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

WEBOVÝ RENDERER LIGHT FIELD DAT

WEB-BASED RENDERER OF LIGHT FIELD DATA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MATĚJ HLÁVKA**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. TOMÁŠ CHLUBNA**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Hlávka Matěj**
Program: Informační technologie
Název: **Webový renderer light field dat**
Web-Based Renderer of Light Field Data
Kategorie: Počítačová grafika

Zadání:

1. Nastudujte možnosti zobrazování 3D scén ve webovém prohlížeči
2. Seznamte se s problematikou light field renderingu
3. Navrhněte webovou aplikaci včetně uživatelského rozhraní pro rendering light fieldových dat
4. Aplikaci implementujte a demonstrujte její použití na různých typech scén
5. Zdokumentujte a zveřejněte výslednou aplikaci
6. Vytvořte video reprezentující výsledky vaší práce

Literatura:

- Levoy, Marc, and Pat Hanrahan. "Light field rendering." *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. 1996.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3, experimenty vedoucí k bodu 4.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Chlubna Tomáš, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 30. října 2020

Abstrakt

Tato bakalářská práce se zabývá generováním light fieldů z 3D scén. Cílem je navrhnout a implementovat webovou aplikaci, která dokáže vygenerovat light field ve formě diskrétní množiny obrázků z uživatelem poskytnuté 3D scény. Práce zahrnuje zpracování souborů poskytnutých uživatelem, vykreslení scény, manipulaci s kamerou a osvětlením a generování light fieldu podle zadaných parametrů.

Abstract

This bachelor thesis deals with generating of light fields from 3D scenes. The goal is to design and implement a web application, that will be capable of generating a light field in a form of discrete set of images based on the user supplied 3D scene. The work includes parsing files supplied by the user, rendering the scene, camera and light manipulation and light field generation according to entered parameters.

Klíčová slova

3D, light field, plenoptický, WebGL, JavaScript

Keywords

3D, light field, plenoptic, WebGL, JavaScript

Citace

HLÁVKA, Matěj. *Webový renderer light field dat.* Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Chlubna

Webový renderer light field dat

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Tomáše Chlubny. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Matěj Hlávka
5. května 2021

Poděkování

Chtěl bych poděkovat svému vedoucímu, panu Ing. Tomáši Chlubnovi, za jeho vůli a ochotu konzultovat se mnou všechny problémy, na které jsem při vypracovávání narazil. Dále bych rád poděkoval své přítelkyni za její péči, lásku a výborné obědy. Velké díky také patří mým rodičům za poskytnuté zázemí a podporu. Do poděkování bych rád zahrnul i své přátele a bratra, kteří mi pomohli překonat koronakrizi příjemnými večery nad skleničkou dobré whiskey a vychlazeného piva. V neposlední řadě bych rád poděkoval i vývojářům id Software za skvělé hry, díky kterým jsem se dostal do světa IT (za toto vděčím i svému bratrovi a tatínkovi) a také za vydání posledního DLC do hry Doom Eternal necelé dva měsíce před odevzdáváním této práce, díky čemuž jsem alespoň na pár dní dostal řádný důvod k prokrastinaci.

Obsah

Kapitola 1

Úvod

Světelné paprsky jsou jedním z nejzákladnějších fyzikálních jevů našeho světa. Umožňují nám vnímat prostředí okolo nás. Na rozdíl od obyčejných obrázků, které zachycují 2D projekci světelných paprsků, light field popisuje distribuci světelných paprsků ve volném prostoru. V závislosti na použité technologii pro zachycení light fieldu je light field většinou tvořen mřížkou obrázků zachycených z různých úhlů pohledu. Díky této vlastnosti lze modifikovat části výsledného light fieldu i po jeho pořízení; je možné např. změnit zaostření, hloubku ostrosti nebo dokonce vytvořit 3D obraz. První model, který popisuje distribuci paprsků, byl definován Gershunem v roce 1936 [7] a dále rozvinut Adelsonem a Bergenem [3]. Tento model je známý pod názvem plenoptická funkce a popisuje světlo ve scéně jako funkci pozice, úhlu, vlnové délky a času.

První zařízení schopné snímat light fieldy bylo vynalezeno již v roce 1908, i přesto ale našly light fieldy své využití až ke konci 20. století. Asi nejznámější využití light fieldu pochází z jedné z nejslavnějších scén kinematografie, známé jako „bullet time“, z filmu Matrix, která byla natočena pomocí kolekce kamer uspořádaných okolo herce, jak lze vidět na obrázku 1.1.

Obrázek 1.1: Natáčení slavné scény, známé jako „bullet time“, z filmu Matrix. Okolo herce jsou rozmístěny kamery, které umožňují snímat ze scény light field[1].

Cílem této práce je navrhnout a implementovat aplikaci, která dokáže vygenerovat light field z uživatelem poskytnuté 3D scény. Vstupem celé aplikace jsou soubory, obsahující informace o geometrii 3D scény. Tyto soubory jsou zpracovány a je z nich vykreslena scéna.

[1] Převzato z http://raytracey.blogspot.com/2017/05/practical-light-field-rendering.html

Uživateli je poté umožněno nastavit parametry light fieldu a vytvořit ze scény light field ve formě archivu obsahující diskrétní množinu obrázků.

V kapitole 2 je popsán light field, jeho vizualizace, vykreslování a jednotlivé způsoby, jak lze light fieldy vytvářet. Dále jsou zde uvedeny příklady využití light fieldů a technologie využité pro vývoj výsledné aplikace. Následně je v kapitole 3 rozebrán postup při návrhu řešení a architektura aplikace. Nakonec je v kapitole 4 popsána implementace aplikace a v kapitole 5 výsledky měření finální aplikace.

Kapitola 2

Teorie

V této kapitole jsou uvedeny minimální nutné informace k porozumění problematiky práce. Jedná se o teoretické výňatky týkající se light fieldů, jejich tvorby, renderingu a vizualizace.

2.1 Light field

Light field je definován jako kolekce světelných paprsků proudících 3D prostorem ve všech možných směrech skrze každý bod. Prostor všech možných paprsků je dán multidimenzionální plenoptickou funkcí [3]. K získání takové funkce je zapotřebí zachytit světelné paprsky v každém možném místě (x, y, z), z každého možného úhlu (θ, ϕ), na každé vlnové délce γ ve všech časech t. Takováto plenoptická funkce je tedy sedmi rozměrná funkce zapsána jako $L(x, y, z, \theta, \phi, \gamma, t)$.

Nicméně takto vícerozměrná data je těžké v praxi zpracovávat. Proto byl light field model dvakrát zjednodušen pro své praktické využití. V prvním zjednodušení je předpokládáno, že měřená funkce je monochromatická a nezávislá na čase. Vlnová délka γ každého světelného paprsku je zaznamenána nezávisle v různých barevných kanálech. Časová sekvence t může být u dynamického light fieldu (tj. light field videa) zaznamenána v různých snímcích. Takto lze z plenoptické funkce odstranit dimenze vlnové délky γ a času t, čímž se model zredukuje ze sedmi dimenzí na pět.

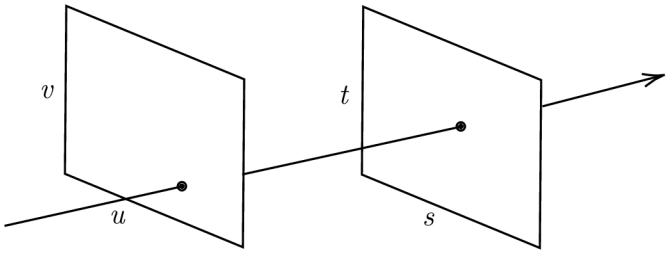
Při parametrizaci light fieldu vznikají tři základní problémy [11]:

- Výpočetní výkonnost
- Kontrola nad kolekcí paprsků
- Jednotné vzorkování

Na základě těchto problémů je nejběžnější způsob reprezentace light fieldů popsat světelné paprsky pomocí souřadnic průsečíků s dvěma plochami umístěnými na libovolných pozicích. Jelikož se ve druhém zjednodušení light field modelu předpokládá, že byl light field zachycen ve volném prostoru (tj. v prostoru bez jakýchkoliv překážek), je možné v tomto způsobu reprezentace pominout zářivost paprsků, protože se po jejich trase nemění, a díky tomu zredukovat model z pěti dimenzí na čtyři.

Systém souřadnic 4D light fieldu je označen (u, v) pro první plochu a (s, t) pro druhou plochu. Orientovaný paprsek světla nejdříve protne plochu uv na souřadnici (u, v) a poté plochu st na souřadnici (s, t) a je tedy zapsán jako $L(u, v, s, t)$. Light field popsaný takovýmto způsobem se nazývá také **light slab**.

Vizualizaci **light slabu** lze vidět na obrázku 2.1. Prostor, kterým je light field reprezentován, poté už není považován za volný, ale je omezen plochami uv a st. Takto je plenoptická funkce popisující light field zredukována ze sedmi dimenzí na čtyři a parametrizována čtyřmi souřadnicemi (u, v, s, t).



Obrázek 2.1: Vizualizace **light slabu**. Orientovaný paprsek světla nejdříve protne plochu uv a poté plochu st.

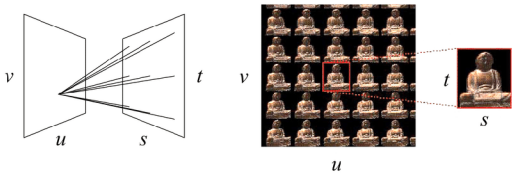
2.2 Vizualizace light fieldů

V light field modelu s dvěma plochami je možné na plochu uv namapovat souřadnice kamer, jejichž ohnisková plocha se nachází na ploše st. Jedná se o kamery, které jsou použity pro zachycení light fieldu. Existují dva různé pohledy pro pochopení tohoto modelu. První předpokládá, že každá kamera zachycuje světelné paprsky vycházející z plochy st dopadající na určitý bod na ploše uv (kolekce světelných paprsků z určitého úhlu pohledu). Takto může být 4D light field reprezentován jako mřížka obrázků, demonstrováno na obrázku 2.2a. Každý z těchto obrázků pořízený kamerou se nazývá **sub-aperture image** nebo také **pinhole view**.

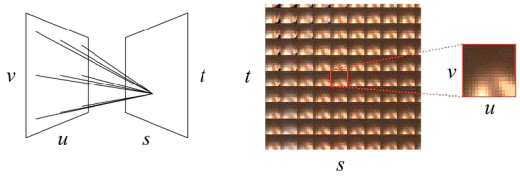
Druhý pohled předpokládá, že určitý bod na ploše st reprezentuje světelné paprsky směřující ke všem bodům na ploše uv (kolekce světelných paprsků z různých úhlů pohledu promítnutých na určitý bod, tj. stejný bod viděn z různých úhlů pohledu, viz obrázek 2.2b). Protože počet vzorků na ploše uv závisí na počtu úhlů pohledu, zatímco počet vzorků na ploše st závisí na rozlišení kamery, jsou dimenze u a v označovány jako úhlové dimenze a dimenze s a t jsou označovány jako prostorové dimenze.

