; navštíveno 04.05.2019].
URL https://www.virtualtravel.cz/brno/hrad-spilberk

[17] *Vue.js*. [Online; navštíveno 19.05.2019].
URL https://vuejs.org/

[18] *Vuex*. [Online; navštíveno 19.05.2019].
URL https://vuex.vuejs.org/

[19] Brosz, J.; Samavati, F.: Shape Defined Panoramas. In *2010 Shape Modeling International Conference*, Červen 2010, s. 57–67.

[20] Brown, M.; Lowe, D. G.: Automatic Panoramic Image Stitching using Invariant Features. *International Journal of Computer Vision*, ročník 74, č. 1, 2007: s. 59–73, ISSN 0920-5691.

[21] Dirksen, J.: *Learning Three.js: The JavaScript 3D Library for WebGL*. Packt Publishing, 2013, ISBN 9781782166283.

[22] Ionescu, D.: *Google Street View Gets Smart Navigation*. [Online; navštíveno 03.05.2019].
URL https://www.pcworld.com/article/166178/google.html

[23] Leff, A.; Rayfield, J. T.: Web-application development using the Model/View/Controller design pattern. In *IEEE Enterprise Distributed Object Computing Conference*, Září 2001, s. 118–127.

[24] Salomon, D.: *Transformations and Projections in Computer Graphics*. Springer, 2006, ISBN 978-1846283925.

[25] Szeliski, R.: Image Alignment and Stitching: A Tutorial. *Foundations and Trends in Computer Graphics and Vision*, ročník 2, č. 1, 2006: s. 1–104, ISSN 1572-2759.