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

NEREALISTICKÉ ZOBRAZOVÁNÍ NA LUME PAD

NON-PHOTOREALISTIC RENDERING ON LUME PAD

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **ANETA CHALIVOPULOSOVÁ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. TOMÁŠ MILET, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

146192

Ústav:	Ústav počítačové grafiky a multimédií (UPGM)
Studentka:	**Chalivopulosová Aneta**
Program:	Informační technologie
Specializace:	Informační technologie
Název:	**Nerealistické zobrazování na Lume Pad**
Kategorie:	Počítačová grafika
Akademický rok:	2022/23

Zadání:

1. Nastudujte techniky nerealistického zobrazování, engine Unity, dokumentaci Lume Pad, teorii stereoskopického zobrazování, lightfield a holografii.
2. Vyberte několik (alespoň tři) techniky nerealistického zobrazování a navrhněte aplikaci v enginu Unity, která techniky využije. Cílem je zjistit, jak jsou dané techniky nerealistického zobrazování vhodné pro zobrazování na autostereoskopickém displeji.
3. Implementujte navrženou aplikaci.
4. Vyhodnoťte výsledky a sepište závěry.
5. Vytvořte demonstrační video a aplikaci zveřejněte.

Literatura:

- Developing for Lume Pad
- Jan Eric Kyprianidis, John Collomosse, Tinghuai Wang, Tobias Isenberg. State of the ”Art”: A Taxonomy of Artistic Stylization Techniques for Images and Video. IEEE Transactions on Visualization and Computer Graphics, Institute of Electrical and Electronics Engineers, 2013, 19 (5), pp.866-885. 10.1109/TVCG.2012.160. hal-00781502

Při obhajobě semestrální části projektu je požadováno:
Body jedna a dva a kostra aplikace.

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

Vedoucí práce:	**Milet Tomáš, Ing., Ph.D.**
Vedoucí ústavu:	Černocký Jan, prof. Dr. Ing.
Datum zadání:	1.11.2022
Termín pro odevzdání:	10.5.2023
Datum schválení:	31.10.2022

Abstrakt

Tato bakalářská práce se zabývá nerealistickými metodami zobrazování na Lume Pad. Cílem práce je ověřit funkčnost, kvalitu a uživatelskou přívětivost implementace několika různých nerealistických efektů na 3D scénách zobrazovaných na holografickém tabletu Lume Pad. Pro testování je využita aplikace vyvinutá v nástroji Unity. Je implementováno několik vybraných nerealistických efektů využívajících post-processing, které jsou také v rámci této práce popsány. Jmenovitě jde o detekci hran, maticový dithering, náhodné rozptýlení, prahování, šrafování, barevné stínování, efekt pokřivení a zvlnění obrazu a efekt tahů tužky. Dále je v textu popsán teoretický úvod do holografie, steroskopie a light field.

Abstract

This bachelor's thesis focuses on methods of non-realistic rendering on Lume Pad. This thesis aims to test the functionality, quality, and user-friendliness of a few selected non-realistic effects applied to 3D scenes on the holographic tablet Lume Pad. These non-realistic post-processing effects are implemented and described in this thesis. Namely, edge detection, ordered dithering, random dithering, thresholding, hatching, color shading, wobbling effect and wobble carving effect, and pencil effect. The text also describes a theoretical introduction to holography, stereoscopy, and the light field.

Klíčová slova

nerealistické zobrazování, nerealistické efekty, post-processing, light field, holografie, stereskopie, hardware pro zobrazování ve 3D, Lume Pad, Unity, zobrazovací řetězec, shader, detekce hran, dithering, šrafování, efekt tahů tužky, barevné stínování

Keywords

non-realistic rendering, non-realistic effects, post-processing, light field, holography, stereoscopy, harware for viewing 3D, Lume Pad, Unity, rendering pipeline, shader, edge detection, dithering, hatching, pencil effect, color shading

Citace

CHALIVOPULOSOVÁ, Aneta. *Nerealistické zobrazování na Lume Pad.* Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Milet, Ph.D.

Nerealistické zobrazování na Lume Pad

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením pana Ing. Tomáše Mileta, Ph.D.. Uvedla jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpala.

.....
Aneta Chalivopulosová
9. května 2023

Obsah

Kapitola 1

Úvod

Počítače a počítačová grafika jsou nyní již nedílnou součástí našich životů. Počítačová grafika má často za cíl dosáhnout co největší přesnosti a omezit míru rozdílů digitálně zobrazovaného a reálného světa. Realistické zobrazování světa je důležité například ve filmovém či v herním průmyslu, v oblasti virtuální reality nebo také her simulujících život.

Ovšem ne vždy je důležité napodobit reálný svět zcela přesně. V již zmíněném herním průmyslu jsou velmi často hry upravovány do různých stylů, aby jejich grafická podoba navozovala hráči požadovanou atmosféru. Nerealistické zobrazování reality má také praktické účely jako zobrazení aktivních objektů ve scéně hry či upozornění hráče na důležitou součást scény.

Nerealistické efekty napodobují výtvarné techniky a styly. Příkladem může být komiksový styl s výraznými obrysy a stínováním, často zanedbávající nedůležité detaily. Dále se jedná například o napodobení malby, tedy tahů štětcem s nepříliš výraznými obrysy objektů.

V této práci jsou popsány vybrané nerealistické efekty a tyto efekty jsou implementovány na platformě Lume Pad. Lume Pad je holografický tablet přinášející možnost zobrazení obsahu obrazovky ve 3D prostoru na rozdíl od ploché 2D obrazovky klasického počítače. V práci je porovnána kvalita a uživatelská přívětivost implementace těchto efektů na tabletu Lume Pad ve 3D a 2D režimu. K porovnání slouží vyvinutá aplikace obsahující různá nastavení pro testování. Cílem této práce je tedy získat data o kvalitě zobrazení nerealistických efektů ve 3D.

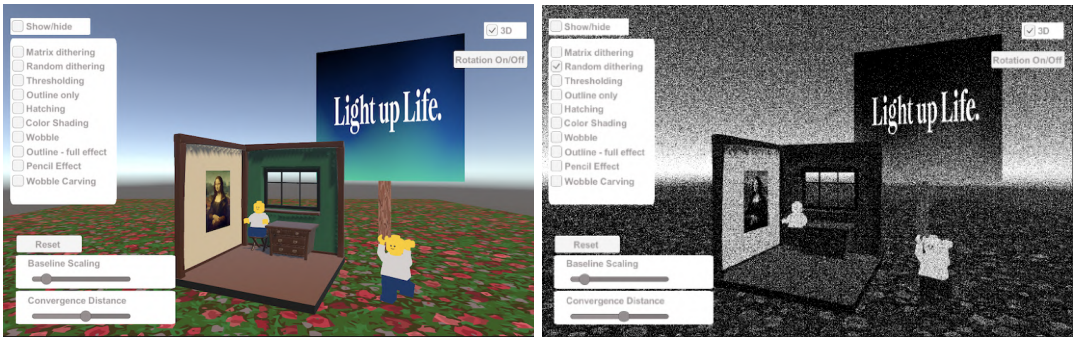
Druhá kapitola obsahuje představení problematiky a popis postupu práce. Ve třetí až páté kapitole je obsažen teoretický úvod – zobrazování ve 3D, nerealistické efekty a jejich algoritmy a popis nástroje Unity sloužící pro implementaci aplikace. Dále je v textu popsán samotný návrh a implementace aplikace, testování s uživateli za pomoci výsledné aplikace a výsledky testování.

Kapitola 2

Představení problematiky a postup práce

Cílem práce je zjistit, zda je možné nerealistické efekty implementovat na stereoskopický displej tabletu Lume Pad ve srovnatelné kvalitě a míře uživatelské přívětivosti jako na klasickém plochém displeji. K zjištění těchto skutečností je využito testování s uživateli. Jako nástroj pro testování slouží aplikace zobrazující několik efektů aplikovaných na scéně (aplikace je zobrazená na obrázku 2.1). Podle sesbíraných dat je možné odvodit, jak dobře je možné efekty na stereoskopický displej implementovat.

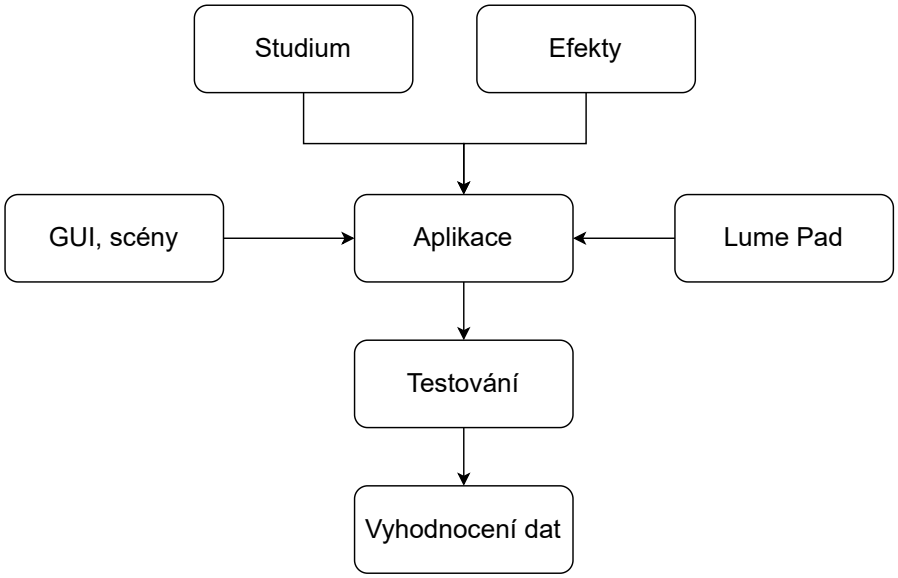
Lume Pad je poměrně novou platformou, která umožňuje zobrazení scén ve 3D bez nutnosti dalších pomůcek, jako například 3D brýlí. Holografický obraz vystupuje z tabletu směrem k uživateli a přináší tím zcela nový rozměr pohledu na scénu.



Obrázek 2.1: Výsledná aplikace implementovaná v nástroji Unity sloužící k testování. Testování slouží k zjištění, zda je možné efekty na stereoskopickém tabletu Lume Pad ve 3D režimu využívat ve srovnatelné kvalitě jako ve 2D režimu. Pravý obrázek zobrazuje efekt náhodného rozptýlení, který je jedním z implementovaných a popsaných efektů v této práci.

2.1 Postup práce

Postup práce je složen z několika hlavních částí zobrazených na diagramu 2.2 . Mimo jiné se jedná o studium potřebných teoretických znalostí, jako je light field a holografie, nerealistické efekty a další (kapitoly 3 a 4). Dále je třeba seznámit se s problematikou nástroje Unity, shaderů, zobrazovacího řetězce (kapitola 5) a tabletu Lume Pad (sekce 3.4.1). Aplikace je implementována pomocí nástroje Unity, ve kterém je také vytvořeno jednoduché uživatelské prostředí pro ovládání aplikace (kapitola 7). Poslední důležitou částí práce je návrh testů a samotné testování (kapitola 8).



Obrázek 2.2: Diagram postupu práce. Chronologicky: prvním krokem je studium teoretických znalostí (označeno **Studium**) a nerealistických efektů (**Efekty**), dále návrh a implementace aplikace (**Aplikace**) pomocí nástroje Unity a připravení aplikace na cílovou platformu Lume Pad (**Lume Pad**). Součástí aplikace je také návrh a implementace uživatelského rozhraní a scén (**GUI, scény**). Posledním krokem je testování s uživateli (**Testování**) a vyhodnocení dat z testování (**Vyhodnocení dat**).

Kapitola 3

Zobrazování ve 3D

Zobrazovat objekty a scény ve 3D je dnes již důležitou součástí moderní techniky. Sen o vytvoření přesného hologramu se dlouhodobě objevoval ve filmech a dnes je již blíže skutečnosti. K zobrazení ve 3D jsou často potřeba speciální pomůcky, jako jsou například brýle. Existují ale také techniky, ke kterým brýle ani jiné pomůcky pro uživatele nejsou potřeba – tento princip využívá například tablet Lume Pad, který je cílovou platformou v této práci.

Velká část technik zobrazování ve 3D využívá binokulárního vidění. Binokulární vidění označuje, jak je uvedeno ve zdroji [17], vidění oběma očima, což umožňuje člověku vnímat prostorově. Obraz ze zorného pole levého a pravého oka se spojuje v jeden a vzniká tak prostorový obraz.

3.1 Stereoskopické zobrazování

Stereoskopie je technika, která umožňuje vyvolat prostorový zrakový dojem z dvourozměrné předlohy. Způsobů, jak tohoto jevu docílit, je mnoho. Některé z nich jsou naznačeny níže.

Stereokotoučky Stereokotouček (na obrázku 3.1) je disk, který obsahuje čtrnáct obrázků tvořících sedm párů. Po vložení do prohlížečky se vytvoří prostorový obraz. Často sloužily jako suvenýry z různých turistických míst a na trh přišly již v první polovině 20. století.

Obrázek 3.1: Na obrázku jsou zobrazeny stereokotoučky, společně s prohlížečkou. Obrázek je přejatý z internetového zdroje [1]

[1] Viewmaster stereoscopic 3D viewer, BBC, https://www.bbc.co.uk/ahistoryoftheworld/objects/zkFmx_c9QM6C23Ld3Qd6sg

Anaglyf Anaglyf [16] je jedním z nejjednodušších, nejlevnějších a také nejznámějších stereoskopických zobrazení. K této technice jsou využity obvykle červeno-modré brýle. Obrázek pro anaglyf vzniká tak, že se převede do červeno-bílých a do modro-bílých odstínů a tyto dva nové obrazy se poté spojí. Pomocí anaglyfu často nevzniká přesný obraz a také nemusí být pro všechny stejně dobře viditelný.

Obrázek 3.2: Srovnání původního obrazu a anaglyfu. Na anaglyfu je možné vidět červené a modré „obrysy“, které po nasazení červeno-modrých brýlí způsobí, že se obraz bude jevit prostorový. Původní obrázek z vlastního archivu autora byl převeden do anaglyfu pomocí internetového nástroje [2].

Polarizovaná skla brýlí Přesnější metodou, která také využívá speciální brýle podobně jako anaglyf, je metoda použití různě polarizovaných skel brýlí. Tato technika bývá využita v kinech k promítání 3D filmů.

Dalšími způsoby zobrazení 3D jsou holografie a také obrazovky využívající light field, tyto dvě techniky jsou blíže popsány v nasledujících sekcích 3.3 a 3.2.

3.2 Holografie

Teoretické základy pro holografii položil v roce 1948 maďarsko-britský fyzik Dennis Gabor (1900-1979) při jeho snaze zdokonalit elektronový mikroskop. První trojrozměrný záznam se podařilo vytvořit v roce 1962 vědcům Emmettu Leithovi a Jurisu Upatnieksovi, jak uvádí práce [16].

Cílem holografie je zobrazit prostorový obraz na dvourozměrný nosič. Holografický obraz obsahuje informaci o intenzitě světla, jako u klasické fotografie, a navíc informaci o fázi světla – reprezentující směr světla. Tedy zjednodušeně informaci o tom, jak se světlo na povrchu objektu rozptyluje – podle toho lze určit prostorové rozložení objektu.

Hologramy se, jak je uvedeno v internetovém zdroji [18], dělí do dvou kategorií – transmisní a reflexní.

Transmisní hologramy Transmisní hologramy vznikají tak, že pomocí vhodných optických prostředků rozdělí dostatečně široký svazek světelných paprsků laseru na dva svazky – osvětlovací a referenční. Osvětlovací svazek dopadá na objekt a jeho odrazem pak vzniká

[2] https://convertimage.net/online-photo-effects/create-anaglyph-stereoscopic-3d-images-online.asp

předmětový svazek. Ten v sobě nese informaci o rozložení intenzity světla a také fázi odražené vlny. Tyto informace jsou zaznamenány na dvourozměrném médiu pomocí toho, že se předmětový svazek prolíná s referenčním svazkem. (text je částečně převzatý z [11])

Reflexní hologramy Reflexní hologramy pracují pouze s jedním svazkem světelných paprsků laseru. Tento paprsek se poté prolíná s paprskem odraženým od zobrazovaného předmětu.

U obou typů hologramů vzniká obraz objektu až po rekonstrukci hologramu pomocí dalších světelných paprsků z laseru.

Hologramy se, jak je uvedeno v internetovém zdroji [23], v běžné praxi využívají zejména pro ochranu dokumentů – například pro ochranu před padělaním bankovek. U bankovek jde o malou samolepku se strukturou nejčastěji z plastu, která vytváří hologram. Dalším využitím jsou například holografické paměti.

3.3 Light field

Light field obsahuje kompletní model obrazu, tedy všechny paprsky světla v prostoru procházející skrz všechny body ve všech směrech.

Pro popis všech světelných paprsků v prostoru slouží *plenoptická funkce* [12]. Tato funkce (uvedená na obrázku 3.3), stejně jako light field reprezentace, je kompletním modelem obrazu. Popisuje všechny informace o obraze, včetně například intenzity a barvy světla. Pokud by se podařilo získat všechny světelné paprsky mezi plochou promítání a objektem, vznikl by plně věrohodný holografický obraz objektu. Toto ovšem v reálném světě zatím není možné.