Z 4D light fieldu $L(u, v, s, t)$ lze shromážděním vzorků na fixních souřadnicích u^* a v^* plochy uv získat 2D část $I_{u^*,v^*}(s, t)$. Tuto část lze považovat za obrázek pořízený kamerou (**sub-aperture image**) na pozici (u^*, v^*). Obdobně lze shromážděním vzorků na fixních souřadnicích s^* a t^* plochy st získat část $I_{s^*,t^*}(u, v)$. Tato část (často označována jako **light field subview**) je vytvořena seskupením paprsků v každém bodě z různých úhlů pohledu.

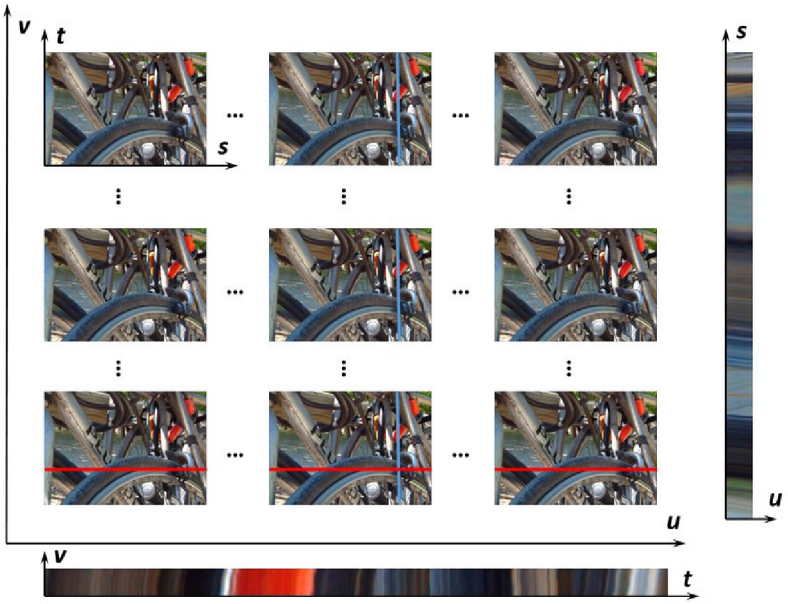
Výše zmíněné 2D části jsou získány shromážděním vzorků buď dvou prostorových dimenzí nebo dvou úhlových dimenzí. Shromážděním light field vzorků s fixní úhlovou souřadnicí v (resp. u) a prostorovou souřadnicí t (resp. s) lze získat část $E_{v^*,t^*}(u, s)$ (resp. $E_{u^*,s^*}(v, t)$). Tento výsledek se nazývá **epipolar-plane image** (dále jen EPI) [5]. Na rozdíl od **sub-aperture image** nebo **light field subview**, EPI obsahuje informace o prostorové i úhlové dimenzi, jak lze vidět na obrázku 2.2c.



(a) Každý obrázek v mřížce reprezentuje světelné paprsky dopadající na jeden bod na ploše uv ze všech bodů na ploše st. Vyznačený obrázek se nazývá **sub-aperture image** nebo **pinhole view**. Převzato z [11].



(b) Každý obrázek reprezentuje světelné paprsky vycházející z jednoho bodu na ploše st směřující ke všem bodům na ploše uv. Vyznačený obrázek bývá označován jako **light field subview**. Převzato z [11].



(c) **Epipolar-plane image** (EPI) je vytvořen zafixováním souřadnic v prostorové i úhlové dimenzi. Dole lze vidět EPI $E_{u^*,s^*}(v,t)$ s fixními souřadnicemi u a s, po pravé straně pak $E_{v^*,t^*}(u,s)$ s fixními souřadnicemi v a t. Převzato z [12].

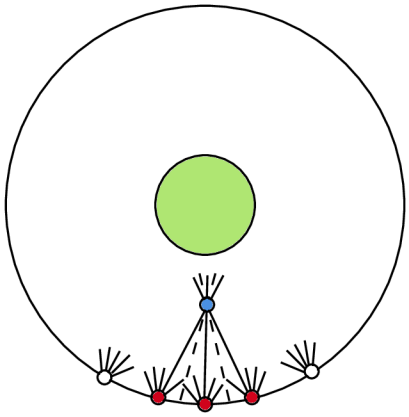
Obrázek 2.2: Různé způsoby vizualizace light fieldu.

2.3 Light field rendering

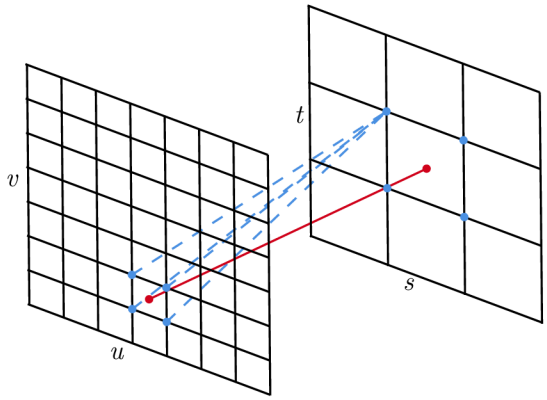
4D light field $L(u,v,s,t)$ lze považovat za kolekci pohledů zachycených kamerami. Tyto kamery jsou reprezentovány dvěma paralelními plochami st a uv. V praxi mohou být ale tyto plochy také kulové. V porovnání s rovinnými plochami, kulové plochy vzorkují světelné paprsky více jednotně a poskytují pevnou vzdálenost k objektu.

Převzorkováním a interpolací světelných paprsků lze s dostatečným počtem kamer vytvořit virtuální pohled na jakékoliv pozici povrchu koule nebo i blíže k objektu, viz obrázek 2.3a. Jelikož zářivost paprsků zůstává ve volném prostoru stále stejná, některé světelné paprsky mohou být převzorkovány z již existujících pohledů. Nicméně stále zůstávají světelné paprsky, které žádné z kamer nezachycují a tudíž musí být odvozeny interpolací.

Pro vykreslení těchto paprsků jsou nejdříve vypočítány jejich průsečíky s oběma plochami st a uv. Poté se použije 16 nejbližších zachycených paprsků pro interpolaci virtuálního paprsku. Tento postup je ilustrován na obrázku 2.3b a označuje se jako light field rendering [11].



(a) Ilustrace light field vzorkovacího systému s kulovými plochami. Objekt (zelený kruh) se nachází v centru koule, po jejímž povrchu jsou rozprostřeny kamery. Modrá tečka představuje virtuální pohled umístěný blíže k objektu. V tomto pohledu mohou být některé paprsky převzorkovány z existujících pohledů (plné čáry), nicméně některé paprsky nejsou zachyceny kamerami (čárkované čáry).



(b) Pro interpolaci virtuálního paprsku je použito nejbližších 16 zachycených paprsků (zde jsou pro ilustrační účely znázorněny pouze 4).

Obrázek 2.3: Vytvoření virtuálního pohledu za pomoci zachycené diskrétní reprezentace light fieldu.

Nedostatečný počet vzorků způsobí u nově vytvořených pohledů ghosting effect (způsobuje, že některé objekty na výsledném pohledu budou částečně průhledné). Nicméně při příliš vysokém počtu vzorků může výsledný light field dosáhnout enormních rozměrů. Pro vykreslování nových pohledů bez ghostingu je zapotřebí, aby od sebe sousedící pohledy nebyly vzdáleny více než 1 pixel (hodnota úzce spojená s rozlišením kamer a hloubkou scény). Čím blíže k sobě sousedící pohledy jsou, tím ostřejší budou interpolované body. Ghosting lze také částečně odstranit, pokud je pro interpolaci virtuálních paprsků použita navíc informace o hloubce či geometrii scény [8].

2.4 Tvorba light fieldů

Tato sekce se zaměřuje na popis existujících zařízení či metod pro tvorbu a pořizování light fieldů.

2.4.1 Plenoptické kamery

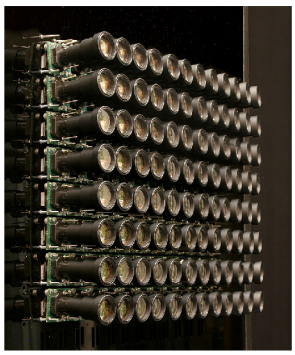
Plenoptické kamery umožňují snímání dynamického light fieldu s jediným obrazovým senzorem avšak za cenu sníženého úhlového nebo prostorového rozlišení. V roce 1992 byla popsána kamera, která má před senzorem umístěnou hlavní čočku a mřížku čoček [2]. Toto

zařízení bylo schopné snímat light fieldy s úhlovým rozlišením 5×5, kde každý obrázek měl 100×100 pixelů. Poté byla vytvořena příruční light field kamera vložením mřížky čoček o velikosti 296×296 mezi senzor a hlavní čočku [14]. Tato kamera vypadala stejně jako obyčejná kamera a byla schopná snímat light fieldy s úhlovým rozlišením 14×14. V současné době jsou plenoptické kamery komerčně dostupné od společností Raytrix a Lytro; jednu z těchto kamer lze vidět na obrázku 2.4.

Obrázek 2.4: Lytro Illum, plenoptická kamera, která je komerčně dostupná[1].

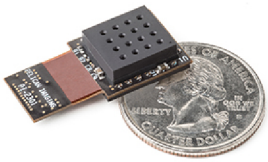
2.4.2 Mřížky kamer

Tento přístup vyžaduje rozprostření kamer do mřížky na kulové či rovinné ploše pro souběžné snímání light field vzorků z různých úhlů pohledu. Prostorové dimenze (s a t) light fieldu jsou dány senzory, zatímco úhlové dimenze (u a v) jsou dány počtem kamer a jejich rozmístěním. 4D light field je tvořen kombinací nasnímaných obrázků.



(a) Systém kamer 8×12. Převzato z [18].

(b) Systém kamer 6×8 s možností dynamického přemisťování kamer při snímání light fieldů. Převzato z [21].



(c) PiCam; ultra tenká mřížka kamer 4×4, kterou lze připojit do mobilního telefonu. Převzato z [16].

Obrázek 2.5: Různé interpretace mřížek kamer pro snímání light fieldů.

Statická mřížka kamer

V roce 2002 byla navrhnuta mřížka kamer o velikosti 8×8 pro snímání dynamického light fieldu [20]. Ve stejném roce bylo použito 6 CMOS obrazových senzorů pro nahrání

[1] Převzato z https://www.ephotozine.com/article/lytro-illum-v2-professional-light-field-camera-review-26434

synchronizované video datové sady [19]. Tento systém byl dále rozvinut na 125 kamer a byl použit pro snímání videí, jejichž počet snímku za sekundu se pohyboval v řádech tisíců (jedna z konfigurací tohoto systému je zobrazena na obrázku 2.5a) [18].