3.3.1 Plenoptická funkce

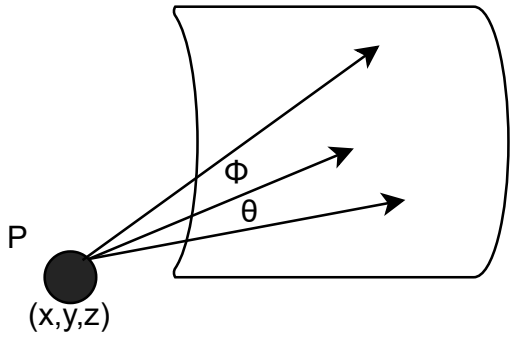
Plenoptická funkce (na obrázku) obsahuje podle původního znění sedm parametrů, jak je uvedeno ve zdrojích [13] a [15]:

- tři prostorové – pozice pozorovatele, respektive kamery,
- dva směrové – otočení pozorovatele, respektive kamery,
- vlnovou délku,
- čas.

Tuto 7D plenoptickou funkci lze charakterizovat vztahem 3.1 uvedeným níže. Vztah popisující plenoptickou funkci. Prostorové parametry kartézského systému souřadnic (x, y, z) popisují pozici. Sférické souřadnice (θ, ϕ) popisují směr. Parametr λ popisuje vlnovou délku a τ popisuje čas.

$$l(x, y, z, \theta, \phi, \lambda, \tau) \tag{3.1}$$

Plenoptická funkce je často zjednodušena pouze na pět rozměrů, je zanedbána vlnová délka a čas.



Obrázek 3.3: Nákres plenoptické funkce. Písmenem **P** je označen pozorovatel s prostorovými souřadnicemi *(x,y,z)* a sférickými souřadnicemi otočení (θ, ϕ).

3.4 Hardware pro zobrazování ve 3D

Hardwarových technologií pro zobrazení obsahu ve 3D je poměrně mnoho. Některé technologie jsou starší jako například Nintendo 3DS, jiné naopak velmi nové, jako je Lume Pad či produkty firmy Looking Glass Factory. Technologie se stále vylepšují a již dnes přinášejí poměrně kvalitní zobrazení 3D.

3.4.1 Lume Pad

Lume Pad je tablet s operačním systémem Android přinášející možnost zobrazení obsahu ve 3D bez nutnosti klasických 3D brýlí. Zobrazení ve 3D funguje na principu light field technologie – konkrétně Diffractive Lightfield Backlighting [2]. Při této light field technologii se výsledný obraz skládá ze čtyř různých pohledů. Složením čtyř pohledů vzniká 3D efekt – holografický obraz „vystupující“ z obrazovky směrem k uživateli. Nástupce původního Lume Padu z roku 2021, který je použitý v této práci, je Lume Pad 2 z roku 2023. Lume Pad 2 disponuje light field technologií složenou z osmi pohledů.

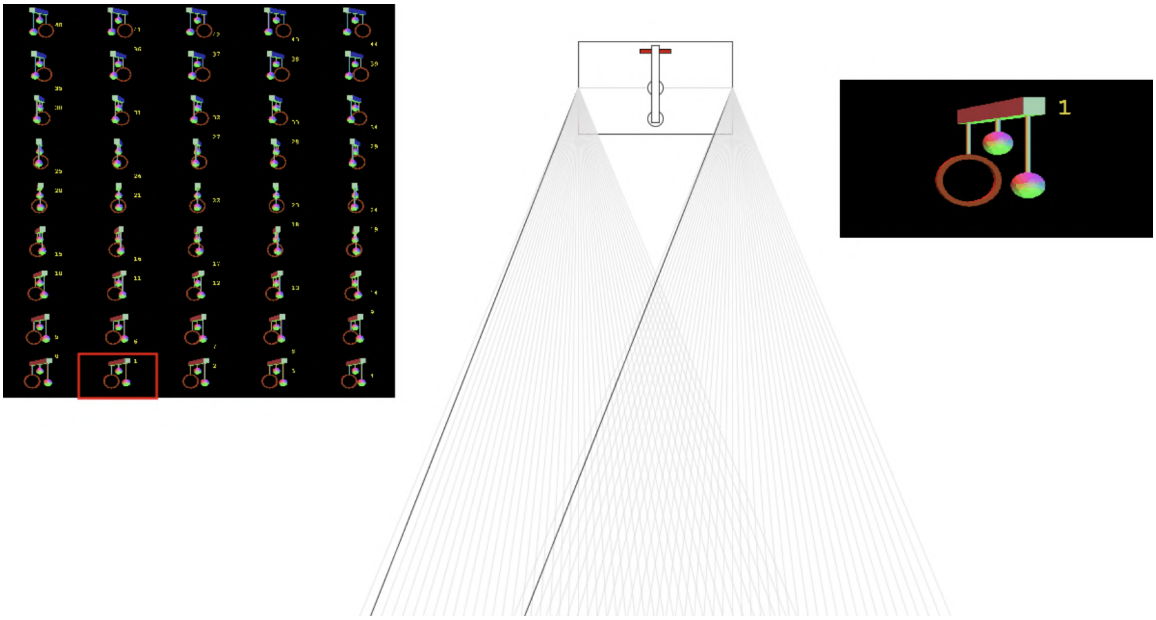
Lume Pad nabízí možnost přepnutí zpět do 2D režimu. Důvodem je mimo jiné to, že technologie pro zobrazení 3D je obecně náročnější na výkon a nabízí menší rozlišení a rychlost. Po přepnutí do 2D režimu je tedy znovu k dispozici vysoké rozlišení, menší zátěž baterie a standardní obnovovací frekvence displeje [6].

3.4.2 Looking Glass Factory

Looking Glass Factory [1] je americká firma zabývající se vývojem holografických zobrazovacích zařízení. V roce 2020 uvedli na trh první osobní holografický displej a v roce 2022 největší holografický displej. Looking Glass Factory v tuto chvíli nabízí několik holografických displejů – největší z nich je Looking Glass 65”, poté Looking Glass 32” a Looking Glass 16”, nejmenší je pak Looking Glass Portrait (obrázek 3.4).

Stejně jako Lume Pad funguje i tento displej na základě light field technologie. Displej poskytuje sto pohledů, které po spojení vytvářejí 3D dojem viditelný pod zorným kuželem širokým asi 58 stupňů. Displej směruje různé pohledy podle diagramu na obrázku 3.5.

Obrázek 3.4: Looking Glass Portrait je nejmenším z nabízených displejů. Obrázek je převzatý z oficiální webové stránky Looking Glass Factory [3].



Obrázek 3.5: Diagram směrování pohledů v Looking Glass displeji. Pro každý úhel pohledu je zobrazen jeden z pohledů. Všech 45 pohledů je na na obrázku vlevo. Uprostřed je samotný Looking Glass displej společně s naznačeným úhlem pohledu. Tmavými čarami v úhlu pohledu, červeným rámečkem vlevo a obrázkem vpravo je označen právě vybraný pohled pro daný úhel. Obrázek i popis je převzatý z webové dokumentace k Looking Glass displeji [4].

[4] https://lookingglassfactory.com/product/looking-glass-portrait
[4] https://docs.lookingglassfactory.com/keyconcepts/how-it-works

3.4.3 Nintendo 3DS

Nintendo 3DS je přenosná konzole z roku 2010, která umožňuje stereoskopické zobrazení bez speciálních pomůcek jako například brýlí. Konzole také obsahuje posuvník (zobrazený na obrázku 3.6) pro nastavení hloubky 3D efektu, který přinášel pro uživatele možnost přizpůsobit si individuálně hloubku prostorového dojmu.

Obrázek 3.6: Nintendo 3DS. Vrchní ze dvou displejů je displej disponující možností zobrazení her ve 3D režimu. Spodní z displejů je dotykový pro ovládání konzole. Modrou barvou je označen posuvník pro nastavení hloubky 3D efektu. Pokud je posuvník na nejnižší hodnotě, je 3D režim úplně vypnutý. Obrázek je z osobního archivu autora.

Rozdílem oproti některým jiným hardwarovým možnostem, jako například Lume Pad, je to, že k zobrazení her ve 3D musel uživatel využít pouze speciálních herních karet uzpůsobených 3D režimu. Tedy například herní karty ze starší přenosné konzole Nintendo DS byly hratelné pouze ve 2D režimu. Pro zobrazení 3D využívá tzv. paralaxové bariéry, jak uvádí Jonathan Strickland na webové stránce [19]. Paralaxová bariéra je další vrstva – LCD – nad displejem konozole. Krystaly LCD (liquid crystal display) vytváří bariéry, které usměrňují světlo. Pokud je 3D režim vypnutý, krystaly umožní světlu procházet volně, a tedy obě oči dostanou stejný obraz – klasické 2D zobrazení. Pokud je 3D režim zapnutý, upraví se umístění krystalů paralaxové bariéry, a tak každé oko dostane jiný obraz – vzniká prostorový dojem. Nevýhoda této techniky stereoskopického zobrazení je, že přívětivý a dobře viditelný 3D dojem je možný pouze v určitém poměrně úzkém úhlu pohledu.

Nintendo 3DS umožňuje také pořizování fotografií ve 3D režimu. Konzole obsahuje dva zadní fotoaparáty (zobrazeno na obrázku 3.7), obrazy z obou fotoaparátů se spojí a vzniká tak 3D dojem.

Obrázek 3.7: Dva zadní fotoaparáty konzole Nintendo 3DS. Pomocí dvou rozdílných obrazů a jejich následného spojení lze pořizovat fotografie ve 3D. Obrázek je z osobního archivu autora.

3.5 Barva a barevné modely

Barva je vjem, který vzniká v mozku pomocí vnímání odraženého světla od pozorovaného objektu okem. Pro míchání různých odstínů barev je třeba zvolit vhodné základní barvy, tedy například červenou, zelenou a modrou. Michání barev lze rozdělit do dvou kategorií:

- **Aditivní** míchání barev funguje na principu práce se světlem a je využíváno například v monitorech. Základními barvami jsou červená, zelená a modrá – *red, green, blue*. Složením všech tří složek v nejvyšší intenzitě vznikne bílá barva.
- **Substraktivní** míchání barev funguje na principu práce s pigmenty a je využíváno například v tiskárnách. Základními barvami jsou azurová, purpurová a žlutá – *cyan, magenta, yellow*. Složením všech tří složek v nejvyšší intenzitě vznikne černá barva.

Hlavními barevnými modely založenými na základě způsobu míchání barev jsou RGB a CMY modely, jak uvádí Jana Dannhoferová v knize [10]. RGB model odpovídá aditivnímu míchání barev a CMY (případně CMYK po přídání černé barvy do základní sady barev) odpovídá substraktivnímu míchání barev. Mezi další modely patří například HSV model či chromatický diagram.

Model RGBA je založený na modelu RGB, ale přidání další čtvrtou složku popisu barvy pixelu, a to alfa kanál. Alfa kanál udává informaci o průhlednosti pixelu.

Alpha blending Alpha blending je technika spojení obrázku s jiným obrázkem, zpravidla pozadím, za účelem dosáhnutí určité průhlednosti. Tato technika je používaná mimo jiné ve filmovém průmyslu. V počítačové grafice často slouží k přidání elementů nad vrstvu pozadí.

Pokud uvažujeme výpočet barvy jednoho pixelu při technice alpha blending, lze jej provést, jak je naznačeno na rovnici 3.2 a jak je uvedeno v knižních zdrojích [21], [8] a internetovém zdroji [4]. C označuje výslednou barvu pixelu, C_1 a C_2 označují barvy původních obrázků a α_1 označuje alfa kanál barvy C_1.

$$C = C_1 + C_2 \cdot (1 - \alpha_1) \tag{3.2}$$

Tento způsob výpočtu se označuje jako „pre-multiplied alpha“. Alfa kanál i RGB složky barvy jsou při této technice propojeny. Aby byl pixel průhledný, je třeba snížit hodnoty složek RGB i alfa kanálu.

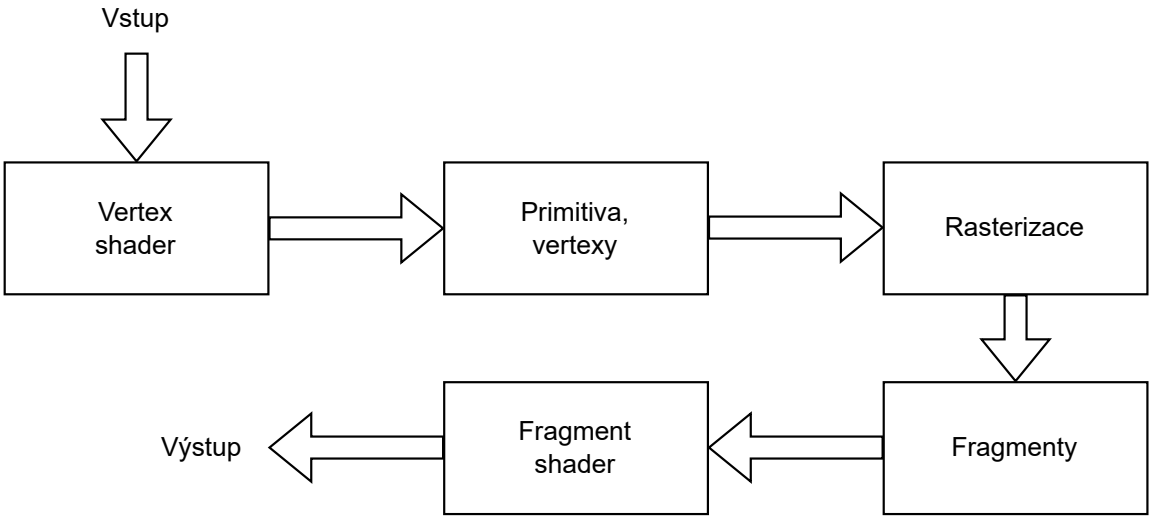
Druhým způsobem je „straight alpha“, výpočet pomocí této techniky naznačen rovnicí 3.3. Při této metodě jsou RGB složky a alfa kanál samostatné. RGB udává barvu pixelu a alfa kanál jeho průhlednost. Pro úplnou průhlednost je tedy třeba nastavit pouze hodnotu alfa kanálu, RGB složky zůstávají stejné.

$$C = (C_1 \cdot \alpha_1) + (C_2 \cdot (1 - \alpha_1)) \tag{3.3}$$

V počítačové grafice se více využívá „pre-multiplied alpha“, protože pomocí této techniky lze dosáhnout lepších výsledků.

3.6 Zobrazovací řetězec grafické karty

Zobrazovací řetězec (anglicky „rendering pipeline“) v počítačové grafice popisuje proces použitý k vykreslení 2D či 3D scény na obrazovku počítače. Vstupem pro zobrazovací řetězec jsou jednotlivé vertexy – trojúhelníky – scény a textury objektu a výstupem pak výsledný objekt s aplikovanými texturami. Jednotlivé kroky zobrazovacího řetězce se liší podle použité platformy. Obvykle obsahuje mimo jiné vertex a fragment shader, tyto kroky jsou také zobrazeny na obrázku 3.8 níže.



Obrázek 3.8: Zjednodušené obecné schéma zobrazovacího řetězce. Vstup nejprve projde vertex shader, který produkuje primitiva, respektive vertexy. Vertexy poté projdou rasterizací. Výstupem rasterizace jsou fragmenty, tyto fragmenty jsou zpracovány fragment shaderem a poté tvoří výstup zobrazovacího řetězce. Výstupem zobrazovacího řetězce je framebuffer obsahující pixely scény. Informace pro zhotovení schématu jsou přebrány z knižního zdroje [24] a internetového zdroje [22]

Vertex shader Vertex shader je jednou z programovatelných částí zobrazovacího řetězce. Vstupem je jeden vertex s atributy a výstupem je také jeden vertex s atributy – musí být zajištěno mapování 1:1. Kromě programátorem definovaných atributů vertexu na výstupu obsahuje také speciální atribut, a to výslednou pozici vertexu. Pomocí vertex shaderu lze měnit pozici, barvu či například texturu vertexu.

Součástí vertex fáze zobrazovacího řetězce je také vytvoření primitiv z vertex dat a jejich příprava pro rasterizaci.

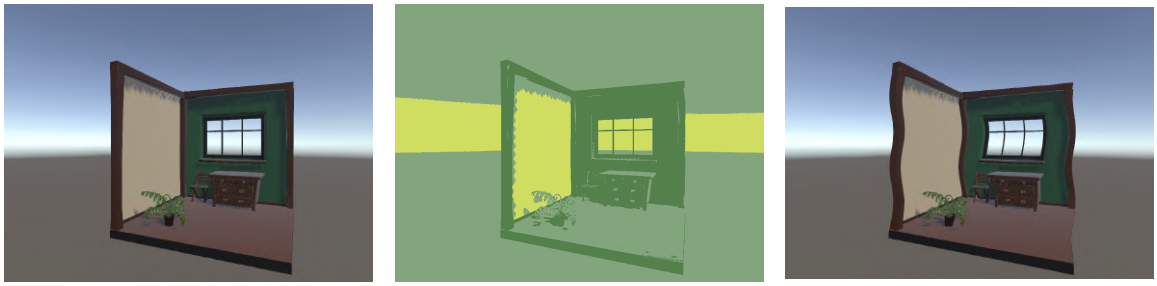
Rasterizace Proces rasterizace dostane na vstupu primitiva, které jsou rasterizována. Výsledkem rasterizace je sekvence fragmentů. Fragment následně slouží k výpočtu konečných dat pro pixel ve výstupním framebufferu.

Fragment shader Fragment shader je zpravidla poslední částí zobrazovacího řetězce, v rámci něj lze měnit například barvu fragmentu. Vstupem je jeden fragment obrazu, výstup je jeden pozměněný fragment.

Součástí fragmentu jsou obvykle pozice v prostoru scény, barva, hloubka a další. Pozice jsou většinou dvě – jedna ve 2D a druhá ve 3D prostoru. Ve 2D jde o tzv. „UV“ pozici vycházející z UV mapování, tedy převedení povrchu trojrozměrného objektu do 2D prostoru. Druhá z pozic je pozice ve 3D obsahující souřadnice na ose x, y a z.

3.6.1 Shadery a post-processing

Shader je počítačový program běžící na GPU (*graphics processing unit*), který slouží ke specifikaci chování jednotlivých částí zobrazovacího řetězce GPU. Používají se ke změně vzhledu celých scén nebo jen jednotlivých objektů scény.



Obrázek 3.9: Na obrázku vlevo je zobrazena scéna před aplikací post-processing efektů. Prostřední obrázek obsahuje efekt měnící barvu pixelu a pravý obrázek efekt změny pozice pixelu.

Shadery ovlivňují chování částí zobrazovacího řetězce, pro realizaci různých efektů pro scénu či objekty je zejména využíváno fragment shaderu.

Existuje několik druhů shaderů, které by se dalo rozdělit do tří hlavních kategorií. První kategorií jsou **shadery, které jsou částí grafického zobrazovacího řetězce**. Tyto shadery definují barvu pixelu a jsou nejčastěji používaným druhem shaderů.

Další kategorií shaderů jsou **„Compute Shadery“**. Tato kategorie shaderů provádí výpočty na GPU, ovšem mimo klasický zobrazovací řetězec. Umožňují programování paralelních algoritmů důležitých například k zvýšení výkonnosti a rychlosti vyvíjené hry. Implementace compute shaderů je zpravidla složitější a jsou k ní potřeba dostatečné znalosti paralelního programování.

Poslední kategorií shaderů jsou **„Ray tracing shadery“**. Tyto shadery, jak název napovídá, provádí výpočty spojené s ray tracing metodou – sledování paprsku.

Post-processing Pomocí shaderů je možné implementovat také efekty, které jsou aplikované na celou obrazovku a ne jen na objekt.

Post-processing je proces aplikace tzv. full-screen efektů, respektive shaderů, na scénu před jeho zobrazením na obrazovce. Může například plošně přes celou scénu měnit barvy a pozici jednotlivých pixelů.

Kapitola 4

Realistické a nerealistické zobrazování

V oblasti počítačové grafiky je hojně využíváno různých realistických i nerealistických efektů. Obvykle je v počítačové grafice cílem dosáhnout co největší podobnosti se zobrazovanou realitou, ovšem v mnoha oblastech, jako jsou například počítačové hry, je často požadováno, aby byl daný obraz vykreslen nerealisticky, expresivně. Zpravidla se nerealistické efekty snaží imitovat různé výtvarné techniky, například tahy štětcem, náčrtky (zobrazeno na obrázku 4.1), zvýraznění obrysů, šrafování a mnoho dalších.

Obrázek 4.1: Příklad výtvarné techniky napodobitelné pomocí nerealistických efektů. Tento efekt napodobuje tahy tužkou či náčrtky. [1]

[1] Pencil draw house, autor: picrustable, 2017, dostupné na https://pixabay.com/vectors/pencil-draw-house-architecture-2825548/

4.1 Vybrané nerealistické efekty

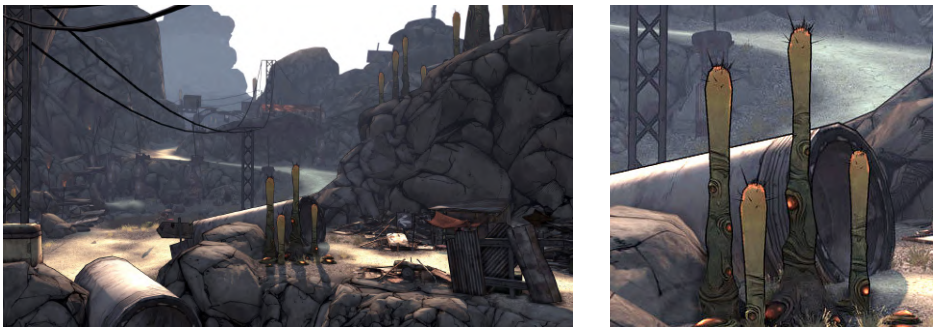
V této sekci je popsáno několik vybraných nerealistických efektů, které jsou také, pokud to bylo technicky možné, v rámci práce implementovány.

Některé algoritmy jsou přebrány z internetových zdrojů [2]. Konkrétní zdroje pro jednotlivé algoritmy jsou blíže popsány v kapitole 7.

4.1.1 Detekce hran

Detekce hran (anglicky *edge detection*) je jedním z důležitých nerealistických efektů. Může napodobovat například tzv. komiksový styl vykreslení scény, tahy tužkou či perem. Slouží také v herním průmyslu například k zobrazování aktivního objektu ve scéně pro lepší orientaci hráče.

Příkladem počítačové hry využívající detekci hran, je Borderlands [3] zobrazená na dvojici obrázků 4.2.



Obrázek 4.2: Snímky ze hry Borderlands využívající detekci hran. Pro dokreslení komiksového stylu se detekce hran v herním průmyslu používá poměrně často. Na výřezu ze snímku vpravo je dobře patrný efekt detekce hran.

Algoritmů pro detekci a vykreslení hran existuje mnoho. Některé pracují pouze s barvou jednotlivých bodů scény, některé využívají k detekci hran hloubku a normály.

Detekce hran pomocí hloubky a normál Algoritmus, převzatý z internetového zdroje [4], postupuje nejprve přes hloubku, poté přes normály. Prvním krokem je získání pozice čtyř okolních pixelů aktuálně zpracovávaného pixelu. Dále je spočtena hodnota hloubky v rozmezí 0 až 1 těchto čtyř okolních pixelů a poté jejich rozdíly. Rozdíly jsou spočteny mezi hloubkou spodního levého a horního pravého pixelu – zobrazeno na obrázku 4.3.

Tyto dva rozdíly poté reprezentují každý přibližně polovinu detekovaných hran – obrázek 4.4.

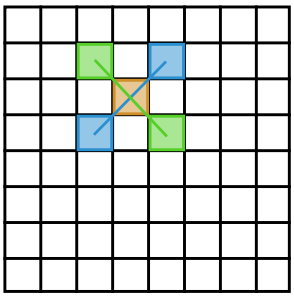
Dále je nutné tyto dva rozdíly spojit, to je provedeno pomocí tzn. Robbert's cross operátoru [7] – rozdíly diagonálně sousedících pixelů jsou nejprve umocněny na druhou, poté jsou tyto mocniny sečteny a následně je součet odmocněn. Posledním krokem je rozhodnutí, zda bude mít pixel černou, nebo bílou barvu, podle vhodně zvoleného prahu.

[2] https://www.shadertoy.com/,http://www.shaderslab.com/index.html
[3] https://store.steampowered.com/app/8980/Borderlands_Game_of_the_Year/
[4] Outline Shader, autor: Roystan, dostupné na https://roystan.net/articles/outline-shader/

Celý algoritmus vykreslení hran pomocí hloubky je popsán níže 1.

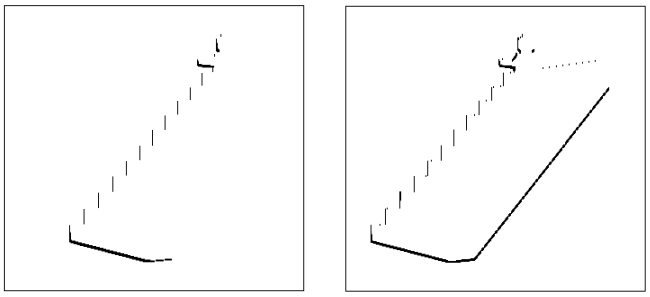
Input: $\mathcal{P}$ – pixely scény, T – vhodně zvolený práh pro hloubku
Input: *color* – výsledná barva pixelu
foreach $p \in \mathcal{P}$ **do**
  $\mathbf{N}_{bL} \leftarrow \text{getPixel}((p.x-1),(p.y-1))$
  $\mathbf{N}_{bR} \leftarrow \text{getPixel}((p.x+1),(p.y-1))$
  $\mathbf{N}_{uL} \leftarrow \text{getPixel}((p.x-1),(p.y+1))$
  $\mathbf{N}_{uR} \leftarrow \text{getPixel}((p.x+1),(p.y+1))$
  $\mathbf{D} \leftarrow \text{getDepth}(N)$
  $\mathbf{depthDiff}_1 \leftarrow \mathbf{D}_{uR} - \mathbf{D}_{bL}$
  $\mathbf{depthDiff}_2 \leftarrow \mathbf{D}_{uL} - \mathbf{D}_{bR}$
  $robbertsCross \leftarrow \sqrt{\mathbf{depthDiff}_1^2 + \mathbf{depthDiff}_2^2}$
  $color \leftarrow robbertsCross > T \ ? \ 1 : 0$

Algoritmus 1: Tento algoritmus využívá hloubky pro detekci hran. Funkce $\text{getPixel}(x, y)$ vrátí pixel na dané pozici *(x,y)* a funkce $\text{getDepth}(p)$ vrátí hloubku daného pixelu p v rozsahu 0 až 1. Algoritmus postupně prochází přes všechny pixely scény. Nejprve jsou do vektoru $\mathbf{N}$ uloženy pixely, které sousedí s právě zpracovávaným pixelem. Poté jsou zjištěny jejich hloubky v rozsahu 0 až 1. Dále jsou vypočteny rozdíly diagonálně sousedících pixelů. Posledním krokem je použití Robbert's cross operátoru a následné prahování výsledné hodnoty. Použitím tohoto algoritmu vzniká černý obrázek s bílými hranami objektů scény.

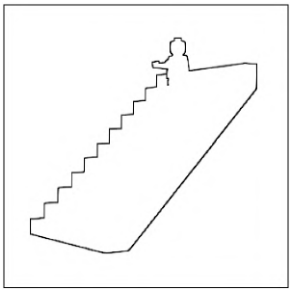


Obrázek 4.3: Rozdíl hloubky okolních pixelů aktuálně zpracovávaného pixelu. Okolní pixely jsou zobrazeny zelenou a modrou barvou, aktuálně zpracovávaný pixel oranžovou barvou. Je spočten rozdíl hloubky zelených pixelů a poté rozdíl hloubky modrých pixelů.

Pomocí hloubky jsou detekovány pouze vnější hrany objektu – na obrázku 4.5. Pro vykreslení vnitřních hran objektu je nutné využít normály.



Obrázek 4.4: Porovnání vykreslených hran pomocí rozdílů hloubek. Každý rozdíl reprezentuje přibližně polovinu vnějších hran. Vpravo je patrné, že se zobrazují více „pravé“ hrany, vlevo pak více „levé“ hrany objektu.

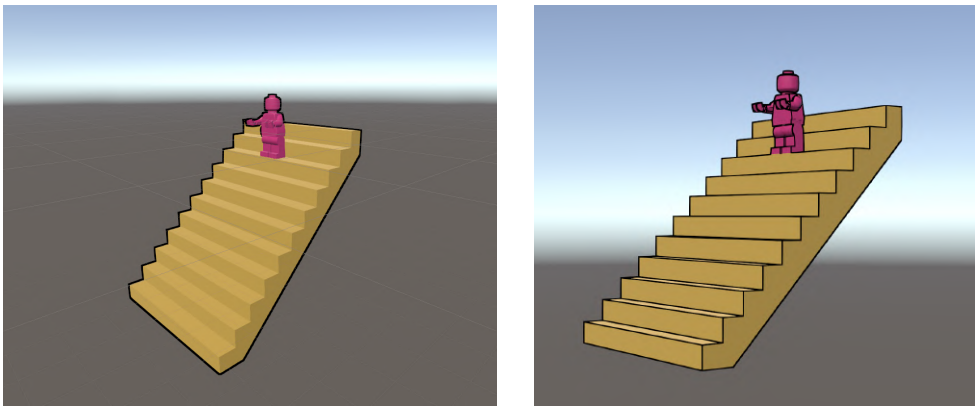


Obrázek 4.5: Pomocí hloubky jsou vykresleny pouze vnější obrysy objektu – zjednodušeně je vykreslena hrana pouze tam, kde se liší hloubka sousedících pixelů.

Postup vykreslení hran pomocí normál je velmi podobný postupu vykreslení hran pomocí hloubky. Jsou znovu využity okolní pixely aktuálně zpracovávaného pixelu. Jsou vypočteny normály těchto pixelů a poté jejich rozdíl. Namísto Robbert's cross operátoru je tentokrát využit skalární součin. Poté je, stejně jako u hloubky, rozhodnuto, zda bude mít pixel černou, nebo bílou barvu, na základě vhodně zvoleného prahu.

Po spojení detekce hran pomocí hloubky a normál je již možné vykreslit vnější i vnitřní hrany objektu, jako je zobrazeno na obrázku 4.6. Spojení probíhá tak, že se nejprve vybere větší ze dvou hodnot, tedy z hodnot barvy po prahování hloubky a normál – vznikne výsledná barva hrany. Poté je pomocí tzv. alpha blending spojena původní barva pixelu a výsledné barvy hrany. Princip alpha blending techniky je naznačen níže pomocí rovnic 4.1. Nejprve je vypočtena RGB hodnota výsledné barvy – *color* a poté výsledná alpha hodnota – *alpha*. Tyto dvě hodnoty jsou spojeny do výsledné barvy *result* – model RGBA.

$$\begin{aligned} color &= (top_{color} \cdot top_{alpha}) + (bottom_{color} \cdot (1 - top_{alpha})) \\ alpha &= top_{alpha} + (bottom_{alpha} \cdot (1 - top_{alpha})) \\ result &= (color, alpha) \end{aligned} \tag{4.1}$$



Obrázek 4.6: Obrázek vlevo zobrazuje hrany vykreslené pouze pomocí hloubky. Tímto způsobem vykreslení se zobrazí pouze vnější hrany objektů, což je patrné například na jednotlivých schodech. Na obrázku vpravo jsou pomocí hloubky i normál vykreslené všechny obrazové hrany – vnitřní i vnější.

Detekce hran pomocí Sobelova operátoru Sobelův operátor se využívá pro detekci horizontálních a vertikálních hran. K tomuto algoritmu jsou použity dvě masky (respektive konvoluční jádra) – pro osu x a pro osu y, obě jsou zobrazeny níže 4.2. Matice M_x je konvoluční jádro pro osu x, matice M_y konvoluční jádro pro osu y. Výsledný efekt je zobrazený na obrázku 4.7

$$M_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \quad M_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \tag{4.2}$$

Konvoluční jádra jsou aplikována na obraz, pixely a jim odpovídající elementy matice jsou vynásobeny. K tomu je využit tzv. *Hadamardův součin* [20]. Princip Hadamardova součinu je popsán rovnicí níže 4.3.

$$\begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} \circ \begin{pmatrix} b_1 & b_2 \\ b_3 & b_4 \end{pmatrix} = \begin{pmatrix} a_1 \cdot b_1 & a_2 \cdot b_2 \\ a_3 \cdot b_3 & a_4 \cdot b_4 \end{pmatrix} \tag{4.3}$$

Poté je vypočtena suma elementů výsledné matice. Výsledné číslo určuje, jak výrazná je na daném místě vertikální či horizontální hrana.

Pokud je dána matice elementů obrazu o velikosti 3×3 s následujícími hodnotami zobrazenými v matici 4.4, vyjde vypočtená hodnota po aplikaci konvolučního jádra pro osu x nulová – výpočet zobrazen níže 4.5. To značí, že v v matici elementů není žádná výrazná hrana.

$$\begin{pmatrix} 30 & 30 & 30 \\ 30 & 30 & 30 \\ 30 & 30 & 30 \end{pmatrix} \tag{4.4}$$

$$X = -30 + 0 + 30 - 60 + 0 + 60 - 30 + 0 + 30 = 0 \tag{4.5}$$

Pokud je naopak matice elementů obrazu zadána jako je zobrazeno v matici 4.6, vyjde vypočtená hodnota po aplikaci konvolučního jádra pro osu x velmi vysoká – zobrazeno níže 4.7. To značí, že v matici elementů je výrazná změna – hrana.

$$\begin{pmatrix} 5 & 30 & 100 \\ 3 & 20 & 80 \\ 2 & 10 & 90 \end{pmatrix} \tag{4.6}$$

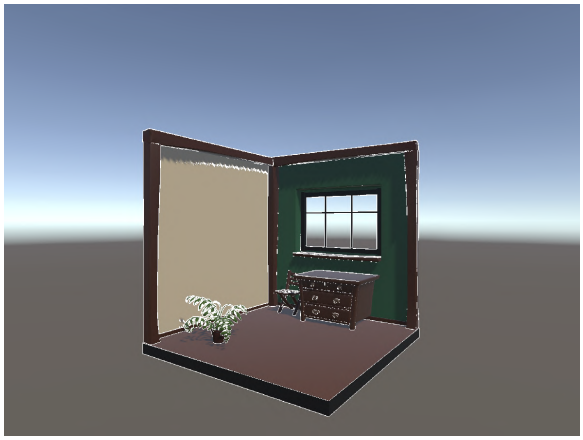
$$X = -5 + 0 + 100 - 6 + 0 + 160 - 2 + 0 + 90 = 337 \tag{4.7}$$

Po aplikaci detekce hran v obou směrech je použit vektor vypočtené hodnoty pro osu x a pro osu y. Z tohoto vektoru je následně vypočtena délka a tato délka je prahována.