Mřížka s pozicovatelnými kamerami

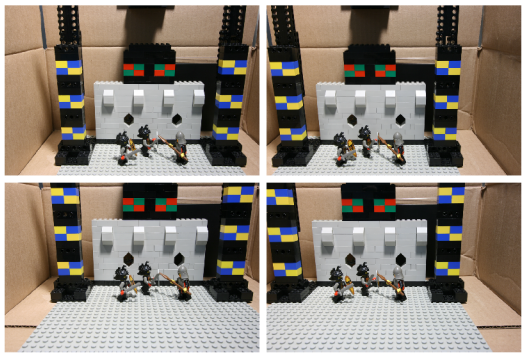
Zhang a Chen [21] vytvořili systém sestávající z mřížky kamer o velikosti 6×8 s možností měnit pozice kamer během procesu snímání. Tento systém lze vidět na obrázku 2.5b. Zároveň představili algoritmus, který automaticky posunuje kamery na základě kvality syntetizovaného virtuálního pohledu.

Příruční mřížka kamer

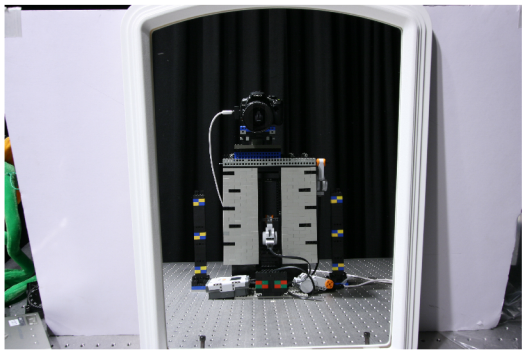
V roce 2013 byla představena ultra tenká monolitická mřížka kamer, která se jmenuje PiCam (Pelican Imaging Camera-Array, zobrazena na obrázku 2.5c) [16]. PiCam je mřížka kamer 4×4. Každá z těchto kamer je schopna snímat 1000×750 pixelů. Celé zařízení je menší než mince a je možné jej připojit do mobilního telefonu.

2.4.3 Veřejné datové sady

Některé pořízené light fieldy byly zveřejněny pro volné použití. Mezi nejznámější patří The (New) Stanford light field archive obsahující 13 úhlových a prostorových light fieldů ve vysokém rozlišení zachycené pomocí konstrukce sestavené z Lega, kterou lze vidět na obrázku 2.6b, dále pak 7 mikroskopických light fieldů, 4 light fieldy získané pomocí počítačem ovládané konstrukce a 2 light fieldy získané pomocí mřížky kamer [1]. Jeden z těchto light fieldů lze vidět na obrázku 2.6a.



(a) Light field lego scény obsahující poměrně jednoduché geometrické objekty v různých hloubkách. Pro ilustraci jsou zde zobrazeny pouze rohové obrázky.



(b) Lego konstrukce, která byla použita pro zachycení části datové sady. Součástí sady je i light field samotné konstrukce, pro jehož zachycení bylo použito zrcadlo.

Obrázek 2.6: Ukázky z veřejné datové sady z The (New) Stanford light field archive. Převzato z [1].

2.5 Využití light fieldů

Když byl light field poprvé představen, bylo předpokládáno využití pouze pro vytváření nových pohledů. Nicméně s vývojem výpočetních systémů začala popularita light fieldů

rychle stoupat, zejména pak v odvětví počítačového vidění. Zde jsou uvedeny nejvýznamnější využití light fieldů.

2.5.1 Změna zaostřovací roviny

Díky light fieldu lze s dostatečně velkými clonami vidět skrze překážky [10]. Jelikož pozorovaný objekt a překážka jsou v rozdílných hloubkách, každý pohled je schopen zachytit malou část pozorovaného objektu. Vybráním požadované části z každého pohledu lze syntetizovat nový obrázek v určité hloubce, jak lze vidět na obrázcích 2.7.

(a) Původní obrázek z light fieldu. Pevnina v pozadí je překryta stromem a nelze ji tak vidět celou.

(b) Nový obrázek syntetizovaný z původních obrázků light fieldu.

Obrázek 2.7: S dostatečně velkou clonou lze vytvořit nový pohled, ve kterém lze vidět přes překážky. Převzato z [10].

2.5.2 Detekce, klasifikace a rozpoznávání

Díky schopnosti rozeznat a manipulovat s překážkami ve scéně mají light fieldy výborné předpoklady pro řešení problémů detekce a rozpoznávání. Tato vlastnost je užitečná zejména pro detekci význačných bodů (rozeznávání pozadí a popředí). Oproti detekci význačných bodů u obyčejných obrázků je tato metoda schopna zvládat náročnější případy jako je např. scéna s podobným pozadím a popředím.

Light fieldy lze využít i pro rozpoznávání materiálů. Oproti rozpoznávání objektů se ale jedná o poměrně složitý problém z důvodu rozmanitosti vzhledu materiálů. Díky rozvinutým technikám strojového učení byla vyvinuta konvoluční neuronová síť, která využívá 4D light field datové sady pro rozpoznávání materiálů [17]. V porovnání s rozpoznáváním materiálů u obyčejných obrázků je tato metoda přesnější přibližně až o 6–7 %.

Za pomoci light fieldu lze ve scéně detekovat transparentní objekty. Průhledné objekty deformují pozadí kvůli refrakci a tím způsobují pokřivení paprsků ve 4D light fieldu. Tento jev se nazývá „funkce pokřivení light fieldu“ a používá se pro rozpoznání transparentních objektů [13].

2.6 Editace light fieldů

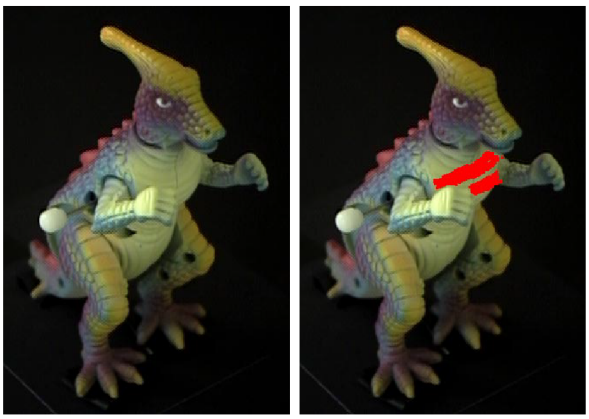
Editace light fieldů se často považuje za zobecnění klasických úprav 2D obrázků. Nicméně toto zobecnění nelze považovat za jednoduché, jelikož oproti obyčejným 2D obrázkům obsahují light fieldy více informací. Takovéto editace lze rozdělit do dvou kategorií:

1. Lokální editace – jedná se o úpravy aplikované pouze na určité části light fieldu.
2. Globální editace – tyto úpravy modifikují celý light field.

2.6.1 Lokální editace

Jedna z prvních metod pro úpravu light fieldů byla založena na voxelové rekonstrukci geometrie zachycené v light fieldu [15]. Takováto reprezentace umožňuje provádět úpravy jako je např. kreslení či ořezávání. Nicméně tyto úpravy velmi závisí na kvalitě voxelizace. Příklad editace light fieldu pomocí této metody lze vidět na obrázcích 2.8.

(a) Vpravo lze vidět původní obrázek hračky dinosaura, vlevo pak stejný obrázek pokreslený dvěma červenými čarami.



(b) Nalevo lze vidět obrázek hračky dinosaura pořízený z jiného úhlu pohledu, napravo je pak obrázek vygenerovaný pomocí propagace kreslení na obrázek vlevo.

Obrázek 2.8: Ukázka editace light fieldu pomocí voxelové rekonstrukce geometrie. Převzato z [15].

Rozdělením light fieldu na více sub-light fieldů lze nezávisle deformovat každý ze sub-light fieldů a poté je vykreslit dohromady [6]. Takovýmto způsobem lze deformovat původní light field. Pro zachování konzistence osvětlení po deformaci je tento systém limitovaný pouze na koaxiální osvětlení. Díky metodě nelineární editace v prostorové dimenzi light fieldu lze v prostorové dimenzi provádět komplexnější deformace a zároveň zachovat konzistenci v úhlové dimenzi a vlastnosti light fieldu jako např. přeostření [4].

2.6.2 Globální editace

Globální editace jsou jednou z nejrozšířenějších forem úpravy light fieldů a byly zahrnuty v některých komerčních softwarech, kde byly obvykle založené na aplikaci předdefinovaných filtrů. Pomocí paprskového shader jazyka lze napsat programy, které umožňují manipulaci s light fieldy, skládání light fieldů nebo komplexní pokřivení paprsků [9]. Využitím metody založené na obrázcích lze vložit reálné a syntetické objekty do jedné scény za pomoci dvou 4D light fieldů; jedním zachyceným z dané scény a druhým promítnutým do dané scény.

2.7 Použité technologie

V této sekci jsou uvedeny jazyky, technologie a knihovny, které byly využity pro vývoj cílové aplikace.

2.7.1 WebGL

WebGL je multiplatformní API sloužící pro tvorbu 3D grafiky ve webovém prohlížeči. Je založeno na OpenGL a používá stejný shader jazyk - GLSL. Jelikož se jedná o DOM API, může ho používat jakýkoliv DOM-kompatibilní jazyk, avšak nejčastěji je používáno v kombinaci s jazykem JavaScript. WebGL je pouze nízkoúrovňové API, proto musí vývojáři psát vlastní shadery, předávat do nich proměnné a také obstarávat maticové operace pro aplikaci transformací.

2.7.2 GLSL

Pro tvorbu shaderů aplikace využívá jazyk GLSL. Tento jazyk vychází z jazyka C a obsahuje funkce zaměřené na manipulaci s vektory a maticemi. Shadery napsané v tomto jazyce lze rozdělit do dvou kategorií:

1. Vertex shader – získává vstup ve formě atributů, uniformních proměnných či textur. Při každém volání tohoto shaderu je nutné nastavit výstupní proměnnou `gl_Position` na požadované souřadnice daného vrcholu. Dále je v něm pak také možné předávat proměnné fragment shaderu.

2. Fragment shader – obdobně jako vertex shader přijímá vstup ve formě uniformních proměnných nebo textur, namísto atributů je ale schopen přijímat proměnné přímo od vertex shaderu. Jeho výstupem je proměnná `gl_FragColor` určující barvu daného pixelu.

Oba druhy shaderů jsou samostatnou součástí vykreslovacího řetězce grafické karty.