Celý postup je naznačen níže v algoritmu 2.

Input: $\mathcal{P}_{\text{grayscale}}$ – pixely scény převedené do šedotónových odstínů, T – vhodně zvolený práh, $\mathbf{M^x}$ – konvoluční jádro pro osu x, $\mathbf{M^y}$ – konvoluční jádro pro osu y, m – velikost matice
Output: *color* - výsledná barva pixelu
foreach $p \in \mathcal{P}_{grayscale}$ **do**
  $G_x \leftarrow 0$
  $G_y \leftarrow 0$
  $i \leftarrow 0$
  for $i \leq m$ **do**
    $j \leftarrow 0$
    for $j \leq m$ **do**
      $G_x \leftarrow G_x + \mathbf{M^x}_{ij} \cdot \text{getPixel}(p_x + i, p_y + j)$
      $G_y \leftarrow G_y + \mathbf{M^y}_{ij} \cdot \text{getPixel}(p_x + i, p_y + j)$
      $j \leftarrow j + 1$
    $i \leftarrow i + 1$
  $f \leftarrow \sqrt{G_x^2 + G_y^2}$
  $val \leftarrow \max(f, T)$
  if $val > T$ **then**
    $color \leftarrow val$
  else
    $color \leftarrow \text{getColor}(p)$

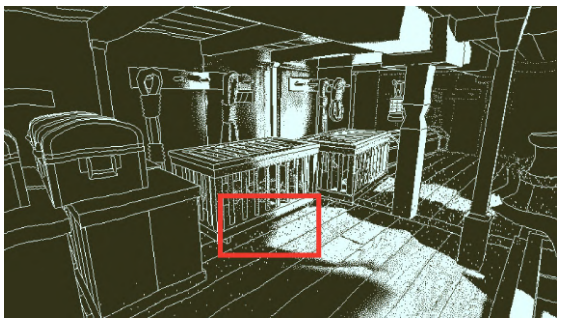
Algoritmus 2: Algoritmus pro detekci hran pomocí Sobelova operátoru. Funkce $\text{getPixel}(x, y)$ vrací pixel scény na pozici *(x,y)*, funkce $\max(a, b)$ vrací větší z hodnot *a,b* a funkce $\text{getColor}(p)$ vrací barvu pixelu *p*. Algoritmus postupuje přes všechny okolní body právě zpracovávaného bodu včetně daného bodu. Každý bod vynásobí s příslušnou hodnotou konvolučních jader, výsledky uloží do proměnných G_x a G_y. Následně tyto gradienty spojí – výpočet velikosti vektoru – a uloží do proměnné *f*. Posledním krokem je získání větší z hodnot f a T (prahu) a následné prahování.



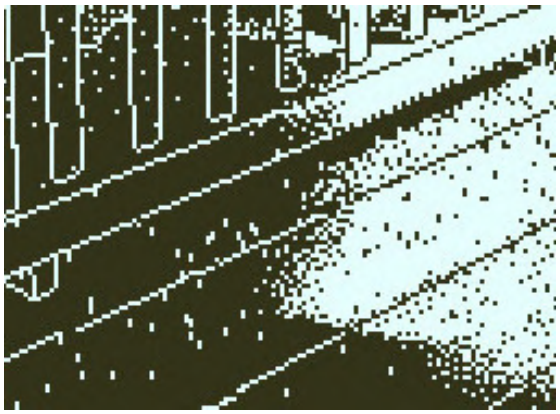
Obrázek 4.7: Implementovaný efekt detekce hran pomocí Sobelova operátoru.

4.1.2 Dithering

Dithering je jedním z algoritmů redukce šedotónových obrazů na černo-bílé. Algoritmus zachovává rozměry obrazu a snaží se zajistit maximálně odpovídající si podobu s původním obrázkem. Výsledný obraz se podobá například šumu CRT televize. I tento efekt je stejně jako detekce hran využíván například v herním průmyslu pro vykreslení atmosféry. Jednou z počítačových her, která využívá efektu podobného ditheringu, je Obra Dinn [5], efekt je patrný na obrázku 4.8 a výřezu z obrázku 4.9.



Obrázek 4.8: Snímek obrazovky ze hry Obra Dinn využívající efekt podobný ditheringu.



Obrázek 4.9: Na výřezu je dobře patrný efekt podobný ditheringu.

Nejjednodušším způsobem, jak implementovat dithering, je prahování. Pokročilejší technikou je pak metoda náhodného rozptýlení či maticový dithering. Algoritmy jsou uvedeny v knižním zdroji [24].

[5] "https://store.steampowered.com/app/653530/Return_of_the_Obra_Dinn/"

Prahování spočívá v nastavení vhodného prahu a rozdělení pixelů různých odstínů šedi na černé a bílé. Pixely s intenzitou šedé přesahující nastavený práh jsou nastaveny na černé a ostatní na bílé pixely.

Níže je prahování popsáno rovnicí 4.8. *P(x,y)* popisuje výslednou barvu pixelu na souřadnicích x, y, *I(x,y)* představuje intenzitu šedé barvy pixelu v původním obrazu na souřadnicích x, y a T představuje práh.

$$P(x,y) = \begin{cases} 1, & \text{pro } I(x,y) \geq T \\ 0, & \text{pro } I(x,y) < T \end{cases} \tag{4.8}$$

Náhodné rozptýlení je také metodou prahování, ovšem pro každý pixel je náhodně generován nový práh – popsáno vztahem 4.9. Může simulovat staré fotografie a videa či již zmíněný šum.

$$P(x,y) = \begin{cases} 1, & \text{pro } I(x,y) \geq rand() \\ 0, & \text{jinak} \end{cases} \tag{4.9}$$

Maticový dithering Maticový dithering neboli maticové rozptýlení pracuje na principu porovnání pixelů obrazu s odpovídajícími hodnotami distribuční (rozptylovací) matice a následné prahovaní. Postup je popsán pomocí algoritmu 3.

Používaných rozptylovacích matic je mnoho, některé z nich jsou naznačeny níže 4.10. [6] Používají se matice různých velikostí, časté jsou 4×4, nebo také 8×8.

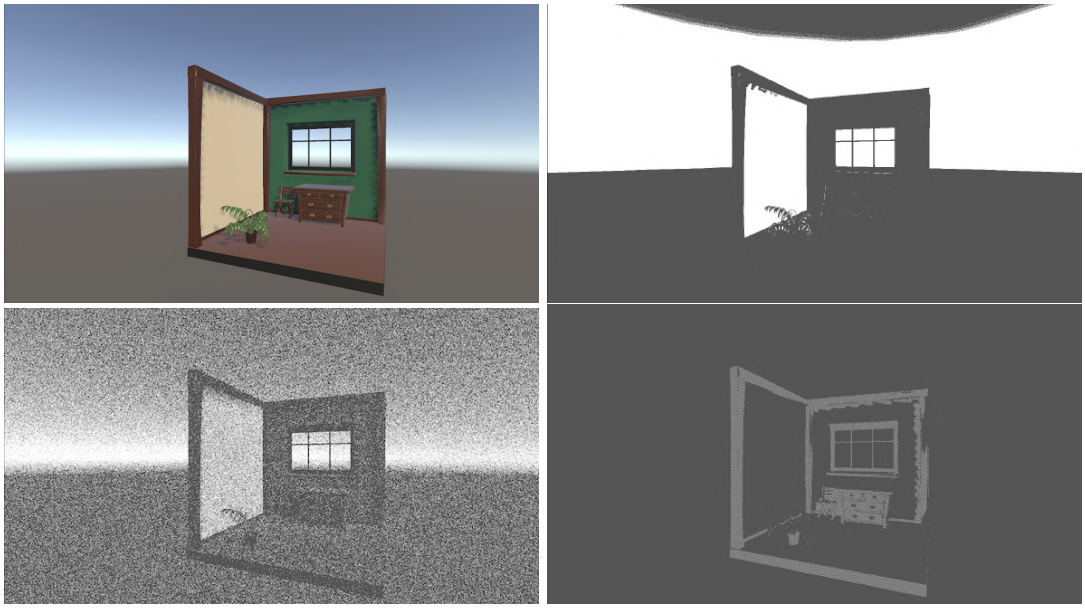
$$\begin{pmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{pmatrix} \begin{pmatrix} 1 & 5 & 9 & 2 \\ 8 & 12 & 13 & 6 \\ 4 & 15 & 14 & 10 \\ 0 & 11 & 7 & 3 \end{pmatrix} \tag{4.10}$$

Porovnání výsledků všech tří popsaných technik ditheringu je zobrazeno níže na čtveřici obrázků 4.10.

[6] První matice je převzata z internetového zdroje, dostupné na https://www.shadertoy.com/view/4lcyzn. Druhá matice je převzatá z materiálů předmětu IZG, https://www.fit.vut.cz/study/course/231044/.cs

Input: $\mathcal{P}_{\text{grayscale}}$ – pixely scény převedené do šedotónových odstínů,
$\mathbf{M}$ – rozptylovací matice, m – velikost strany matice
Output: *color* – výsledná barva pixelu
foreach $p \in \mathcal{P}_{grayscale}$ **do**
  $x \leftarrow$ getXPosition(p)
  $y \leftarrow$ getYPosition(p)
  $j \leftarrow x \mod m$
  $k \leftarrow y \mod m$
  $r \leftarrow$ getColor(p)
  if $r < \mathbf{M}_{jk}/(m \cdot m)$ **then**
    $color \leftarrow white$
  else
    $color \leftarrow black$

Algoritmus 3: Algoritmus maticového ditheringu. Funkce getXPosition(p), respektive getYPosition(p), vrací souřadnici daného pixelu p na ose x, respektive y, a funkce getColor(p) vrací barvu pixelu p. Postup je takový, že jsou nejprve vypočteny hodnoty j a k, které poté slouží k nalezení hodnoty pro porovnání v rozptylovací matici. Poté je už jen hodnota barvy pixelu p porovnána s hodnotou rozptylovací matice. Pokud je hodnota menší, zapíše se na místo pixelu p bílá barva; pokud je naopak hodnota větší, zapíše se černá barva.



Obrázek 4.10: Postupně: bez efektu, prahování, náhodné rozptýlení, maticový dithering. Při použití metody prahování vznikají ostré hrany a velké tmavé a bílé plochy. Metoda náhodného rozptýlení nejvíce připomíná šum CRT televize a zachovává velmi dobře viditelnost původního obrazu. U maticového ditheringu s použitou rozptylovací maticí vzniká tmavý obraz, který zachovává zejména okraje objektů scény.

4.1.3 Šrafování

Použitý algoritmus šrafování vytváří linky černé barvy po ploše celé obrazovky, přičemž zbytek obrazovky zůstává bílý – jak je patrné na obrázku 4.11. V oblastech scény s nižší intenzitou světla jsou vykreslené linky s větší frekvencí. Naopak v oblastech s vyšší intenzitou jsou linky vykreslené řídce, či vůbec. Rozdílná hustota vykreslených linek má za cíl napodobovat stínování tužkou. Algoritmus pracuje na způsobu vykreslení různě hustých linek podle intenzity dané oblasti. Nejprve zjistí, kolik šraf má v oblasti maximálně být, poté spočte vzdálenost od linky s určitou tloušťkou a prahuje ji. Podle prahování se do barvy pixelu uloží buď bílá, nebo černá barva.

Celý postup je naznačen níže v algoritmu 4.

Input: $\mathcal{P}$ – pixely scény, **LineDir** – vhodně zvolený směr linky šrafování, *steps* – maximální počet šraf na jednom úseku, *tresh* – práh pro vytvoření šrafy, *ratio* – konstanta určující poměr černé a bílé při šrafování, *width* – konstanta určující šířku výsledné šrafy

Output: *color* – výsledná barva pixelu

foreach $p \in \mathcal{P}$ **do**

- $col \leftarrow \text{getColor}(p)$
- $x \leftarrow \text{getXPosition}(p)$
- $y \leftarrow \text{getYPosition}(p)$
- $d \leftarrow \textbf{LineDir} \cdot\ p$
- $intensity \leftarrow \text{getIntensity}(col)$
- $s \leftarrow \lfloor \text{intensity} \cdot \text{steps} \rfloor / \ steps$
- $n \leftarrow ((width \cdot (1 - s) \cdot d)) \mod 1) < tresh + ratio \cdot s$
- $color \leftarrow$ n

Algoritmus 4: Algoritmus šrafování, kde funkce $\text{getColor}(p)$ vrací barvu pixelu p, funkce $\text{getXPosition}(p)$ (respektive $\text{getYPosition}(p)$) vrací souřadnici pixelu p na ose x (respektive y) a funkce $\text{getIntensity}(col)$ vrací intenzitu barvy *col*. Nejrpve je vypočtena vzdálenost d pixelu od linky pomocí skalárního součinu. Dále je vypočtena hustota šraf podle intenzity, maximální počet je *steps*. Dále je spočtena vzdálenost od linky s tloušťkou danou hodnotou *width*. Tato vzdálenost je poté prahována pomocí porovnání s hodnotou, která celkově určuje poměr bílé a černé barvy u šrafování. Do výsledné barvy pixelu se poté podle výsledku prahování uloží černá, nebo bílá barva.

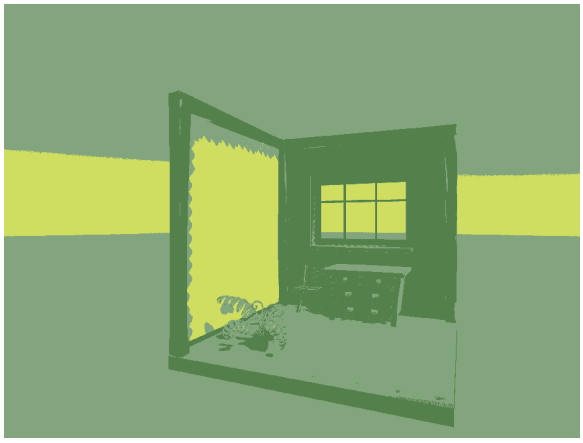
Obrázek 4.11: Použití metody šrafování. Na obrázku je vidět, že pozadí s nejvyšší intenzitou barvy je na některých místech dokonce bez linek šrafování. Na tmavších místech scény jsou naopak linky vykresleny ve vysoké hustotě.

4.1.4 Stínování pomocí barev

Stínování pomocí barev je velmi jednoduchý efekt pracující na základě porovnání hodnoty barevného kanálu s hodnotami prahu pro konkrétní barvy určené ke stínovaní. Původní pixel obrazu je nejprve převeden do šedotónového barevného prostoru a poté je jeho barva porovnána s danými prahy – jedná se tedy i o algoritmus redukce původních barev obrazu. Tento algoritmus nijak neupravuje pozici pixelu, pouze jeho barvu. Obraz tedy zůstává stejný, ale lze dosáhnout velmi zajímavého efektu použitelného například pro komiksový styl počítačových her. Postup této techniky je popsán níže v algoritmu 5.

Input: $\mathcal{P}$ – pixely scény, $\mathbf{T}$ – vektor vhodně zvolených prahů, $\mathbf{C}$ – tabulka uživatelských barev.
Output: *color* – výsledná barva pixelu
foreach $p \in \mathcal{P}$ **do**
  $col \leftarrow \text{getColor}(p)$
  $color \leftarrow \mathbf{C}_1$
  $i \leftarrow 1$
  for $i \leq |\mathbf{T}|$ **do**
    if $col_r > \mathbf{T}_i$ **then**
      $color \leftarrow \mathbf{C}_{i+1}$

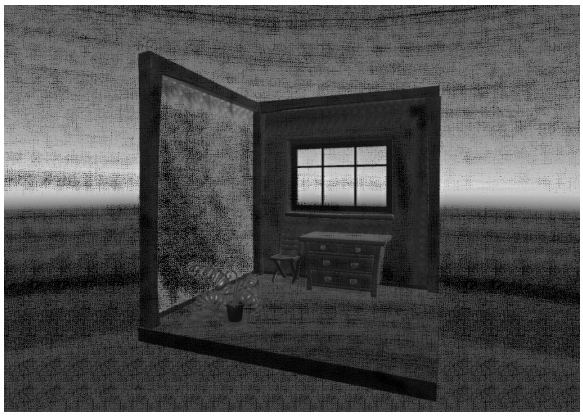
Algoritmus 5: Algoritmus pro stínování pomocí barev. Funkce getColor(p) vrací barvu právě zpracovávaného pixelu p. Hlavní částí algoritmu je cyklus, který postupuje přes všechny prahy uložené ve vektoru $\mathbf{T}$ a na základě porovnání s barvou pixelu p poté vybírá odpovídající barvu z tabulky uživatelských barev $\mathbf{C}$.



Obrázek 4.12: Použití metody stínování podle barev. V tomto případě jsou použity tři prahy, a tedy i tři různé barvy – konkrétně tři odstíny zelené. Světlejší části scény jsou vykresleny nejsvětlejší zelenou barvou, naopak nejtmavší části tou nejtmavší barvou. Vzniká tak efekt použitelný například k dokreslení komiksového stylu.