2.7.3 Wavefront OBJ formát

Jedná se o formát souboru popisující geometrii 3D objektů[2]; mezi nejdůležitější položky z hlediska aplikace patří:

- Vrcholy geometrie – jedná se o záznamy specifikující souřadnice ve tvaru $(x, y, z, [w])$. Navíc může obsahovat i informace o barvách; v takovém případě bude záznam obsahovat za souřadnicemi navíc i hodnoty jednotlivých složek modelu RGB.
- Normálové vektory – ovlivňují stínování a vykreslování geometrie. Záznam obsahuje souřadnice ve tvaru (x, y, z).
- Souřadnice textur – specifikují souřadnice ve tvaru $(u, [v], [w])$. Podle počtu souřadnic se může jednat o 1D, 2D nebo 3D texturu.
- Stěny – jsou definovány indexy vrcholů geometrie, normálových vektorů a souřadnic textur v tomto pořadí. Dohromady vytvářejí polygony.

Ukázku souboru ve Wavefront OBJ formátu lze vidět na výpisu 2.1. Mnoho modelovacích programů umožňuje exportovat vytvořené scény ve Wavefront OBJ formátu. Mezi tyto programy patří např. Blender nebo SketchUp.

```
# List of geometry vertices:
v 0.0 0.0 0.0 # Bottom left.
v 5.0 0.0 0.0 # Bottom right.
v 5.0 5.0 0.0 # Top right.
v 0.0 5.0 0.0 # Top left.

# List of faces:
f 1 2 3 4 # Face of a square. Only geometry vertex indices are present.
```

Výpis 2.1: Zápis tvaru čtverce ve Wavefront OBJ formátu. V souboru jsou zaznamenány pouze vrcholy geometrie (řádky začínající písmenem `v`). Z toho důvodu také obsahuje stěna čtverce pouze indexy vrcholů geometrie.

MTL soubory

Tento typ souboru obsahuje informace o materiálech použitých v přidruženém OBJ souboru. Příklad obsahu takového souboru lze vidět na výpisu 2.2. Nejpodstatnější položky souboru jsou následující:

- Ambientní, difúzní, spekulární a emisivní odrazivost – všechny tyto položky specifikují barvu vektorem tří hodnot RGB modelu.
- Spekulární exponent – jedná se o číslo, kterým bude umocněno spekulární zvýraznění světla.
- Průhlednost – udává alfa složku výsledné barvy objektu.
- Názvy zdrojových souborů textur

[2]Podrobnější dokumentace Wavefront OBJ formátu: http://paulbourke.net/dataformats/obj/

```
newmtl initialShadingGroup # Material name
Ns 323.9 # Specular exponent
Ka 1.0 1.0 1.0 # Ambient reflectivity
Kd 0.7 0.5 0.6 # Diffuse reflectivity
Ks 0.5 0.5 0.5 # Specular reflectivity
Ke 0.0 0.0 0.0 # Emissive reflectivity
d 1.0 # Transparency
map_Kd cube_diffuse.jpg # Texture linked to diffuse reflectivity
```

Výpis 2.2: Zápis materiálu ve formátu MTL.

2.7.4 Pomocné knihovny

Ke zjednodušení implementace aplikace využívá několik podpůrných knihoven zajišťující zejména maticové operace a manipulaci se shadery.

Tiny WebGL Helper Library

Tato knihovna usnadňuje práci se shadery pomocí rozhraní poskytující funkce pro nahrávání, kompilaci a linkování shaderů, dále pak pro předávání atributů a proměnných do shaderů ve formátu JSON, nahrávání textur a vykreslení poskytnutých dat. Vývojáři tak nemusí předávat do shaderů každý atribut či proměnnou zvlášť, ale stačí pouze vytvořit JavaScriptový objekt s požadovanými hodnotami a poté jej předat příslušné funkci.

glMatrix

Obsahuje funkce pro základní maticové operace jako je např. translace, rotace či násobení.

Webpack

Jelikož aplikace importuje knihovny jako moduly a implementované třídy jsou také realizované jako moduly, je nutné použít balíčkovač (module bundler), který veškeré zdrojové soubory (včetně kaskádových stylů, které zpracovává pomocí css-loader) zkompiluje do výstupních JavaScript souborů, na které se odkazuje HTML stránka.

Light-server

Pro nasazení na server byla zvolena knihovna light-server, jelikož je jednoduchá na použití, umožňuje automatickou obnovu stránky po změně sledovaných souborů a je proto vhodná i pro vývoj a testování.

Bootstrap

Tvorba uživatelského rozhraní byla usnadněna použitím CSS frameworku Bootstrap. Framework značně napomáhá zejména při rozmístění vstupních prvků a jejich stylizací.

JSZip a file-saver

Tyto knihovny jsou použity pro vytvoření archivu s obrázky light fieldu a jeho následné uložení. Aplikace využívá následující funkce:

- `file()` – přidá nový soubor k vybranému archivu.
- `generateAsync()` – vygeneruje výsledný archiv, který lze následně uložit.
- `saveAs()` – jako parametr přijímá soubor, který uloží na straně klienta. Používá se po zavolání funkce `generateAsync()` pro uložení vygenerovaného archivu.

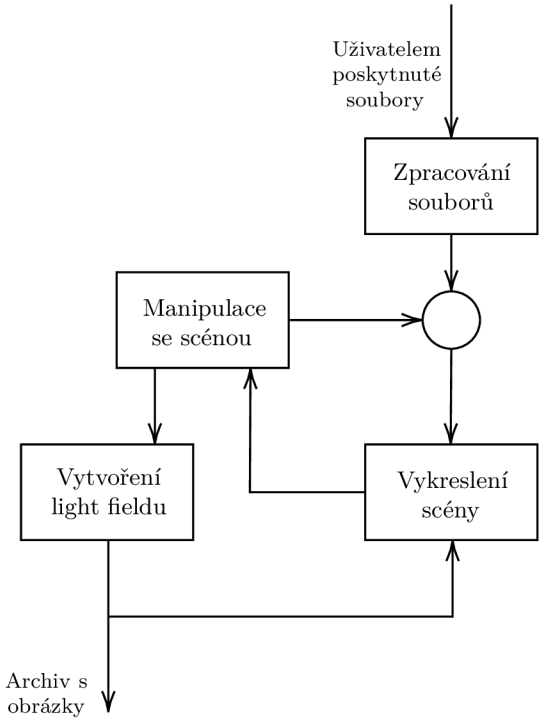
Kapitola 3

Návrh

Tato kapitola se zabývá návrhem řešení. Popisuje architekturu aplikace a interakci s uživatelem.

3.1 Architektura

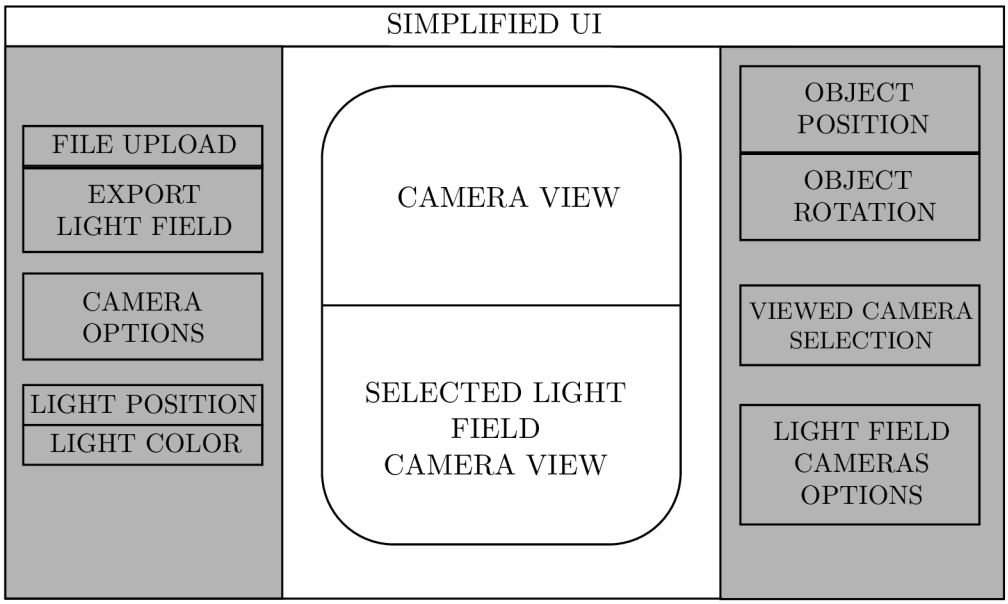
Celý proces tvorby light fieldu lze rozdělit do několika částí. Nejdříve je nutné zpracovat soubory nahrané uživatelem, ze kterých je následně vykreslena scéna. Poté je uživateli umožněno pracovat se scénou a vytvořit light field dle zadaných parametrů. Na obrázku 3.1 lze vidět propojení jednotlivých částí.



Obrázek 3.1: Diagram popisující chod aplikace. Vstupem celé aplikace jsou soubory nahrané uživatelem, ze kterých je vykreslena scéna. Uživateli je dále umožněno manipulovat s objekty scény a exportovat light field ve formě archivu s obrázky. Po vytvoření light fieldu je scéna opětovně vykreslena.

3.2 Uživatelské rozhraní

Klíčovou vlastností uživatelského rozhraní je možnost prohlédnout si scénu z různých úhlů pohledu a poté nasměrovat vykreslovaný objekt a pořídit light field podle přání uživatele. Light field je v aplikaci chápán jako statická mřížka kamer, je tedy pevně umístěn a není možné měnit jeho pozici. Namísto toho je uživateli umožněno přemisťovat a otáčet vykreslovaný objekt. Zároveň je důležité správně vizualizovat light field dle zadaných parametrů uživatele, aby bylo zřejmé, jakou plochu bude light field schopen pokrýt.



Obrázek 3.2: Zjednodušená ilustrace uživatelského rozhraní. Na levém panelu je možnost nahrání souborů a tlačítko pro exportování light fieldu. Zároveň se zde nachází nastavení kamery a světla. Pravý panel slouží pro pohyb s vykreslovaným objektem a nastavení možností light field kamer. Uprostřed se nachází plátno zobrazující scénu z pohledu kamery a vybrané light field kamery.

3.2.1 Rozložení

Jelikož je pro uživatele důležité vidět, jak bude vypadat výsledný light field ještě před jeho pořízením, je plátno pro vykreslení rozděleno na dvě části; v horní části lze vidět vykreslenou scénu z pohledu obyčejné kamery. S touto kamerou lze jednoduše manipulovat pomocí klávesnice a myši. Součástí této scény je i mřížka kamer reprezentující light field. V dolní části se pak nachází pohled z vybrané kamery light fieldu. Tuto kameru lze navolit buď kliknutím na požadovanou kameru na plátně nebo v pravém panelu, kde se společně s tímto výběrem nachází i ostatní prvky rozhraní pro nastavení light fieldu.

Prvky pro manipulaci s kamerou a světlem jsou umístěny v levém panelu. Uživateli je pomocí těchto prvků umožněno měnit pozici a barvu světla. Zároveň má uživatel možnost obnovit pozici kamery pro jednoduchý návrat k vykreslovanému objektu. Navržené rozložení lze vidět na obrázku 3.2.

3.3 Zpracování scény

Pro práci se scénou bylo nejdříve nutné zvolit formát souboru, který bude používán. Pro svou rozšířenost a jednoduchost byl vybrán Wavefront OBJ formát. Zpracování scény lze rozdělit do dvou fází:

1. Načtení scény – provede se potom, co uživatel poskytne potřebné soubory. Zahrnuje zpracování souborů a převedení dat do takového formátu, jaké očekává rozhraní WebGL.

2. Vykreslení scény – po úspěšném zpracování souborů je nutné získaná data předat rozhraní WebGL, které se následně za pomoci připravených shaderů postará o samotné vykreslení scény.

3.3.1 Načtení scény

Nejdůležitější částí načtení scény je zpracování souborů dodaných uživatelem. Největší důraz je kladen na korektní analýzu souboru obsahující geometrii objektů a převedení jeho obsahu do požadovaného formátu.

Zpracování OBJ souboru

Při analýze souboru je důležité zachování indexů jednotlivých druhů vrcholů tak, jak jdou za sebou. Při nesprávné indexaci by ve scéně vznikaly nežádoucí artefakty. Zpracování souboru do formátu, které očekává rozhraní WebGL, lze znázornit algoritmem 1.

Algorithm 1: Zjednodušený algoritmus analýzy OBJ souboru a rozdělení dat do polí, která mohou být předána rozhraní WebGL. V algoritmu je pro jednoduchost znázorněno pouze zpracování nejdůležitějších záznamů.

```
Result: Data structure containing arrays of attributes.
objFile ← user supplied OBJ file;
Initialize arrays for holding coordinates of vertices;
Initialize data structure for holding arrays of attributes;
foreach line in objFile do
    keyword ← getKeyword(line);
    data ← parseData(line);
    if keyword starts with v then
        Register vertex coordinates represented by data in appropriate array;
    else if keyword = f then
        foreach index in data do
            Find coordinates in appropriate array at index and put these
             coordinates into appropriate array in data structure;
        end
    else
        Handle other keywords;
    end
end
```

Zpracování MTL souboru

Analýza MTL souborů je podstatně jednodušší, jelikož soubory obsahují pouze informace o použitých materiálech, které není nutné indexovat. Pro zpracování souborů jsou tyto informace uloženy do objektů rozdělených dle názvu jednotlivých materiálů. Před vykreslením jsou tyto objekty přiřazeny k jednotlivým objektům scény a následně předány rozhraní WebGL. Jelikož uživatel nemusí poskytnout žádný MTL soubor nebo také některé objekty scény nemusí mít žádný materiál přidělený, je nutné takovýmto objektům přiřadit implicitní připravený materiál, aby vykreslení proběhlo v pořádku. Příklad objektu s přiděleným implicitním materiálem a zpracovaným souborem s materiály lze vidět na obrázcích 3.3.

(a) Příklad objektu s využitím implicitních hodnot pro materiály.

(b) Příklad objektu s využitím hodnot zpracovaných ze souboru s materiály. Lze si povšimnout zejména rozdílných barev a také méně výrazné spekulární i difúzní odrazivosti.

Obrázek 3.3: Vizualizace stejného objektu s využitím implicitních hodnot materiálů a s využitím hodnot získaných ze souboru s materiály.

Zpracování textur

Při zpracování textur je nutné dbát na jejich správné namapování na příslušné souřadnice uvedené v OBJ souboru a zároveň na jejich korektní přidělení k odpovídajícím materiálům. Z toho důvodu je zapotřebí pracovat nejen s objekty textur vytvořenými pomocí WebGL, ale i jejich odpovídajícími názvy z původních souborů. Postup vytvoření a přidělení textur k materiálům lze popsat algoritmem 2.

3.3.2 Vykreslení scény

Po zpracování veškerých souborů a vytvoření potřebných datových struktur a objektů je nutné připravit rozhraní WebGL, aplikovat veškeré maticové operace dle uživatelského vstupu a připravit proměnné, které se budou předávat shaderům.

Algorithm 2: Algoritmus popisující načítání textur a jejich přidělení k odpovídajícím materiálům.

```
Input: Array of uploaded images.
Result: Textures binded to corresponding materials.
foreach image in images do
    texture ← createTexture();
    bindTexture(texture);
    imageToTexture(image);
    if isPowerOf2(image.resolution) then
        generateMipmaps();
    else
        setClampToEdge();
        turnOffMipmaps();
    end
    materials.replace(image.name, texture);
end
```

Osvětlovací model

Aplikace využívá Phongův osvětlovací model. Emisivní, ambientní, difúzní i spekulární složky tohoto modelu jsou zjišťovány ze souboru s materiály. Tento model je navíc obohacen o technologii nazývanou normal mapping, která umožňuje přidat detaily do vykreslovaného objektu pomocí modifikace normálových vektorů daného objektu. Pro aplikaci této technologie je nutné, aby byla v souboru s materiály uvedena příslušná textura a zároveň byly v OBJ souboru zadány normálové vektory. Poté mohou být ze souřadnic textur a vrcholů geometrie vypočítány tangenty, které jsou dále využity v shaderech pro modifikaci normálových vektorů. Pokud souřadnice textur nebo normálové vektory chybí, je pro tangenty využita implicitní hodnota.

3.4 Light field

Light field je v aplikaci reprezentován statickou mřížkou kamer. Nejjednodušší způsob, jak jej definovat a manipulovat s ním, je tedy dvourozměrné pole kamer.

3.4.1 Nastavení

Přestože je light field chápán jako statická mřížka kamer, je uživateli umožněno modifikovat vlastnosti celé mřížky i samotných kamer před zahájením tvorby light fieldu. Mezi klíčové vlastnosti patří:

- Počet kamer – určuje výsledný počet obrázků light fieldu. Je možné nastavit celkový počet řádků i sloupců light fieldu.
- Mezery mezi kamerami – uživateli je umožněno editovat vertikální a horizontální mezeru mezi jednotlivými kamerami.
- Rozlišení výsledných snímků – jelikož je velikost jednotlivých obrázků při jejich pořizování závislá na velikosti plátna, je možné v aplikaci navolit rozlišení, kterému se plátno při tvorbě light fieldu automaticky přizpůsobí.

3.4.2 Zvolení kamery

Kameru je možné navolit nejen ze seznamu, ale i přímo kliknutím na požadovanou kameru na plátně. Pro realizaci takovéto vlastnosti je zapotřebí přidělit každé kameře unikátní barvu, podle které bude identifikována. Poté už stačí jen při kliku na plátno zjistit, jakou barvu má pixel pod kurzorem a pokud se barva shoduje s některou z barev kamer, je tato kamera označena jako zvolená. Vybraná kamera je poté pro přehlednost zvýrazněna červenou barvou.

3.4.3 Tvorba light fieldu

Jelikož je light field v aplikaci reprezentován dvourozměrným polem kamer, je pro vygenerování light fieldu nutné přes toto pole iterovat, vykreslit scénu z pohledu aktuální kamery a poté pořídit snímek plátna. Před začátkem iterace je zapotřebí nastavit plátnu rozměry zadané uživatelem. Zároveň je důležité vypnout veškeré vstupy, jelikož při generování light fieldu je veškerá manipulace se scénou nežádoucí a mohla by způsobit inkonzistenci výsledných obrázků. Po úspěšném vytvoření light fieldu jsou všechny vstupy opět povoleny a plátnu jsou nastaveny původní rozměry.

Kapitola 4

Implementace

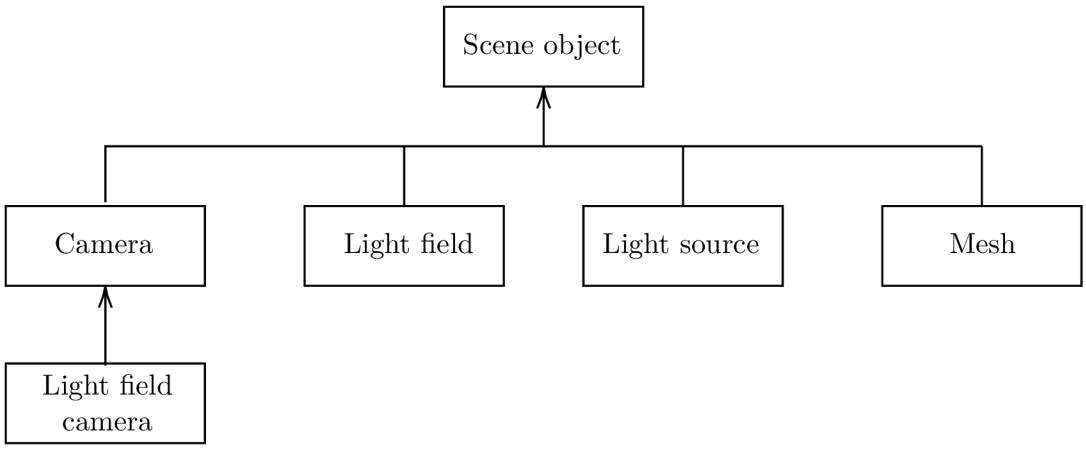
Kapitola zabývající se implementační části aplikace. Je zde popsáno využití WebGL, podpůrných knihoven a GLSL pro realizaci výsledné aplikace pro tvorbu light fieldů.

4.1 Implementace modulů

Tato sekce popisuje jednotlivé moduly, které implementují výslednou aplikaci. Jsou zde popsány nejdůležitější funkce jednotlivých modulů, dále pak vzájemná komunikace a propojení mezi nimi a jejich využití.

4.1.1 Scene object

Tento modul představuje základ pro všechny objekty scény. Obsahuje atributy reprezentující pozici a matici daného objektu a dále pak funkce manipulující s těmito atributy. Na obrázku 4.1 lze vidět jednotlivé objekty scény, které rozšiřují tento modul.



Obrázek 4.1: Třídní diagram znázorňující hierarchii objektů scény. Všechny objekty dědí základní atributy a metody od třídy `Scene object`.

4.1.2 Kamera

Pro zvýšení míry abstrakce a zjednodušení implementace obsahuje aplikace modul reprezentující kameru, který zahrnuje funkce aplikující maticové operace týkající se kamery. Zároveň

je tento modul základem pro kamery light fieldu, které jej obohacují o statickou metodu `getBufferInfo()`, která slouží pro získání atributů potřebných k vykreslení samotného light fieldu.

4.1.3 Light field

Hlavní komponentou tohoto modulu je dvourozměrné pole light field kamer, nad kterým je implementována řada pomocných metod. Nejdůležitější z těchto metod je metoda `initCameras()`, která obstarává inicializaci light fieldu na základě zadaného počtu kamer a mezer mezi nimi.

Velmi užitečná je pak také funkce `iterateCameras()`, která umožňuje jednoduše iterovat přes kamery light fieldu a na každou aplikuje callback, který je poskytnut jako parametr funkce. Pro účely generování light fieldu bylo zároveň nutné tuto funkci implementovat i v asynchronní verzi.