4.1.5 Efekt tahů tužky

Efekt tahů tužky napodobuje stínování tužkou – na obrázku 4.13. Jde o jeden ze složitějších algoritmů v této kapitole. Algoritmus pracuje na způsobu výpočtu extrému v daném pixelu pomocí derivací. Celý postup je naznačený v algoritmu níže 6



Obrázek 4.13: Použití metody tahů tužky.

Input: $\mathcal{P}$ – pixely scény, *angles* – vhodně zvolený počet úhlů, *step* – vhodně zvolený krok cyklu, *sensitivity* – vhodně zvolená citlivost, *tresh* – vhodně zvolený práh pro gradient, *range* – vhodně zvolený rozsah, *num* – vhodně zvolená hodnota pro podělení *weight*, aby nepřesáhla 1
Output: *color* – výsledná barva pixelu
foreach $p \in \mathcal{P}$ **do**
  $weight \leftarrow 1$
  $i \leftarrow 0$
  for $i <$ *angles* **do**
    $dir \leftarrow (1,0)$
    $dir \leftarrow \text{angle}(i)$
    $grad \leftarrow (\text{-dir}_y, \text{dir}_x)$
    $i \leftarrow i + 1$
    for $(j = -range) \leq range$ **do**
      $n \leftarrow \text{normalize}(dir)$ $pos \leftarrow \text{getPos}(p) + (b_x, b_y) \cdot 2$
      if *outOfBounds*(pos) **then**
        continue
      $g \leftarrow \text{getGrad}(pos)$
      if $\sqrt{g \cdot g} < tresh$ **then**
        continue
      $grad \leftarrow \text{normalize}(grad)$
      $g \leftarrow \text{normalize}(g)$
      $weight \leftarrow weight$ - $(|grad \cdot g|^{sensitivity})/num$
      $j \leftarrow j + step$
  $color \leftarrow \text{lerp}(black, background, weight)$

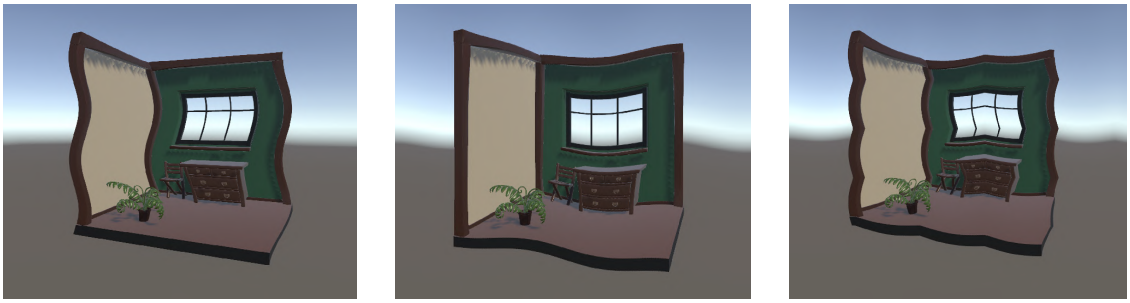
Algoritmus 6: Algoritmus efektu tahů tužky. Funkce normalize(dir) vrací normalizovací vektor vstupu *dir*, getPos(p) vrací pozici pixelu p, outOfBounds(pos) testuje, zda je pozice *pos* mimo zobrazovaný obsah, getGrad(pos) počítá směr největšího růstu pomocí derivací a funkce lerp$(start, end, t)$ je lineární interpolace. Lineární interpolaci popisuje knižní zdroj [24] a funkci *lerp* v nástroji Unity internetový zdroj [5].

4.1.6 Efekt zvlnění obrazu

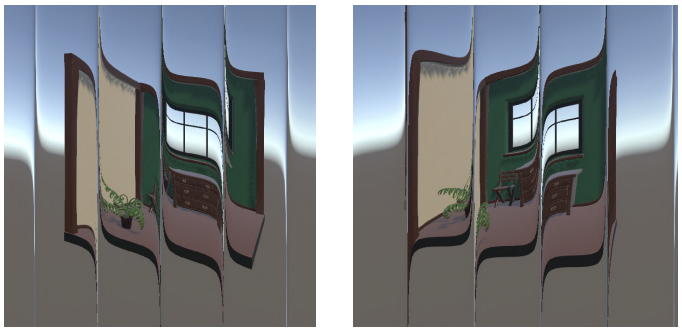
Efekt zvlnění obrazu pracuje na základě posunutí právě zpracovávaného pixelu o určitou vzdálenost. Respektive na místo právě zpracovávaného pixelu je uložen jiný pixel obrazu, což navozuje dojem změny pozice. Pro výpočet vzdálenosti lze využít různých periodických funkcí, například goniometrických. Pixel lze posunout v obou směrech – jak v ose x, tak y. Příklady funkcí jsou zobrazeny níže na obrázcích – sinus a cosinus na obrázcích 4.14, tangens a kontangens na obrázcích 4.15. Je možné využít také například obdélníkového průběhu signálu – na obrázcích 4.16. Obecný algoritmus je naznačen níže 7.

Input: $\mathcal{P}$ – pixely scény
foreach $p \in \mathcal{P}$ **do**
    $o_x \leftarrow \text{userFunctionX}(p)$
    $o_y \leftarrow \text{userFunctionY}(p)$
    $p_{xy} \leftarrow p_{xy} + o_{xy}$

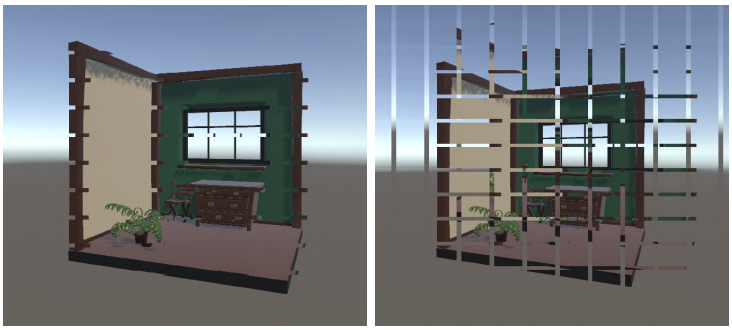
Algoritmus 7: Algoritmus pro zvlnění obrazu. Pomocí vybrané periodické funkcí $\text{userFunctionX}(p)$ a $\text{userFunctionY}(p)$ je vypočten posun ve směru osy x a osy y. Pokud je jeden z posunů konstantně nula, pak se pokřivení projevuje pouze v jednom směru. Následně se na pozici právě zpracovávaného pixelu zapíší nové souřadnice posunuté o vypočtené vzdálenosti ve směru osy x i y – $offsetX$ a $offsetY$.



Obrázek 4.14: Použití funkcí sinus a kosinus pro zvlnění obrazu. Funkce sinus je použita pro posun na ose x v levém obrázku, funkce kosinus pak pro posun na ose y na prostředním obrázku. Na pravém obrázku je využita funkce sinus v absolutní hodnotě v obou směrech – vytváří se efekt „obloučků“.



Obrázek 4.15: Použití funkcí tangens a kotangens pro zvlnění obrazu ve směru osy Y. Na levém obrázku je využita funkce tangens a na pravém obrázku funkce kotangens.



Obrázek 4.16: Použití periodického obdélníkového průběhu. Na levém obrázku je posun použit pouze ve směru osy x, na pravém obrázku je použit v obou směrech – vytváří se efekt čtverců.

4.1.7 Efekt pokřivení obrazu

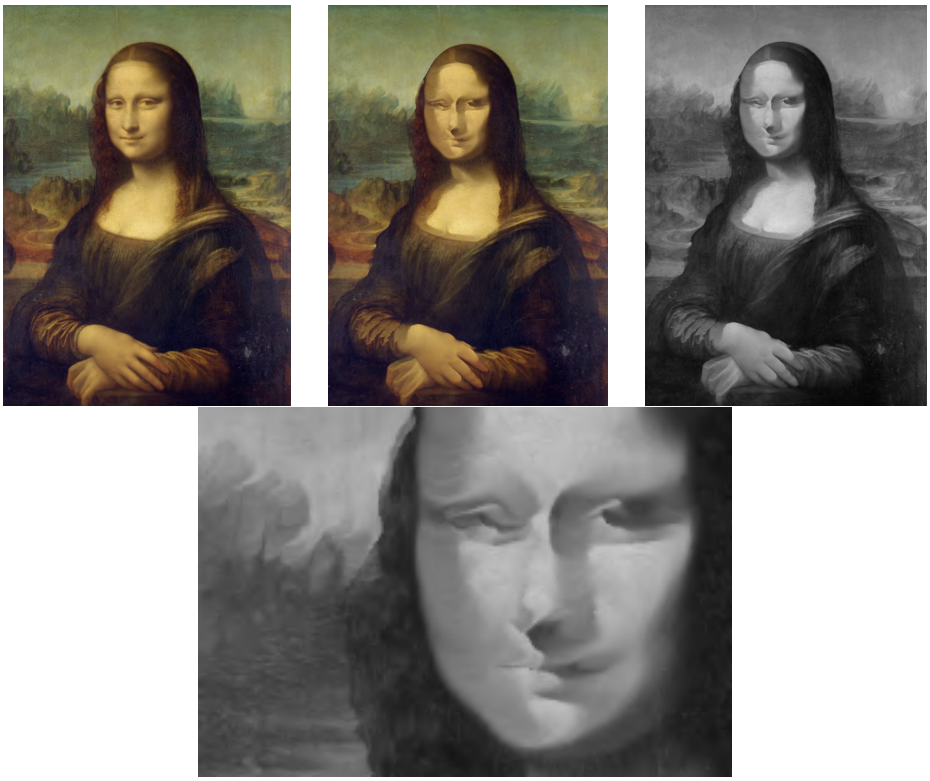
Metoda pokřivení obrazu je jedním z efektů, který navozuje dojem změny pozice právě zpracovávaného pixelu. V cyklu je provedeno několik „skoků“ od pozice právě zpracovávaného pixelu. Ke skoku je využito úhlu, který je vypočtený z intenzity barvy pixelu na pozici, kterou právě cyklus prochází. Výsledná pozice po konci cyklu je pozice pixelu, který bude uložen na místo původně zpracovávaného pixelu. Postup algoritmu je naznačen níže 8. Počet kroků cyklu a také další zvolené parametry algoritmu ovlivňují míru pokřivení obrazu.

Input: $\mathcal{P}$ – pixely scény, *steps* – vhodně zvolený počet kroků cyklu, R – vhodně zvolený úhel v radiánech, *radius* – vhodně zvolený poloměr
Output: *color* – výsledná barva pixelu
foreach $p \in \mathcal{P}$ **do**
  $pos \leftarrow \text{getPosition}(p)$
  $i \leftarrow 0$
  for $i < steps$ **do**
    $col \leftarrow \text{getColor}(pos)$
    $intensity \leftarrow \text{getIntensity}(col)$
    $angle \leftarrow \text{getRadians}(intensity \cdot R)$
    $pos \leftarrow pos + (\cos(angle), \sin(angle)) \cdot radius$
  $color \leftarrow \text{getColor}(pos)$

Algoritmus 8: Algoritmus pro pokřivení obrazu. Funkce $\text{getPosition}(p)$ vrací pozici pixelu p, $\text{getColor}(p)$ vrací barvu pixelu p, $\text{getIntensity}(col)$ vrací intenzitu barvy *col* a funkce $\text{getRadians}(x)$ převede dané číslo x do radiánů. Algoritmus nejprve získá původní pozici právě zpracovávaného pixelu *pos*. Následuje cyklus o daném počtu kroků *steps*. V tomto cyklu je nejprve zjištěna barva pixelu na dané pozici *pos* a její intenzita *intensity*. Poté je spočten úhel *angle* v radiánech pomocí dané konstanty R. Do pozice *pos* je poté uložena nová pozice posunutá o $\cos(angle)$ ve směru osy x a o $\sin(angle)$ ve směru osy y vynásobené o zvolený poloměr zvlnění.

Na obrázcích 4.17 níže je zobrazen původní obraz[7] a poté použití efektu zvlnění obrazu jak v barevné, tak v šedotónové variantě.

[7] Zdroj původního obrázku: Leonardo da Vinci, Public domain, via Wikimedia Commons, dostupné na https://commons.wikimedia.org/wiki/File:Mona_Lisa,_by_Leonardo_da_Vinci,_from_C2RMF_retouched.jpg



Obrázek 4.17: Použití metody pokřivení obrazu. První obrázek je původní obraz bez využití efektu. Na druhém obrázku je již aplikovaný efekt a na třetím je obraz s efektem převeden do šedotónových odstínů. Poslední obrázek je výřez, na kterém je efekt dobře patrný. Efekt zvlnění obrazu je dobře patrný na scénách s vysokou mírou detailu.

Kapitola 5

Unity

Unity je multiplatformní herní engine a IDE vyvinutý Unity Technologies pro vývoj her na počítače, konzole, mobilní zařízení a web. Zpočátku sloužil zejména pro menší vývojáře. První verze Unity vyšla v roce 2005 pouze pro Mac OS X, podpora pro vývoj aplikací na Windows byla přidána později. Po zveřejnění iPhone v roce 2007 byla vydána i verze Unity pro vývoj mobilních aplikací na iPhone. [9]

Projekt v Unity běžně obsahuje jednu či více scén. Scény se skládají z objektů – tzv.„Game Objects“. Objekty lze ve scéně posouvat a otáčet jak pomocí samotných číselných souřadnic, tak pomocí posuvníků přímo ve scéně. Tyto objekty mohou být jak jednoduché geometrické tvary, tak například součásti uživatelského rozhraní vytvářené aplikace. Hlavním objektem většiny aplikací v Unity je jedna či více kamer, tyto kamery nastavují svým natočením, pozicí a úhlem pohledu, co bude ve výsledné aplikaci ze scény vidět. Další důležitou částí je osvětlení scény, kterému lze nastavovat barvu, pozici, otočení, délku a šířku světelného kuželu a také mimo jiné to, jak ostré jsou ve scéně stíny, které světlo vytváří.

Většině objektů lze nastavovat vzhled pomocí materiálů. Materiály mohou aplikovat na objekt barvu, texturu či použít přiložený shader, který mění vzhled objektu pomocí uživatelem napsaného kódu – ke změně vzhledu jednoho objektu přes materiál se často používají níže popsané *Unlit* shadery. K objektům lze také připojovat skripty. Skripty, napsané nejčastěji v jazyce C#, mohou měnit různé vlastnosti či parametry objektu. Mohou objektem například rotovat – to lze využít pro animace rotací objektu či otáčení kamery, a tedy i pohledu hráče. Pro objekty uživatelského prostředí lze pomocí skriptů nastavit chování aplikace po stisku tlačítka. V neposlední řadě lze skripty aplikované na kameru využít pro změnu vzhledu celé scény – tzv. full-screen efekty, neboli post-processing.

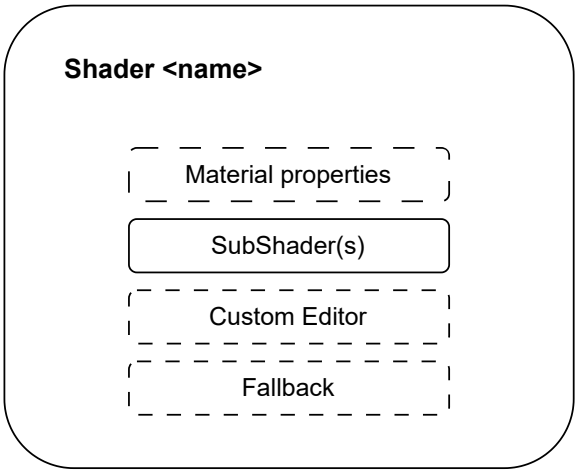
Aplikace z Unity lze exportovat na mnoho zařízení – počítače s různými operačními systémy, mobilní zařízení a také holografický tablet Lume Pad. Pro vývoj na této na cílové platformě práce, tabletu Lume pad, je třeba využít oficiálního SDK pro vývoj aplikací na Lume Pad od společnosti Leia Inc. [3]

Shadery v Unity Shadery v Unity stejně jako v jiných nástrojích mohou měnit vzhled scény či jednotlivých objektů. S Shadery se v Unity se často pracuje pomocí tzv. „Shader objects“ popsaných níže.

Shader object, ShaderLab, SubShader a Pass Shader object je instance Shader třídy. Jde o způsob, jak v Unity pracovat se shadery. Shader object reprezentuje „obálku“ pro shader programy a další informace, například o nastavení na GPU. Umožňuje definovat více shader programů v jednom souboru a říká Unity, jak je používat. Unity přináší dva

způsoby vytváření Shader objects. Prvním z nich je textový soubor s příponou *.shader* obsahující kód, druhým novějším způsobem jsou tzv. „Shader Graphs“. [1] Shader object se v kódu defininuje pomocí speciálního jazyka, tzv. ShaderLab. Základní struktura Shader object je naznačena níže 5.1.

Informace, které Shader object obsahuje, jsou obsažené v jeho strukturách SubShader a Pass. SubShader umožňuje rozdělit Shader object na části kompatibilní s různým hardwarem, či zobrazovacím řetězcem. SubShader obsahuje jeden či více Pass bloků. Pass bloky obsahují již samotný shader program a jiné nastavení.