4.1.4 Shadery

Aplikace využívá dohromady 4 shadery. Pro vykreslení celé scény jsou využity komplexnější shadery pracující se světlem, texturami a vlastnostmi materiálů objektů. Tyto shadery jsou podrobněji popsány níže. Light field je vykreslen pomocí jednoduchých shaderů, které pouze pracují s jeho pozicí a aplikují na něj konstantní barvu.

Vertex shader

Hlavní funkce vertex shaderu je poměrně jednoduchá. Proměnná `gl_Position` je vypočítána vynásobením atributu pozice a projekční a pohledovou maticí. Poté už je nutné pouze předat proměnné do fragment shaderu. Normálové vektory jsou vynásobeny s invertovanou transponovanou maticí vykreslovaného objektu a poté normalizovány. Souřadnice textur a informace o barvě jsou jednoduše předány bez nutnosti jejich modifikace.

Dále je zapotřebí sdělit fragment shaderu, jakým způsobem má být odráženo světlo od povrchu vykreslovaného objektu. Toho je docíleno pomocí vektorů `v_surfaceToLight` a `v_surfaceToView`. K jejich výpočtu je nutné zjistit pozici povrchu ve world space, reprezentovanou vektorem `surfaceWorldPosition`, čehož lze dosáhnout vynásobením matice vykreslovaného objektu a atributem pozice. Poté už jen stačí pro `v_surfaceToLight` odečíst pozici světla od `surfaceWorldPosition` a pro `v_surfaceToView` odečíst pozici kamery od `surfaceWorldPosition`.

Fragment shader

Ve fragment shaderu probíhají veškeré výpočty týkající se osvětlovacího modelu, textur a materiálů. Nejdříve je nutné normalizovat vektory `v_surfaceTolight` a `v_surfaceToView` přijaté od vertex shaderu a uložit je do nových proměnných `surfaceToLightDirection` a `surfaceToViewDirection`. Proměnná `light` reprezentující světlo je poté vypočítána jako skalární součin těchto dvou vektorů. Dále je vypočítána proměnná `specularLight` pomocí skalárního součinu normálového vektoru a normalizovaného součtu vektorů `surfaceToLightDirection` a `surfaceToViewDirection`.

Následně už stačí jen zjistit výslednou spekulární a difúzní barvu. Ty jsou vypočítány vynásobením RGB složek příslušného materiálu, textury a zvolené barvy světla. Difúzní barva je navíc vynásobena proměnnou `light` a spekulární barva proměnnou

`specularLight`, kterou je předtím nutné umocnit spekulárním exponentem. Výsledná RGB složka proměnné `gl_FragColor` je pak daná součtem emisivní, ambientní, difúzní a spekulární barvy (emisivní a ambientní barvy jsou pouze zjištěny z materiálů). Alfa složka výsledné barvy je určena průhledností získanou ze souboru s materiály.

4.1.5 Analyzátor OBJ souborů

Tento modul slouží pro zpracování OBJ a MTL souborů a textur. K tomu je využita funkce `parseObj()`, která jako parametry přijímá textový řetězec reprezentující obsah souborů s geometrií a materiály a pole textur. Výstupem funkce je nový `Mesh` objekt, kterému je předáno pole JSON objektů obsahující zpracovanou geometrii a druhé pole, ve kterém jsou zahrnuty informace o materiálech a jejich texturách. Tento `Mesh` objekt je dále využíván v modulu `Renderer`.

Modul je schopen zpracovat základní a nejdůležitější záznamy ze vstupních souborů. Mezi tyto patří:

- Vrcholy geometrie
- Normálové vektory
- 2D textury
- Stěny
- Informace o spekulární, emisivní, ambientní a difúzní odrazivosti materiálů

4.1.6 Mesh

Pro zjednodušení práce s nahranými objekty aplikace implementuje modul obsahující funkce pro manipulaci s geometrií daného objektu. Nejdůležitější z těchto funkcí je funkce `getBufferInfo()`, která vrací pole objektů ve formátu JSON, které obsahují veškeré atributy a informace o materiálech potřebné k vykreslení daného objektu. Další užitečnou funkcí je funkce `generateTangents()`, která vytvoří tangenty na základě dodaných souřadnic textur a vrcholů geometrie.

Dále se zde nachází řada pomocných funkcí pro aplikaci maticových operací na daný objekt. K nejdůležitějším patří funkce `centerOffset()`, která je za pomoci funkce `geometriesExtents()` schopna vycentrovat objekt do středu world space.

4.1.7 Světlo

Světlo je v aplikaci reprezentováno modulem `LightSource`. Obsahuje informace o pozici a barvě světla a dále pak také užitečnou funkci `getScaledColor()`, která vrací vektor reprezentující barvu světla, se kterým lze dále pracovat ve fragment shaderu.

4.1.8 Utils

Modul `Utils` obsahuje řadu statických metod usnadňujících různé operace. Mezi nejdůležitější metody patří:

- `convertRange()` – implementuje převod čísla z intervalu hodnot do jiného intervalu hodnot. Využívá se zejména pro převod barevných složek do podoby hodnot, se kterými pracují shadery.

- `create1PixelTexture()` – vytváří texturu o velikosti jednoho pixelu ze zadaného vektoru reprezentujícího požadovanou barvu. Tato funkce je užitečná při zpracování textur a materiálů pro vytvoření implicitní textury, která bude použita pro objekty, které nemají specifikovanou texturu.
- `arrayEquals()` – jelikož v JavaScriptu nelze kontrolovat shodnost prvků dvou polí pouhým použitím porovnávacího operátoru, je nutné implementovat funkci, která projde veškeré prvky prvního pole a porovná je s prvky pole druhého.
- `createLightFieldProgressBar()` – tato funkce je využita pro vytvoření ukazatele průběhu tvorby light fieldu. Vrací dvojici HTML div elementů `progressBarContainer` a `progressBar`, které jsou dále využity v modulu `Renderer` při generování light fieldu.

4.1.9 Renderer

Jedná se o hlavní modul celé aplikace. Je zde obsažena inicializace a zpracování vstupů, o které se stará funkce `initInputs()` za pomoci dalších pomocných funkcí. Pro jednoduchou manipulaci se vstupy je vytvořen objekt ve formátu JSON, ve kterém jsou obsaženy veškeré vstupy rozdělené dle objektů scény, ke kterým náleží.

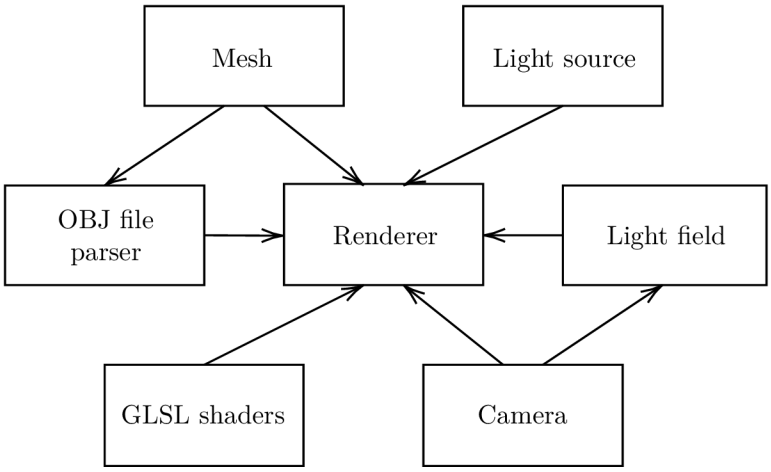
Modul pak dále také obstarává samotné vykreslení scény za pomoci funkce `render()`, kterou lze popsat algoritmem 3. V této funkci se také nachází zpracování vstupu z klávesnice, které je nutné provádět až při vykreslování pro ošetření stisku více kláves naráz. Funkce je při načtení stránky předávána jako callback funkci `requestAnimationFrame()`. Poté je na konci funkce `render()` nutné opět zavolat `requestAnimationFrame()` se stejným parametrem pro opětovné vykreslování scény.

Algorithm 3: Zjednodušený algoritmus vykreslení scény. Cyklus obstarávající samotné vykreslení musí být proveden dvakrát; jednou pro obyčejnou kameru a podruhé pro vybranou kameru light fieldu.

Input: Mesh object and scene uniforms.
Result: Scene rendered to a canvas element.
Prepare WebGL rendering context for drawing;
Split canvas into two parts;
Handle keyboard input;
foreach *camera* **do**
  $sceneUniforms$.set($cameraUniforms$, $camera$.getUniforms());
  foreach *bufferInfo and material in mesh* **do**
    $meshUniforms \leftarrow$ createUniforms($mesh.matrix$, $material$);
    setUniforms($sceneUniforms$, $meshUniforms$);
    setBuffers($bufferInfo$);
    Draw on half of the canvas;
  end
end

4.1.10 Propojení modulů

Každý modul si lze představit jako samostatnou třídu. Všechny tyto třídy umožňují v hlavním modulu `Renderer` jednoduchou a srozumitelnou manipulaci se scénou. Výsledné propojení modulů lze vidět na obrázku 4.2.



Obrázek 4.2: Diagram znázorňující propojení a vzájemné využití modulů. Směr šipek znázorňuje importování modulů. Modul `Renderer` za pomoci ostatních modulů obstarává samotné vykreslování scény a zpracování vstupů. Modul `Utils` zde není uveden, jelikož poskytuje pouze pomocné funkce pro všechny ostatní moduly a nepředstavuje žádný objekt z hlediska aplikace.

4.2 Nahrání souborů

Při zpracování souborů nahraných uživatelem je nutné klást důraz na korektní rozdělení souborů dle jejich přípony. Typy souborů, které aplikace podporuje, lze rozdělit do dvou kategorií:

1. Soubory obsahující geometrii – jedná se o textové soubory formátu OBJ nebo MTL. Při jejich zpracování je důležité zajistit, aby uživatel poskytl přesně 1 OBJ soubor a maximálně 1 MTL soubor, aby mohla být scéna v pořádku vykreslena. Tyto soubory jsou načteny jako textový řetězec pomocí objektu `FileReader` a následně spojeny do jednoho řetězce, který je předán funkci `parseObj()`.

2. Textury – do této kategorie obecně spadají veškeré obrázky (aplikace podporuje formáty PNG, JPG, GIF a TGA). Tyto soubory je nutné také zpracovat pomocí objektu `FileReader`, která z nahraného obrázku vytvoří HTML image element, z nehož je pomocí WebGL vytvořena textura. Všechny tyto textury jsou ukládány do pole a po úspěšném zpracování všech souborů předány funkci `parseObj()`.

Soubory je zapotřebí zpracovávat asynchronně, jelikož funkci `parseObj()` je nutné zavolat pouze jednou a předat jí veškeré zpracované soubory naráz. K tomu je využito `Promise` objektů. Tyto objekty reprezentují dokončení nebo selhání asynchronní operace a její výslednou hodnotu. Pro každý soubor je vytvořen nový `Promise` objekt a při úspěšném zpracování souboru je zavolána funkce `resolve()`, které je jako parametr předán výsledek zpracovaného souboru. Všechny tyto objekty jsou ukládány do pole, jež je předáno funkci `Promise.all()`, pomocí které může být po úspěšném zpracování všech souborů zavolána funkce `parseObj()` a vykreslena scéna.

4.3 Light field

Pro implementaci light fieldu bylo nutné realizovat několik funkcionalit z nichž nejzásadnější jsou:

- Možnost výběru pohledu kamery
- Samotné vytvoření light fieldu

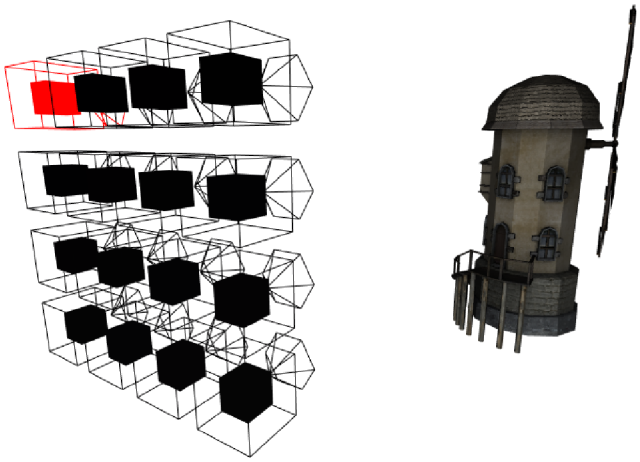
V této sekci je rozebrána a popsána implementace těchto funkcionalit.

4.3.1 Výběr kamery

Pro realizaci zvolení kamery kliknutím na kameru na plátně je využita funkce rozhraní WebGL `readPixels()`. Avšak tato funkce nečte pixely přímo z plátna. Místo toho jsou pixely získávány z framebufferu, proto je při inicializaci vytvořen framebuffer a textura, která bude do framebufferu vložena. Následně je při vykreslování nutné light field nejdříve vykreslit do textury, která je uložena ve framebufferu a pak až na samotné plátno.

Po vykreslení do textury je zavolána funkce `handlePicking()`, která obstarává samotný výběr kamery. Za pomoci callbacku při stisknutí tlačítka myši je zjištěna aktuální pozice kurzoru na plátně. Tato pozice je poté předána funkci `readPixels()`, pomocí které je načten aktuální pixel pod kurzorem. Poté už stačí jen iterovat přes kamery light fieldu a porovnat získanou barvu s barvou kamery a při shodě tuto kameru označit jako vybranou.

Využití framebufferu je velice užitečné, jelikož je do něj vykreslen pouze light field bez zbytku scény a je tak zachována konzistence výběru kamery. Nemůže tedy nastat situace, kdy by po kliku mimo light field byla přečtena stejná barva, jakou má přidělenu některá z kamer. Rozdíl mezi barvami kamer není na první pohled příliš znatelný, jelikož dochází pouze k malým změnám hodnot ve všech barevných kanálech. Uvnitř modelu každé kamery je navíc vykreslena malá krychle, která umožňuje jednodušší navolení kamery. Na obrázku 4.3 pak lze vidět reprezentaci light fieldu ve scéně.



Obrázek 4.3: Výsledná vizualizace light fieldu ve scéně. Uvnitř každé kamery je vykreslena malá krychle pro jednodušší kliknutí na danou kameru. Zvolená kamera je zvýrazněna červenou barvou. Na pravé straně se nachází vykreslovaný objekt.

4.3.2 Generování light fieldu

Pro vytvoření light fieldu ze scény je implementována funkce `exportLightField()`. Jelikož knihovna JSZip generuje archiv se soubory asynchronně, je nutné, aby tato funkce byla také asynchronní. Na začátku funkce je nastaven požadovaný rozměr plátna a vytvořen ukazatel průběhu tvorby light fieldu pomocí funkce `createLightFieldProgressBar()`. Zároveň je přerušen cyklus vykreslování pomocí funkce `cancelAnimationFrame()`.

Obrázky jsou generovány pomocí asynchronní verze funkce `iterateCameras()`, které je jako parametr předáván callback. Tento callback nejdříve provede vykreslení scény z pohledu aktuální kamery a poté vrátí nový `Promise` objekt, ve kterém je zavolána funkce HTML canvas elementu `toBlob()`. Tato funkce vytvoří snímek plátna, který je následně přidán do výsledného archivu s obrázky. Zároveň zde probíhá aktualizace stavu ukazatele průběhu.

Po úspěšném vygenerování souborů jsou zavolány funkce `generateAsync()` a `saveAs()` pro uložení archivu s obrázky. Následně je z HTML dokumentu odstraněn ukazatel průběhu tvorby light fieldu a plátnu jsou nastaveny jeho původní rozměry. Nakonec je obnoven cyklus vykreslování pomocí funkce `requestAnimationFrame()`.

Kapitola 5

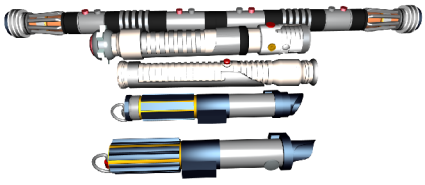
Měření

V této kapitole jsou uvedeny příklady některých scén[1], na kterých bylo provedeno měření rychlosti načtení scény a následné pořízení light fieldu. Měření bylo prováděno na stroji s procesorem Intel Core i7-5820K, 16 GB RAM a grafickou kartou NVIDIA GeForce GTX 750ti. Jako operační systém byl využit Microsoft Windows 10 Pro 64-bit s webovým prohlížečem Opera GX verze 73.0.3856.427.

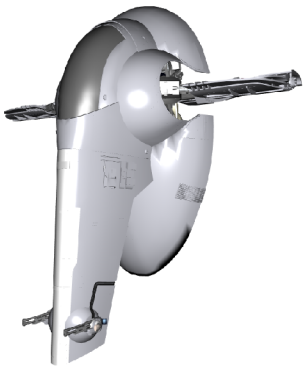
5.1 Zpracování scény

V tabulce 5.1 jsou uvedeny časy v sekundách potřebné pro načtení a následné vykreslení daných scén. Pro porovnání náročnosti bylo měření provedeno na scénách se složitější i jednodušší geometrií, s materiály a bez materiálů a s různými velikostmi textur.

Ze získaných výsledků lze vypozorovat, že i při poměrně jednoduché geometrii výrazně ovlivní čas nahrávání a zpracování textur. Naopak soubory s materiály nemají na výsledný čas téměř žádný vliv. Vizualizace některých scén lze vidět na obrázcích 5.1.



(a) Modely světelných mečů. Scéna má jednodušší geometrii než scéna Slave one, avšak její zpracování trvalo déle vlivem větší velikosti textur.



(b) Model vesmírné lodě Slave one.

Obrázek 5.1: Ukázky některých scén, na kterých bylo prováděno měření. Na obou scénách byly aplikovány materiály i textury.

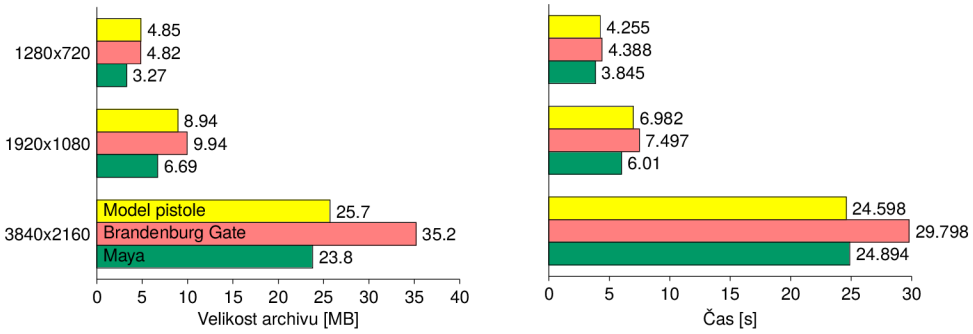
[1] Veškeré scény byly převzaty z těchto stránek: https://free3d.com/3d-models/obj-file

Vstupní scéna	Počet trojúhelníků	Materiály	Velikost textur [MB]	Čas [s]
Model pistole	646	Ne	0	0,021
		Ano	0	0,021
Maya	12 tis.	Ne	46,8	4,585
			0	0,364
		Ano	46,8	4,703
			0	0,462
Světelné meče	237 tis.	Ne	2,55	9,555
			0	8,105
		Ano	2,55	9,655
			0	8,1
Slave one	252 tis.	Ne	1,96	9,02
			0	8,789
		Ano	1,96	9,04
			0	8,955
Brandenburg Gate	390 tis.	Ne	2,01	8,681
			0	8,569
		Ano	2,01	8,718
			0	8,437

Tabulka 5.1: Rychlosti zpracování různých scén. Pro porovnání jsou zde uvedeny scény s jednoduchou i složitější geometrií a různými velikostmi textur.

5.2 Vytvoření light fieldu

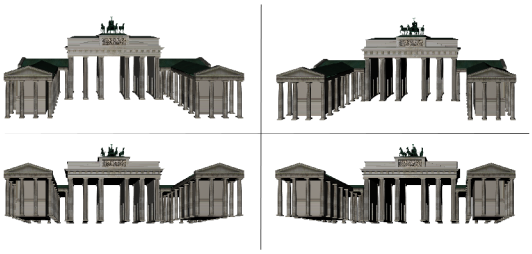
Tato sekce se zabývá měřením tvorby light fieldu z vybraných scén. Sledovanými faktory jsou velikost výsledného archivu a čas potřebný pro vytvoření a následné uložení celého light fieldu v závislosti na zvoleném rozlišení light fieldu. Dosažené výsledky jsou vizualizovány pomocí grafů 5.2. Na obrázcích 5.3 pak lze vidět light fieldy, které byly pořízeny z testovaných scén.



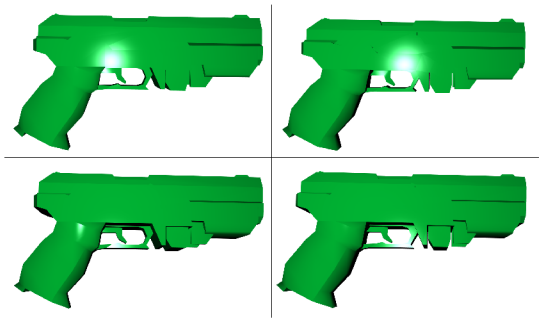
(a) Graf znázorňující výslednou velikost archivu při různých rozlišeních.

(b) Naměřené časy při pořizování jednotlivých light fieldů v různých rozlišeních.