Obrázek 5.1: Základní struktura Shader object. Převzato z [5]. Přerušovanou čarou jsou označeny nepovinné části Shader Object. „Name“ označuje název shaderu, tedy i to, jak je samotný soubor s příponou .shader pojmenovaný. Dále nepovinný blok „Material properties“, ve kterém lze definovat proměnné, které se zobrazí v Unity editoru. Slouží například pro změnu barev bez explicitních změn v kódu. Poté následuje jeden či více SubShader bloků. Jako poslední dvě nepovinné části struktura obsahuje tzv. „custom editor“ a „fallback“. „Custom editor“ slouží k nastavení vlastního zobrazení shaderu v Unity editoru, pokud je například třeba využít typ, který Unity editor automaticky nenabízí. „Fallback shader“ definuje, který shader má Unity využít, pokud nemůže využít ani jeden ze SubShaderů. Tento problém typicky nastává z důvodu, že grafický hardware daný SubShader nepodporuje.

Typy shaderů v Unity V Unity existuje několik základních typů shaderů:

- Standard Surface Shader – umožňuje měnit vzhled konkrétních objektů scény, na které je shader aplikován,
- Unlit Shader – funguje stejně jako Standard Surface Shader, ale neobsahuje žádné výpočty světla, tento typ shaderu je jako příklad zobrazený v kódu v příloze 9,
- Image Effect Shader – slouží pro změnu vzhledu celé scény (tzv. full-screen efekty), tedy post-processingu,
- Compute Shader – popsáno v sekci 3.6.1.

[1] Shader Graph umožňuje implementaci shaderů vizuálně pomocí grafů. Vytváří abstrakci nad kódem a jde tedy o uživatelsky přívětivou metodu implementace.

Jazyky pro implementaci SubShaderů v Unity V Unity lze pro definici SubShader bloků využít programovací jazyk HLSL – *High-Level Shading Language*. V kódu se označuje značkami *HLSLPROGRAM* a *ENDHLSL*. HLSL byl vyvinutý společností Microsoft a je analogií pro jazyk GLSL použitý v OpenGL. HLSL je velmi podobný jazyku Cg (*C for graphics*). Cg byl vyvinutý společností NVIDIA a v kódu se označuje *CGPROGRAM* a *ENDCG*

Post-processing V Unity lze post-processing využít po importování balíčku „Post Processing“. Dále je potřeba například vytvořit a nastavit vrstvu pro post-processing. Pro implementaci post-processing efektů se využívá *Image Effect* shaderů. Unity nabízí také několik zabudovaných post-processing efektů.

Zobrazovací řetezěc v Unity V nástroji Unity jsou využívány čtyři druhy zobrazovacích řetězců, tzn. render pipelines. Prvním je Built-in Render Pipeline. Tento řetězec je obecně využitelný, ale nabízí pouze limitované možnosti vlastního přizpůsobení. Dalším druhem je Universal Render Pipeline (URP), tento zobrazovací řetězec je skriptovatelný, programovatelný, ale stále jednoduchý a rychlý pro vlastní přizpůsobení. Současně je použitelný na velkém množství platforem. Třetí možností je využít High Definition Render Pipeline (HDRP). Je také skriptovatelná, ale využívaná zejména na složitější aplikace na profesionálnějších platformách. Poslední možností je vytvořit si celý vlastní zobrazovací řetězec.

Informace pro text kapitoly 5 jsou přebrány z oficiální dokumentace Unity [5].

Kapitola 6

Návrh

Návrh aplikace obnáší několik důležitých částí. Nejprve je třeba vybrat nástroj pro implementaci a navrhnout celkovou strukturu aplikace – jak zobrazovat efekty, dále navrhnout uživatelské rozhraní. Důležitou částí návrhu je také sestavit testy pro následné testování.

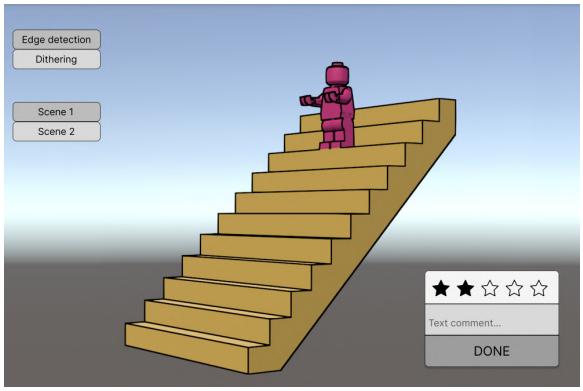
6.1 Aplikace a použité technologie

Aplikace má sloužit k pohodlnému prohlížení jednotlivých efektů a následnému testování, zda je možné nerealistické efekty implementovat stejně kvalitně ve 2D i 3D režimu.

Z toho plyne, že musí obsahovat scénu, kterou si bude možné dobře prohlédnout – k tomu slouží rotace kamery ovládaná uživatelem a animace otáčení některých objektů scény. Scéna by měla obsahovat jak větší plochy s malým množstvím detailu, tak menší plochy s velkým množstvím detailu. Tak, aby si uživatel mohl prohlédnout efekty na různě detailních objektech.

Uživatelské rozhraní aplikace by mělo být intuitivní a jednoduché tak, aby bylo možné testování provést také s méně technicky zkušenými uživateli. Musí obsahovat tlačítka pro ovládání aplikovaných efektů, dále tlačítka k ovládání scény.

Součástí návrhu aplikace je také formulář pro hodnocení efektu při testování. Ten obsahuje hvězdičky pro hodnocení a pole na slovní komentář. Návrh aplikace v nástroji Figma je zobrazen na obrázku 6.1.



Obrázek 6.1: Návrh výsledné aplikace v nástroji Figma. V levé části aplikace se nachází výběr scény a efektu. V pravé části pak formulář na hodnocení efektu obsahující hvězdičky a také pole na slovní komentář.

Jako nástroj pro implementaci aplikace byl zvolen nástroj Unity. Důvodem je existence oficiálního SDK od společnosti Leia Inc., které slouží pro vývoj aplikací na tablet Lume Pad – popsáno v sekci 6.1.1.

6.1.1 Lume Pad

Po instalaci SDK pro Lume Pad do projektu v Unity je mezi objekty scény přidán objekt *Leia Display* a k hlavní kameře scény je přidána komponenta *Leia Camera*. Tyto dvě přidané části umožňují ovládání nastavení tabletu Lume Pad.

Objekt scény *Leia Display* nastavuje pomocí zaškrtávacího políčka v editoru, zda je ve vytvářené aplikaci zapnutý, či vypnutý 3D, light field, režim.

Komponenta *Leia Camera* obsahuje číselné nastavení dvou hodnot důležitých pro zobrazení ve 3D režimu – *Baseline Scaling* a *Convergence Distance*.

Baseline Scaling Tato hodnota slouží k nastavení hloubky 3D scény na displeji, konkrétně nastavuje vzdálenost mezi kamerami pro zobrazení scény ve 3D. Při špatně nastavených hodnotách *Baseline Scaling* se mohou některým uživatelům jevit objekty scény dvojitě a způsobit nepohodlný pohled na scénu.

Convergence Distance Tato hodnota nastavuje umístění, vzdálenost, roviny konvergence z více pohledů light field technologie vzhledem k rovině kamery.

6.1.2 Návrh jednotlivých efektů

Efekty jsou navrhnuty jako tzv. full-screen efekty, tedy aplikované přes celou obrazovku. Pro full-screen efekty (tedy post-processing) slouží v Unity „Image Effect Shader“. Aby bylo možné post-processing efekty využít, nabízí Unity také „PostProcessing“ balíček, který obsahuje všechny potřebné nástroje. Níže je popsán návrh jednotlivých efektů.

Maticový dithering Pro maticový dithering je nejprve potřeba převést právě zpracovávaný pixel do šedotónových odstínů. To je provedeno níže uvedeným převodem 6.1 převzatým z internetového zdroje [1]. Vypočtená hodnota *intensity* se poté využije pro všechny složky barvy pixelu.

Pro prahování je dle algoritmu využito matice níže 6.2. Matice a podmínka využitá k prahování je převzata z internetového zdroje [2].

$$intensity = C_r \cdot 0.3 + C_g \cdot 0.59 + C_b \cdot 0.11 \tag{6.1}$$

$$\begin{pmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{pmatrix} \tag{6.2}$$

[1] Autorem je Alan Zucconi, dostupné na https://www.alanzucconi.com/2015/07/08/screen-shaders-and-postprocessing-effects-in-unity3d/

[2] *Ordered dithering*, autor je *luka712*, dostupné na https://www.shadertoy.com/view/4lcyzn

Náhodný dithering U náhodného rozptýlení je také třeba převést pixel do šedé barvy. Poté dochází k prahování na černou, nebo bílou s náhodně nastaveným prahem. Práh je vypočten pomocí funkce **rand***(seed)*. Tato funkce vrací náhodné číslo v rozsahu 0 až 1 podle zadané hodnoty *seed*. Funkce je přejatá z internetového zdroje [3]. Pro generování dostatečně náhodného vzoru, který připomíná šum, byla po testování jako *seed* použita hodnota z výrazu v rovnici 6.3. Ve vztahu pro *seed* je *pos* pozice právě zpracovávaného pixelu. K pozici je přičtena znovu vygenerovaná náhodná hodnota pomocí funkce **rand**, tentokrát je jako *seed* použita pozice posunutá ve směru osy x i y. Po použití pouze *pos* jako *seed* jsou generovány pravidelné pruhy, což šum nepřipomíná.

$$pos + rand((pos.x + 2, i.pos + 2)) \tag{6.3}$$

Prahování I u prahování je třeba převést pixel do odstínu šedé barvy. Tato barva je poté prahována na černou, nebo bílou s konkrétním prahem 0,1.

Detekce hran K detekci hran je využitý Sobelův operátor, jako je popsáno v kapitole 4. Kód je inspirovaný internetovým zdrojem [4].

Detekce hran – černobílý efekt Tento efekt funguje na stejném principu jako předchozí detekce hran. Rozdíl je pouze v tom, že zatímco v klasické detekci hran zůstává obrázek na pozadí stejný, zde je převeden do šedotónových odstínů.

Šrafování Kód je sestaven dle algoritmu šrafování v sekci 4. Pro výpočet skalárního součinu je využita dostupná funkce *dot*. Kód je převzatý z internetového zdroje [5].

Barevné stínování Algoritmus je stejný jako v kapitole 4. Jsou použity tři odstíny zelené barvy (zobrazeny na trojici obrázků 6.2) a jako prahové hodnoty 0,1 a 0,5. Kód je inspirovaný shaderem z internetového zdroje [6].

Obrázek 6.2: Tři odstíny zelené barvy použité pro stínování pomocí barev. Efekt má redukcí barev navozovat komiksový styl.

Efekt tahů tužky Efekt tahů tužky je jedním ze složitějších algoritmů, a tedy i shaderu. Algoritmus je zapsán stejně, jako je popsáno v teoretické části práce 4. Kód je přejatý z internetového zdroje [7].

[3] *Generative designs* , autory jsou Patricio Gonzalez Vivo a Jen Lowe, dostupné na https://thebookofshaders.com/10/

[4] Kód v jazyce MATLAB, autorem je Aaron Angel, dostupné na: https://www.imageeprocessing.com/2011/12/sobel-edge-detection.html

[5] Autorem je Ing. Tomáš Milet, Ph.D., shader je dostupný na: https://www.shadertoy.com/view/cddGRX

[6] Game Boy effect, dostupné na http://www.shaderslab.com/demo-98---game-boy-effect.html

[7] Pencil Effect, dostupné na http://www.shaderslab.com/demo-99---pencil-effect-1.html

„Wobble“ Pro efekt zvlnění obrazu je jako uživatelská funkce (*userFunctionX* z algoritmu v kapitole 4) pro posun ve směru osy x využita funkce sinus. Ve směru osy y žádný posun není. Amplituda periodické funcke sinus je nastavena na hodnotu 0,005. Shader je implementován za pomoci internetového zdroje [8].

„Wobble carving“ Efekt pokřivení obrazu je implementován podle algoritmu uvedeným v kapitole 4. Rozdíl je pouze ten, že tělo vnitřního cyklu algoritmu je zapouzdřeno do vlastní funkce *jump*. Algoritmus byl použit v diplomové práci [14] a je uveden rovněž v internetovém zdroji [9].

6.2 Návrh testů

V rámci testování je třeba zjistit, jak kvalitně a přívětivě je možné implementovat nerealistické efekty na cílovou platformu Lume Pad. Testování by tedy mělo zajistit data o kvalitě prostorového dojmu s aplikovanými efekty, přívětivosti efektu a také komfortu pohledu na efekt. Pro porovnání efektu ve 2D a 3D by uživatelé měli hodnotit přívětivost jak ve 2D, tak ve 3D režimu. Hodnocená kritéria testování jsou popsána v sekci 6.2.1. Je třeba navrhnout rovněž testovací scénář, tedy jak bude samotné testování s uživateli probíhat. To je popsáno v sekci 6.2.2.

6.2.1 Hodnocená kritéria

Uživatelé tedy testují následující skutečnosti:

- Preferovaná hodnota „Convergance Distance“.
- Preferovaná hodnota „Baseline Scaling“.
- Viditelnost 3D jednotlivých efektů – tedy zda aplikovaný nerealistický efekt zlepšuje, či zhoršuje prostorový dojem.
- Přívětivost jednotlivých efektů ve 3D – jak se uživateli efekt líbí ve 3D režimu.
- Přívětivost jednotlivých efektů ve 2D – jak se uživateli efekt líbí ve 2D režimu.
- Komfortnost pohledu na efekt ve 3D – zda je pro uživatele pohodlné se na efekt dívat delší dobu.
- Nejlépe a nejhůře hodnocený efekt – na konci testování uživatel zhodnotí, který efekt se mu líbil nejvíce a který nejméně.
- Prostorový dojem samotné scény bez efektu – zda je pro uživatele 3D viditelné v tabletu dostatečně, či ne.
- Komfort pohledu na scénu bez efektu – zda je pro uživatele pohodlné se na tablet ve 3D režimu dívat delší dobu.

[8] Simple Wobble, dostupné na https://www.shadertoy.com/view/MdS3RV
[9] Wobble Carving Effect, dostupné na https://www.shadertoy.com/view/ds3GD8

6.2.2 Testovací scénář

Po vysvětlení ovládání aplikace jsou uživatelé požádání o prohlédnutí si jednotlivých efektů, jak ve 3D, tak ve 2D režimu. Dále otestují a vyberou preferované hodnoty „convergance distance“ a „baseline scaling“, tyto vybrané hodnoty mohou v průběhu testování dále měnit. Poté prochází postupně jednotlivé efekty a hodnotí je pomocí čtyř kritérií – viditelnost 3D, přívětivost ve 3D a 2D a komfortnost pohledu na efekt ve 3D. Kritéria jsou všechna hodnocena na škále od 0 do 5, kde 0 je nejhorší známka a 5 nejlepší. V případě viditelnosti 3D je stupnice následující:

- 0 – aplikací efektu prostorový dojem zcela zmizí,
- 1 – aplikací efektu se prostorový dojem zhorší,
- 2 – aplikací efektu prostorový dojem zůstane téměř stejný,
- 3 – aplikací efektu prostorový dojem zůstane stejný,
- 4 – aplikací efektu se prostorový dojem zlepší,
- 5 – aplikací efektu se prostorový dojem výrazně zlepší.

Následně vyberou jimi nejlépe a nejhůře hodnocený efekt. Celý proces testování uživatelé komentují, některé zajímavé poznatky jsou zapsány ve výsledcích testování. Data z testování jsou následně vyhodnocena.

Kapitola 7

Implementace

Výsledná aplikace je implementována pomocí nástroje Unity a používá Built-in Render Pipeline. Rozdíly oproti návrhu a výsledné uživatelské rozhraní jsou popsány v sekci 7.1. Cílová platforma je holografický tablet Lume Pad (popsáno v sekci 7.2). Jednotlivé efekty jsou implementovány pomocí pomocí shaderů. Na obrázcích 7.1 a 7.2 je zobrazena výsledná aplikace se skrytým a zobrazeným uživatelským rozhraním.

Obrázek 7.1: Snímek výsledné aplikace při zobrazeném uživatelském rozhraní. V levé části aplikace obsahuje zaškrtávací tlačítka k zapnutí efektů a ve spodní části pak slidery k ovládání hodnot tabletu Lume Pad. V pravé části aplikace obsahuje tlačítko pro vypnutí a zapnutí 3D režimu a tlačítko pro zapnutí a vypnutí rotace objektů.