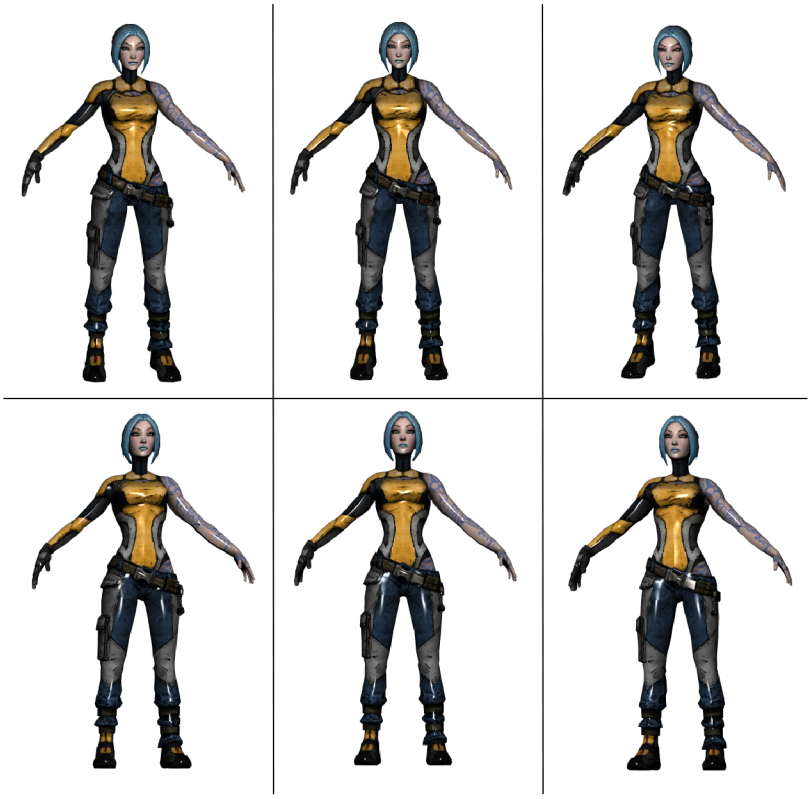
Obrázek 5.2: Hodnoty naměřené při tvorbě light fieldu z různých scén. Při generování light fieldu hraje největší roli výsledné rozlišení obrázků. Všechny light fieldy byly generovány s počtem kamer 8×8.



(a) Light field modelu památky Brandenburg Gate.



(b) Light field modelu pistole. Jelikož scéna neobsahuje textury ani informace o materiálech, je na její povrch nanesena zelená barva.



(c) Light field postavy známé jako Maya ze hry Borderlands 2. Na scéně lze pozorovat výraznější spekulární odrazivost, což je způsobeno chybějícími informacemi o materiálech a využitím implicitních hodnot pro materiály.

Obrázek 5.3: Ukázky pořízených light fieldů z vybraných scén. Všechny ukázky obsahují pro ilustraci pouze několik obrázků z celkového light fieldu.

Kapitola 6

Závěr

Cílem práce bylo vytvořit webovou aplikaci pro vytváření light fieldů z 3D scén ve formě diskrétní množiny obrázků. K řešení práce byla nejdříve nastudována teorie o light fieldech a možnostech zobrazování 3D scén ve webovém prohlížeči. Poté bylo navrhnuto uživatelské rozhraní společně s architekturou a jednotlivými funkcionalitami aplikace. Následně proběhla implementace jednotlivých částí aplikace pro nahrání scény, manipulaci se scénou a samotné vytvoření light fieldu. Nakonec bylo provedeno měření na různých scénách, díky kterému byla ověřena použitelnost celé aplikace.

Důležitou a prvotní součástí celého vývoje bylo seznámení se s rozhraním WebGL, jazykem GLSL a vývojem a použitím modulů v jazyce JavaScript. Za pomoci nabytých znalostí mohla být implementována výsledná aplikace pro tvorbu light fieldů. Tato aplikace byla nasazena na webové stránce[1] a její zdrojový kód byl zveřejněn na platformě GitHub[2].

Finální aplikaci lze využít pro tvorbu light fieldů z 3D scén. Aplikace nabízí základní možnosti pro manipulaci se scénou a prohlédnutí scény před samotným vytvořením light fieldu. Díky nastavením, která může uživatel light fieldu navolit, je možné ze scén vytvořit light fieldy o různých velikostech a rozlišeních.

Možná vylepšení

Aplikace je v současném stavu pro svůj účel použitelná, avšak nabízí se řada vylepšení, která by mohla aplikaci obohatit. Z implementačního hlediska by bylo velice vhodné využít TypeScript[3], jelikož je staticky typovaný a zvyšuje tak čitelnost zdrojového kódu. Analyzátor OBJ souborů momentálně nepodporuje veškeré možné záznamy ve vstupních souborech. Možnými rozšířeními by mohla být např. podpora 3D textur nebo volitelných argumentů jednotlivých záznamů v MTL souborech. V aplikaci je mřížka kamer implementována jako rovinná plocha. Možným vylepšením je přidat možnost si zvolit, zda má být light field reprezentován touto rovinnou plochou nebo kulovou plochou.

[1] Webová stránka aplikace: http://light-field-renderer.herokuapp.com
[2] Repozitář aplikace: https://github.com/Akroman/IBT
[3] Jedná se o jazyk rozšiřující JavaScript. Jeho nejdůležitějším aspektem je statické typování.

Literatura

[1] *The (New) Stanford Light Field Archive* [online]. Únor 2009. Dostupné z: http://lightfield.stanford.edu/lfs.html.

[2] Adelson, E. a Wang, J. Single Lens Stereo with a Plenoptic Camera. *IEEE Trans. Pattern Anal. Mach. Intell.* 1992, sv. 14, s. 99–106.

[3] Adelson, E. a Bergen, J. The Plenoptic Function and the Elements of Early Vision. Srpen 1997.

[4] Birklbauer, C., Schedl, D. C. a Bimber, O. Nonuniform Spatial Deformation of Light Fields by Locally Linear Transformations. *ACM Trans. Graph.* New York, NY, USA: Association for Computing Machinery. červen 2016, sv. 35, č. 5. DOI: 10.1145/2928267. ISSN 0730-0301. Dostupné z: https://doi.org/10.1145/2928267.

[5] Bolles, R. C., Baker, H. H. a Marimont, D. H. Epipolar-plane image analysis: An approach to determining structure from motion. *International journal of computer vision*. Springer. 1987, sv. 1, č. 1, s. 7–55.

[6] Chen, B., Ofek, E., Shum, H.-Y. a Levoy, M. Interactive Deformation of Light Fields. In: *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*. New York, NY, USA: Association for Computing Machinery, 2005, s. 139–146. I3D '05. DOI: 10.1145/1053427.1053450. ISBN 1595930132. Dostupné z: https://doi.org/10.1145/1053427.1053450.

[7] Gershun, A. The light field. *Journal of Mathematics and Physics*. 1939, sv. 18, 1-4, s. 51–151.

[8] Gortler, S. J., Grzeszczuk, R., Szeliski, R. a Cohen, M. F. The Lumigraph. In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: Association for Computing Machinery, 1996, s. 43–54. SIGGRAPH '96. DOI: 10.1145/237170.237200. ISBN 0897917464. Dostupné z: https://doi.org/10.1145/237170.237200.

[9] Horn, D. a Chen, B. LightShop: Interactive light field manipulation and rendering. In:. Leden 2007, s. 121–128. DOI: 10.1145/1230100.1230121.

[10] Isaksen, A., Mcmillan, L. a Gortler, S. Dynamically Reparameterized Light Fields. *Proc. Siggraph*. Leden 2002, sv. 2000. DOI: 10.1145/344779.344929.

[11] Levoy, M. a Hanrahan, P. Light Field Rendering. In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: Association for Computing Machinery, 1996, s. 31–42. SIGGRAPH '96.

DOI: 10.1145/237170.237199. ISBN 0897917464. Dostupné z: https://doi.org/10.1145/237170.237199.

[12] Likun, S., Zhou, W. a Chen, Z. No-Reference Light Field Image Quality Assessment Based on Spatial-Angular Measurement. Srpen 2019.

[13] Maeno, K., Nagahara, H., Shimada, A. a Taniguchi, R.-i. Light Field Distortion Feature for Transparent Object Recognition. In:. červen 2013, s. 2786–2793. DOI: 10.1109/CVPR.2013.359.

[14] Ng, R., Levoy, M., Br, M., Duval, G., Horowitz, M. et al. Light Field Photography with a Hand-held Plenoptic Camera. In:. 2005.

[15] Seitz, S. M. a Kutulakos, K. N. Plenoptic image editing. In: *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*. 1998, s. 17–24. DOI: 10.1109/ICCV.1998.710696.

[16] Venkataraman, K., Lelescu, D., Duparre, J., McMahon, A., Molina, G. et al. PiCam: An Ultra-Thin High Performance Monolithic Camera Array. In:. Listopad 2014, sv. 32. DOI: 10.1145/2508363.2508390.

[17] Wang, T.-C., Zhu, J.-Y., Hiroaki, E., Chandraker, M., Efros, A. et al. A 4D Light-Field Dataset and CNN Architectures for Material Recognition. In:. říjen 2016, sv. 9907, s. 121–138. DOI: 10.1007/978-3-319-46487-9_8. ISBN 978-3-319-46486-2.

[18] Wilburn, B., Joshi, N., Vaish, V., Talvala, E.-V., Antúnez, E. et al. High Performance Imaging Using Large Camera Arrays. *ACM Trans. Graph.* Červenec 2005, sv. 24, s. 765–776. DOI: 10.1145/1073204.1073259.

[19] Wilburn, B., Smulski, M., Lee, K. a Horowitz, M. The Light field video camera. *In: Media Processors*. Prosinec 2001, sv. 2002. DOI: 10.1117/12.451074.

[20] Yang, J., Everett, M., Buehler, C. a Mcmillan, L. A Real-Time Distributed Light Field Camera. In:. Leden 2002, s. 77–86.

[21] Zhang, C. a Chen, T. A Self-Reconfigurable Camera Array. In: *ACM SIGGRAPH 2004 Sketches*. New York, NY, USA: Association for Computing Machinery, 2004, s. 151. SIGGRAPH '04. DOI: 10.1145/1186223.1186412. ISBN 1581138962. Dostupné z: https://doi.org/10.1145/1186223.1186412.

Příloha A

Obsah paměťového média

- **app/** – výsledná aplikace, obsahuje zdrojové kódy, knihovny, konfigurace a složku **examples/** s ukázkovými scénami
- **doc/** – programová dokumentace vygenerovaná pomocí JSDoc
- **text/** – obsahuje text práce v PDF a složku **src/** se zdrojovým kódem textu v LaTeX.
- **video/** – demonstrační video
- **README.md** – návod k instalaci a spuštění aplikace, odkaz na webové stránky, na kterých je aplikace nasazena