7.1 Rozdíly oproti návrhu a výsledné GUI

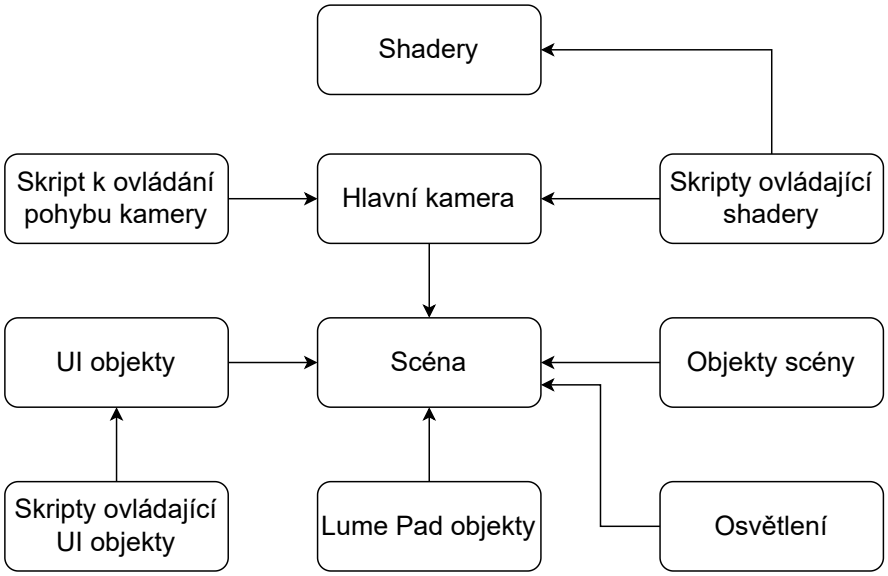
Výsledná aplikace se od návrhu liší v několika skutečnostech. Obsahuje pouze jednu složitější scénu na rozdíl od několika scén na výběr. Jedna složitější scéna je pro účely testování dostačující. Dále aplikace neobsahuje formulář pro hodnocení. Od formuláře je upuštěno z důvodu zbytečně složitého uživatelského rozhraní v aplikaci. Při návrhu testovacího scénáře bylo zjištěno, že by do formuláře bylo třeba zakomponovat příliš mnoho otázek. To by ubíralo na přehlednosti a uživatelské přívětivosti aplikace. Oproti návrhu je přidáno do uživatelského rozhraní také tlačítko k přepnutí se z 3D do 2D módu. Důvodem je možnost porovnání efektů ve 2D a 3D. Dalším rozdílem je přidání slideru k nastavení tzv. „convergence distance“ a „baseline scaling“. Další částí GUI a také rozdílem oproti návrhu je tlačítko na skrytí veškerých tlačítek na obrazovce. Skrytí uživatelského rozhraní slouží

Obrázek 7.2: Snímek výsledné aplikace při skrytém uživatelském rozhraní. V tomto případě aplikace obsahuje pouze scénu a umožňuje otáčení kamery uživatelem.

k lepšímu prohlédnutí si efektu. Toto tlačítko při skrytí umožní otáčení kamery, při znovu zobrazení uživatelského rozhraní naopak možnost otáčení kamery znemožní. Aplikace kromě otáčení kamery umožňuje také pomocí tlačítka zapnout a vypnout animaci otáčení několika objektů scény.

7.2 Struktura aplikace

Struktura aplikace, tedy i projektu v nástroji Unity, je zobrazena na obrázku níže 7.3.



Obrázek 7.3: Struktura práce. Ve středu diagramu je scéna aplikace, součástí scény jsou objekty (*Game Objects*), objekty uživatelské rozhraní (*UI*), hlavní kamera, osvětlení a objekty tabletu Lume Pad. Na objekty uživatelského rozhraní jsou napojeny skripty, které je ovládají. K hlavní kameře jsou připojeny skripty k ovládání kamery a dále skripty ovládající post-processing efekty. Na tyto skripty jsou napojeny samotné shadery realizující nerealistické efekty.

7.2.1 Objekty scény

Scéna obsahuje několik základních objektů Unity – rovina vytvořená pomocí *Cube* a cedule vytvořená pomocí *Cube* a *Plane*. Pro vytvoření „hráče“, který slouží pro zapouzdření kamery z důvodu možnosti otáčení, slouží objekt *Capsule*. Dále scéna obsahuje pokoj, obraz a dvě figurky.

Pokoj je vytvořen pomocí zdarma přístupného demo modelu pokoje *Cube Diorama* z oficiálních webových stránek nástroje Blender, který slouží k vytváření 3D počítačové grafiky. [1]

Figurky jsou převzaty z oficiálního výukového projektu Unity Learn – *Lego Microgame*, která slouží pro seznámení se s nástrojem Unity a obsahuje mimo jiné Lego modely. [2]

Obraz je vytvořen jako základní objekt *Plane* s aplikovanou texturou.

Materiály a textury Objekty mají na svém povrchu aplikovány materiály s texturami pro změnu vzhledu podle potřeby. Některé použité materiály jsou základními Unity materiály s texturou jedné barvy – tyto materiály jsou použity figurkách. Pokoj vytvořený z modelu nástroje Blender obsahuje vlastní materiály a textury, nejsou vytvořeny pomocí nástroje Unity. Spodní část cedule a také rovina pod objekty jsou obarveny texturami ze zdarma přístupného balíčku textur *Lowpoly Textures Pack*, který je dostupný na Unity Asset Store. [3] Další použitou texturou je obrázek dostupný na webových stránkách [4], který je použitý pro vytvoření obrazu na stěně. Pro obličej figurek je využit obrázek z internetového zdroje[5]. Poslední použitou texturou je obrázek *Leia Loft SDK*, který je dostupný po importu Lume Pad SDK od společnosti Leia Inc. [6]

Diagram 7.4 níže zobrazuje, z jakých částí je výsledná scéna aplikace složena.

7.2.2 UI objekty scény

GUI je vytvořeno pomocí „UI game objects“ (objekty uživatelského prostředí) v Unity. Většinově se jedná o zaškrtávací políčka – tzv. „toggle“. Pro výběr hodnot convergence distance a baseline scaling je použit *slider*. Zaškrtávací políčka i slidery jsou napojeny na odpovídající skripty blíže popsané v podkapitole 7.2.3. Aplikace umožňuje také pohyb kamery při skrytém uživatelském rozhraní.

7.2.3 Shadery a skripty

K implementaci shaderů byl využit jazyk CG a k implementaci skriptů jazyk C#. Jednotlivé soubory aplikace tvoří z větší části dvojice. Dvojici tvoří shader a k němu odpovídající skript, dále zaškrtávací políčko a jeho chování ovládající skript. Skripty pro ovládání pohybu kamery a animace objektů netvoří dvojici s dalším souborem.

[1] *Cube Diorama* z oficiálních stránek nástroje Blenderu, dostupné na https://www.blender.org/download/demo-files/

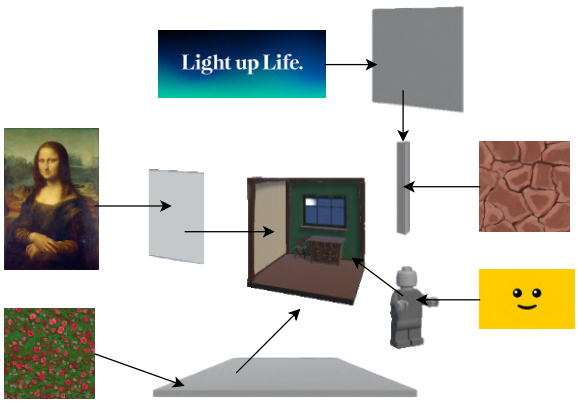
[2] *Lego Microgame* z oficiálních stránek nástroje Unity, dostupné na https://learn.unity.com/project/lego-template

[3] *Lowpoly Textures Pack* dostupný zdarma na Unity Asset Store, auteorem je *N-hance Studio, https://www.nhance.studio/*, dostupné na https://assetstore.unity.com/packages/2d/textures-materials/lowpoly-textures-pack-140717

[4] Zdroj původního obrázku: Leonardo da Vinci, Public domain, via Wikimedia Commons, dostupné na https://commons.wikimedia.org/wiki/File:Mona_Lisa,_by_Leonardo_da_Vinci,_from_C2RMF_retouched.jpg

[5] Isaac Brown, dostupné na https://pin.it/2L2BrRf

[6] https://www.leiainc.com/developer-resources



Obrázek 7.4: Diagram struktury scény. Celá scéna je zobrazena na obrázku 7.1. Hlavním objektem je pokoj s vlastními texturami. Pro obličej figurek je využit obrázek z internetového zdroje, stejně jako pro obraz na stěně pokoje. Podstavu tvoří rovina s aplikovanou texturou květin. Venkovní ceduli tvoří sloupek s aplikovanou texturou připomínající dřevo a samotnou cedulí s obrázkem.

Jako shadery jsou využity tzv. „Image Effect“ shadery umožňující post-processing v Unity. Konkrétní shadery jsou blíže popsány v návrhu 6.1.2.

Detekci hran pomocí hloubky a normál zmíněnou v kapitole 4 nebylo možné z blíže neidentifikovatelných technických problémů implementovat na cílovou platformu Lume Pad.

Skripty pro ovládání efektů Všechny skripty pro ovládání nerealistických efektů mají stejnou strukturu, mění se pouze název a použitý shader, případně některé parametry. Skripty obsahují jednu třídu, tato třída obsahuje dvě metody – *Awake* a *OnRenderImage*. *Awake* se spouští při načtení skriptu a v tomto případě „přichystá“ příslušný shader použitý efekt – vyhledá jej mezi shadery pomocí funkce *find()* a uloží jej. Metoda *OnRenderImage* je volána po dokončení vykreslení scény kamerou a lze tedy změnit vzhled celé scény po vykreslení – post-processing. Ve skriptech k ovládání efektů obsahuje použití funkce *Blit*. Tato funkce slouží ke zkopírování dat pixelu z jedné textury do jiné. V tomto případě z původní textury pixelu do finální pomocí příslušného uloženého shaderu.

Skript k zobrazení či skrytí UI Skript ovládající zobrazení či skrytí UI je napojen na odpovídající zaškrtávací políčko. Pokud je políčko zaškrtnuté, je UI skryto a je povoleno otáčení kamery. Při zapnutí aplikace je políčko zaškrtnuto.

Třída skriptu ovládající skrytí či zobrazení UI obsahuje dvě metody – *Start* a *ToggleValueChangedUI*. Metoda *Start* je volána při zapnutí skriptu časově před jinými metodami. V tomto případě obsahuje přichystání a zapnutí skriptu pro otáčení kamery. Dále nalezení a skrytí panelu, který obsahuje všechny části UI kromě zaškrtávacího políčka ovládajícího zobrazení či skrytí UI. Panel je skryt pomocí funkce *SetActive()*. Poslední částí této metody je použití funkce *OnValueChanged*, která zavolá příslušnou vlastní funkci reakce, pokud je detekována změna hodnoty zaškrtávacího tlačítka. Tato vlastní funkce je *ToggleValueChangedUI*, ve které se na základě hodnoty zaškrtávacího políčka rozhoduje, zda zapnout, či vypnout skript otáčení kamery a zda skrýt, či zobrazit UI.

Skripty ovládající políčka pro zapnutí efektů Skripty ovládající políčka pro zapnutí efektů mají velmi podobnou strukturu jako skript ovládající políčko pro skrytí UI. Obsahuje také metody *Start* a *ToggleValueChanged<name>* – v tomto případě *<name>* označuje název efektu. V metodě *Start* je nalezen a vypnut skript pro ovládání příslušného efektu a poté, jako v případě skrytí UI, použití funkce *OnValueChanged*. Vlastní funkce reakce na změnu hodnoty políčka pak obsahuje znovu podmínku, na základě které se rozhoduje, zda zapnout, či vypnout skript příslušného efektu.

Skripty pro ovládání tabletu Lume Pad Dva skripty pro ovládání sliderů k nastavení hodnot *Convergence Distance* a *Baseline Scaling* mají stejnou strukturu. Obsahují tři metody – *Start*, *Update* a vlastní funkci *ShowSliderValue*. V metodách *Start* i *Update* je pouze volána funkce *ShowSliderValue*. Funkce *ShowSliderValue* získá zvolenou hodnotu sliderů, zapíše ji do textu u sliderů a poté také tuto hodnotu změní v komponentě *Leia Camera* u hlavní kamery.

Skript k ovládání políčka pro zapnutí a vypnutí 3D režimu tabletu má znovu stejnou strukturu jako jako skript ovládající políčko pro skrytí UI. V tomto případě se pouze ve funkci, která je reakcí na změnu hodnoty, zapne, či vypne *Light Field Mode* u objektu *Leia Display*.

Projekt dále obsahuje skript pro ovládní tlačítka *Reset* u sliderů. Tento skript změní hodnoty sliderů, a tedy i *Convergence Distance* a *Baseline Scaling*, na původní hodnoty – konkrétně $5,5$ u *Convergence Distance* a 1 u *Baseline Scaling*.

Skripty pro otáčení kamery a objektů Skript pro otáčení kamery obsahuje metody *Start* a *Update*. V metodě *Start* se nastaví pozice „kurzoru“ pro pohyb na střed obrazovky. V metodě *Update* je nejprve získána pozice „kurzoru“ ve směru osy x a osy y pomocí funkce *GetAxis*. Poté je vypočteno a uskutečněno otočení kamery, respektive „hráče“, který kameru zapouzdřuje. Kód tohoto skriptu je částečně přejatý z internetového zdroje [7].

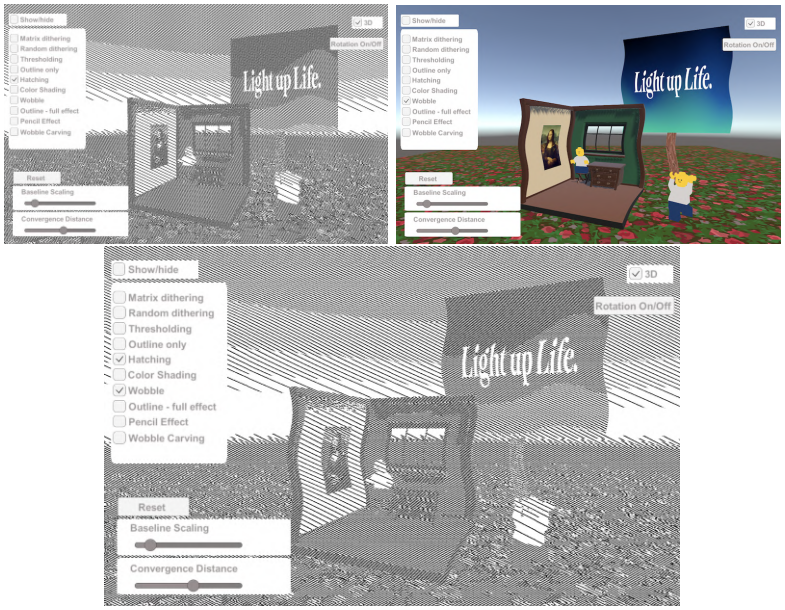
Skript ovládající otáčení objektů je napojený na příslušné objekty scény, které se mají otáčet. Po zapnutí skriptu začne otáčet několika objekty scény – cedulí a figurkami. Po vypnutí skriptu se objekty vrátí zpět do původního stavu. Tento skript je zapínán a vypínán dalším skriptem, který je napojený na tlačítko *Rotation On/Off* v levé části aplikace. Při kliknutí na tlačítko se skript pro rotaci buď zapne, nebo vypne podle předchozí hodnoty.

7.2.4 Kombinace implementovaných efektů

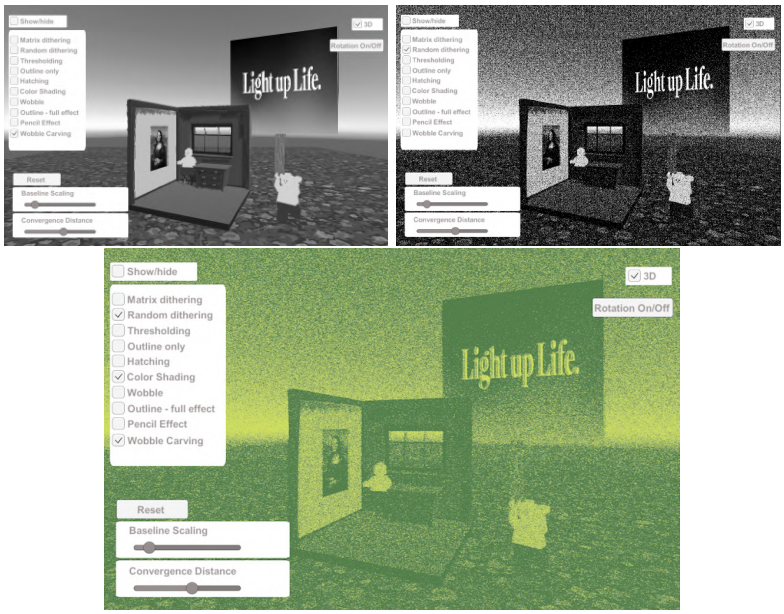
V aplikaci je možné zapnout několik post-processing efektů přes sebe, mohou tak vznikat nové zajímavé efekty. Vybrané kombinace efektů jsou zobrazeny níže na obrázcích 7.5, 7.6 a 7.7.

[7] Zveřejněný kód skriptu *MouseLook.cs* na stránce GitHub, autorem je *MasterKiller1239*, dostupné na https://github.com/MasterKiller1239/Gitapp/blob/2017af443a145be8eee809347be505ccf91859ee/Assets/Scripts/MouseLook.cs

Obrázek 7.5: Kombinace barevného stínovaní a detekce hran pomocí Sobelova operátoru. V obrázku tak vznikají světle zelené obrysy objektů místo bílých při samotné detekci hran.



Obrázek 7.6: Kombinace šrafování a efektu zvlnění obrazu. Původní obraz je zvlněný a šrafy zůstávají rovnoběžné.



Obrázek 7.7: Kombinace efektu barevného stínování (7.5), efektu pokřivení obrazu a náhodného rozptýlení. Vzniká efekt rozmazaného zeleného šumu.

Kapitola 8

Testování

Testování je provedeno na 10 uživatelích různých věkových kategorií.

8.1 Data z testování

V této sekci jsou v tabulkách zapsána průměrná hodnocení z testování jednotlivých efektů. Dále jsou vyhodnoceny průměrně nejlépe a nejhůře hodnocené efekty. Je také zapsán efekt, který nejvíce uživatelů označilo jako nejlepší, a efekt, který naopak nejvíce uživatelů označilo jako nejhorší. Na konci sekce jsou také zapsány vybrané hodnoty „convergance distance“ a „baseline scaling“ jednotlivých uživatelů společně s jejich věkem.

8.1.1 Průměrné hodnocení jednotlivých efektů

V této sekci jsou v tabulce 8.1 zobrazena průměrná hodnocení jednotlivých kritérií efektů. Níže jsou popsána také slovní hodnocení efektů.

Matrix dithering – maticový dithering Maticový dithering je jeden z průměrně hodnocených efektů. Dle hodnocení uživatelů efekt jemně zhoršoval prostorový dojem.

Random dithering – náhodný dithering Náhodný dithering je uživateli výrazně lépe hodnocený ve 2D, než ve 3D. Způsobené to je menším rozlišením ve 3D režimu. Uživatelé rovněž hodnotili efekt z hlediska viditelnosti tak, že výrazně zhoršoval prostorový dojem.

Tresholding – prahování Prahování je také průměrně hodnoceným efektem, který nezlepšoval, ani nezhoršoval prostorový dojem.

Outline – detekce hran Detekce hran je lépe hodnocena ve 2D, než ve 3D, protože hrany jsou ve 2D režimu více jemné a tenké. Detekce hran napříč uživateli spíše zlepšuje prostorový dojem.

Hatching – šrafování Šrafování je jeden z hůře hodnocených efektů. Dle uživatelů výrazně zhoršuje prostorový dojem. Hodnocení ve 2D a 3D je srovnatelné.

Color shading – barevné stínování Barevné stínování je spíše nadprůměrně hodnocené. Prostorový dojem zůstává srovnatelný. Přívětivost se výrazně neliší ve 2D a 3D režimu. Někteří uživatelé označují efekt za uklidňující pro oči. Důvodem může být mimo jiné redukce barev do tří odstínů zelené, je tedy méně přehlcující.

Wobble – efekt zvlnění obrazu Efekt zvlnění obrazu je také efekt s nadprůměrným hodnocením. Většina uživatelů uvádí, že prostorový dojem zůstává stejný či mírně lepší. Přívětivost je srovnatelná ve 2D i 3D.

Outline – full effect – černobílý efekt detekce hran Černobílý efekt detekce hran většina uživatelů hodnotila nadprůměrně z hlediska zlepšení prostorového dojmu. Přívětivost je spíše průměrná a horší ve 3D.

Pencil – efekt tahů tužky Tužkový efekt je jedním z průměrně hodnocených efektů. U některých uživatelů je hodnocen velmi nadprůměrně, u jiných zase výrazně podprůměrným. V průměru vychází hodnocení kolem střední hodnoty, ale jednotlivá hodnocení obsahuje hodnoty z opačných stran hodnotící škály.

Wobble carving – šedotónový efekt pokřivení obrazu Efekt pokřivení obrazu je uživateli průměrně hodnoceným efektem. Prostorový dojem zůstává stejný.

Efekt	**Viditelnost 3D**	**Přívětivost 3D**	**Přívětivost 2D**	**Komfort 3D**
Matrix dithering	2,4	2,3	3	2,8
Random dithering	1	1,4	2,9	1,8
Tresholding	3	2,7	2,9	2,7
Outline	3,1	2,3	3,1	2,9
Hatching	1,2	1,7	1,4	1,7
Color shading	2,8	3,1	3,2	3,4
Wobble	2,9	3,3	3,5	3,6
Outline – full	3,5	2,9	3,2	3,1
Pencil	2,6	2,4	2,3	2,6
Wobble carving	2,9	3,3	2,7	3,3

Tabulka 8.1: Průměrná hodnocení jednotlivých efektů podle daných kritérií. Uživatelé hodnotili efekt podle čtyř kritérií – viditelnost 3D, přívětivost efektu ve 3D a 2D a komfort pohledu na efekt ve 3D. Všechna kritéria jsou hodnocena na škále od nuly do pěti, kde nula je nejhorší hodnocení a pět nejlepší. v kritériu viditelnosti 3D uživatelé hodnotí, zda aplikovaný efekt zlepšuje, či zhoršuje prostorový dojem scény. Kritérium přívětivosti hodnotí, jak danému uživateli efekt „líbí“. Komfort pohledu ve 3D testuje, jak moc je efekt příjemný, či nepříjemný na pohled po delší dobu.

8.1.2 Nejlepší a nejhorší efekty dle jednotlivých kritérií

V tabulce 8.2 níže jsou uvedeny nejhorší a nejlepší efekty dle jednotlivých kritérií.

Kritérium	*Nejlepší*	*Nejhorší*
Viditelnost 3D	Outline – full effect	Random dithering
Přívětivost 3D	Wobble, Wobble carving	Random dithering
Přívětivost 2D	Wobble	Hatching
Komfort 3D	Wobble	Hatching

Tabulka 8.2: Tabulka nejlépe a nejhůře hodnocených efektů v daných kritériích. V kritérii viditelnosti byl nejlépe hodnocený černobílý efekt detekce hran (*Outline – full effect*). Důvodem je to, že hrany napomáhají prostorovému dojmu. Nejhůře hodnoceným efektem podle tohoto kritéria byl efekt náhodného rozptýlení (*Random dithering*). Z hlediska přívětivosti byl jak ve 2D, tak ve 3D nejlépe hodnocený efekt zvlnění obrazu (*Wobble*). Nejhůře hodnocený ve 2D podle tohoto kritéria byl efekt šrafování (*Hatching*) a ve 3D efekt náhodného rozptýlení (*Random dithering*). V rámci komfortu při pohledu na efekt byl nejlépe hodnocený efekt zvlnění obrazu (*Wobble*) a nejhůře pak šrafování (*Hatching*).

8.1.3 Efekty označené nejlepším a nejhorším

Nejlépe hodnocený efektem je *Wobble* – efekt zvlnění obrazu. Nejhůře hodnocenými efekty jsou *Hatching* – efekt šrafování – a *Random dithering* – náhodné rozptýlení.

V tabulce 8.3 níže je zapsán počet uživatelů, kteří označili efekt jako nejlepší a kteří jako nejhorší na konci testování.

Efekt	*Označen nejlepším*	*Označen nejhorším*
Matrix dithering	1	0
Random dithering	0	3
Tresholding	0	2
Outline	0	1
Hatching	0	3
Color shading	1	0
Wobble	6	0
Pencil	1	1
Wobble carving	0	0

Tabulka 8.3: V tabulce je u každého efektu zapsáno, kolik uživatelů efekt označilo na konci testování jako nejlepší a kolik jako nejhorší. Nejčastěji označován nejlepším byl efekt zvlnění obrazu (*Wobble*) a nejhorším efekty náhodného rozptýlení a šrafování (*Random dithering* a *Hatching*).

Uživatel	*Věk*	*Brýle*	*C.D.*	*B.S.*
1	51	NE	5,5	1,0
2	21	NE	1,8	1,5
3	22	NE	5,5	1,0
4	21	ANO	4,6	3,4
5	48	NE	5,5	1,0
6	48	ANO	5,5	1,0
7	48	NE	2,5	2,2
8	73	ANO (na čtení)	5,4	1,3
9	76	ANO (na čtení)	5,5	1,0
10	18	ANO	4,2	1,0

Tabulka 8.4: Tabulka obsahující údaje o uživateli preferovaných hodnotách Convergence Distance (***C.D.***) a Baseline Scaling (***B.S.***). Současně je v tabulce také zapsán věk každého uživatele a fakt, zda nosí brýle. Za brýle jsou považovány brýle na dálku. Pokud nosí uživatel brýle na čtení a měl je u testování, je tato skutečnost uvedena v závorce.

8.1.4 Hodnoty Convergence Distance a Baseline Scaling

Níže v tabulce 8.4 jsou uvedeny uživateli preferované hodnoty Convergence Distance (označeno jako C.D.) a Baseline Scaling (označeno jako B.S.) společeně s věkem a skutečností, zda nosí (a u testování na sobě měli) brýle. Původními hodnotami v aplikaci jsou Convergence distance 5,5 a Baseline Scaling 1,0.

Polovina z deseti uživatelů nenašla lepší hodnoty než původní. U druhé poloviny uživatelů se preferované hodnoty lišily. Největší hodnoty, a tedy i největší míru 3D efektu, má nastavené uživatel s číslem 4. Zda tento fakt souvisí s tím, že uživatel má oční vadu astigmatismus, by mohl posoudit specialista, ale zajisté jde o zajímavý poznatek.

Uživatel číslo 1 uvádí, že u některých efektů se s jinou hodnotou Baseline Scaling přívětivost výrazně zlepší. Konkrétně u efektu *Wobble carving* při nastavení hodnoty 1,8 a u efektu *Color shading* při hodnotě 2,2.

8.2 Vyhodnocení a poznatky z testování

Z dat vyplývá, že nerealistické efekty je možné realizovat na tabletu Lume Pad, ovšem s určitými omezeními a postupy.

Je třeba, aby v každé aplikaci využívající 3D režim měli uživatelé možnost nastavit si individuálně hodnoty Convergence Distance a Baseline Scaling. Důvodem je to, že v testování se uživateli preferované hodnoty výrazně lišily.

Rozlišení, které momentálně nabízí light field technologie tabletu Lume Pad, není pro mnoho efektů dostatečné. To způsobuje výrazné zhoršení kvality a uživatelské přívětivosti ve 3D režimu některých efektů.

Z testování vyplývá také to, že brýle nejsou pro kvalitní pohled ve 3D překážkou. Zda jsou ale překážkou specifické oční vady, je již na uvážení specialistů z oboru.

V rámci testování se také ukázalo, že věk, a tedy často i zhoršená schopnost vidět na blízko, může napomoct s komfortem při pohledu ve 3D režimu. Nejstarší uživatel neměl s 3D

režim v rámci komfortu žádný problém a jako jediný z uživatelů by si dokázal představit tuto light field technologii větších obrazovkách jako jsou televizory.

Několik uživatelů uvedlo, že na začátku testování pro ně byl pohled na tablet nepříjemný, ale postupně se komfortnost zlepšila a na konci testování se již cítili výrazně komfortněji při pohledu.

Jeden z uživatelů uvedl, že technologie vypadá svým prostorovým dojmem stejně jako 3D režim konzole Nintendo 3DS z roku 2010 (sekce 3.4.3). Uvedl také, že právě díky znalosti konzole Nintendo 3DS, pro něj není pohled na tablet nijak nekomfortní.

Kapitola 9

Závěr

Hlavním cílem bakalářské práce bylo zjistit, zda je možné kvalitně a přívětivě implementovat nerealistické efekty na autostereoskopickém tabletu Lume Pad. Pro testování těchto skutečností slouží aplikace vyvinutá v nástroji Unity. Aplikace umožňuje pohodlně prohlížet vybrané nerealistické efekty a obsahuje další prvky uživatelského rozhraní, které pohodlnosti pohledu napomáhají.

Po studiu potřebných teoretických znalostí, jako je holografie, light field, stereoskopie a zobrazování ve 3D, byly vybrány a popsány některé nerealistické efekty. Tyto efekty byly následně implementovány a použity ve výsledné aplikaci.

Pomocí aplikace bylo poté provedeno testování na deseti uživatelích různých věkových kategorií. U jednotlivých efektů byla hodnocena kritéria jako kvalita prostorového dojmu efektu ve 3D, přívětivost efektu ve 2D a 3D a komfort pohledu na efekt po delší dobu ve 3D režimu. Některé efekty ovšem zdaleka nedosahují kvality, se kterou je možné efekty implementovat ve 2D režimu. Hlavním důvodem je výrazně nižší rozlišení ve 3D režimu. Bylo zjištěno také to, že pro vývoj kvalitních aplikací na Lume Pad, by měli mít uživatelé vždy možnost nastavit si míru 3D režimu. Uživatelé preferovali výrazně odlišné hodnoty prostorovosti.

Zadání práce bylo splněno, aplikace byla úspěšně otestována a z testování byly vyvozeny příslušné závěry. V budoucnu by bylo možné aplikaci doplnit o další nerealistické efekty, více možných scén či volný pohyb uživatele po scéně. Současně lze aplikaci implementovat na nové platformě, tabletu Lume Pad 2, a zjistit, zda kvalitnější 3D režim zobrazí efekty kvalitněji a přívětivěji, než na původním tabletu Lume Pad. V neposlední řadě by bylo přínosné spojit se při testování s odborníky z příslušných oborů a zjistit, zda a které oční vady mohou mít za následek odlišné vnímání ve 3D režimu.

Literatura

[1] *Looking Glass Factory* [online]. Dostupné z: https://lookingglassfactory.com/.

[2] *Lume Pad About* [online]. Dostupné z: https://www.leiainc.com/about.

[3] *Lume Pad Unity SDK* [online]. Dostupné z: https://developer.leiainc.com/unity-sdk/unity-sdk-guide.

[4] *Premultiplied alpha* [online]. Dostupné z: https://microsoft.github.io/Win2D/WinUI3/html/PremultipliedAlpha.htm.

[5] *Unity User Manual* [online]. Dostupné z: https://docs.unity3d.com/Manual/index.html.

[6] Aieta, Francesco, Vo, Sonny, Ma et al. Invited Paper: A Diffractive LCD Backlight Approach to Dynamic Lightfield Displays. In:. 2016. DOI: 10.1002/sdtp.10625. Dostupné z: https://sid.onlinelibrary.wiley.com/doi/10.1002/sdtp.10625.

[7] Antonopoulos, G. a Fudos, I. An Interactive Tool Suite for Embossing 2D Images. In:. Leden 2010, s. 207 – 214. DOI: 10.1109/SITIS.2009.42.

[8] Engel, W. F. *Shaderx7: Advanced rendering techniques*. Course Technology, 2009. ISBN 978-1-58450-598-3.

[9] Haas, J. K. A History of the Unity Game Engine. In:. 2014. Dostupné z: https://www.semanticscholar.org/paper/A-History-of-the-Unity-Game-Engine-Haas/5e6b2255d5b7565d11e71e980b1ca141aeb3391d.

[10] Jana, D. *Velká Kniha Barev: Kompletní Průvodce Pro Grafiky, fotografy a designéry*. Computer Press, 2012. ISBN 978-80-251-3785-7.

[11] Králová, M. [online]. Dostupné z: http://edu.techmania.cz/cs/encyklopedie/fyzika/svetlo/holografie.

[12] Larcom, R. a Burnett, T. Viewing Light-field Displays. In:. 2018. DOI: 10.1002/sdtp.12269. Dostupné z: https://doi.org/10.1002/sdtp.12269.

[13] Majumder, A. *Lightfield* [online]. Dostupné z: https://www.ics.uci.edu/~majumder/vispercep/CS213-lightfield.pdf.

[14] Milet, T. *Grafické intro 64kB s použitím OpenGL*. Brno, CZ, 2012. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/9301/.

[15] Milliron, T. S. *Image Based Rendering: Introduction and Theory* [online]. Dostupné z: https://www.cs.princeton.edu/courses/archive/spr98/cs598d/lectures/milliron/Intro/sld004.htm.

[16] Nováková, P. *Digitální holografie a její aplikace v prostředí Matlab*. 2011. Diplomová práce. Univerzita Palackého v Olomouci, Přírodovědecká fakulta. Dostupné z: https://theses.cz/id/cgrr10/Diplomov_prce_final1.pdf.

[17] Portyš, J. *Binokulární vidění*. Brno, CZ, 2013. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. Dostupné z: http://hdl.handle.net/11012/25940.

[18] Reichl, J. a Všetička, M. *Holografie* [online]. Dostupné z: http://fyzika.jreichl.com/main.article/view/457-holografie.

[19] Strickland, J. *How the Nintendo 3DS Works* [online]. Dostupné z: https://electronics.howstuffworks.com/nintendo-3ds.htm.

[20] Styan, G. P. Hadamard products and multivariate statistical analysis. *Linear Algebra and its Applications*. 1973, sv. 6, s. 217–240. DOI: https://doi.org/10.1016/0024-3795(73)90023-2. ISSN 0024-3795. Dostupné z: https://www.sciencedirect.com/science/article/pii/0024379573900232.

[21] Thomas Porter, T. D. Compositing Digital Images. In:. 1984. DOI: 10.1145/964965.808606. Dostupné z: https://dl.acm.org/doi/10.1145/964965.808606.

[22] Tiwari, J. *OpenGL Rendering Pipeline / An Overview* [online]. Dostupné z: https://www.geeksforgeeks.org/opengl-rendering-pipeline-overview/.

[23] Špačková, Z., Hájek, J. a Caletková, L. *Holografie* [online]. Dostupné z: https://tydenvedy.fjfi.cvut.cz/2010/cd/prispevky/pres/holografie.pdf.

[24] Žára, Beneš, Sochor a Felkel. *Moderní Počítačová Grafika*. Computer Press, 2004. ISBN 80-251-0454-0.

Přílohy

```
1
2  Shader "Unlit/Example"
3  {
4      Properties
5      {
6          _MainTex ("Texture", 2D) = "white" {}
7      }
8      SubShader
9      {
10
11         Pass
12         {
13             CGPROGRAM
14             #pragma vertex vert
15             #pragma fragment frag
16
17             #include "UnityCG.cginc"
18
19             struct appdata
20             {
21                 float4 vertex : POSITION;
22                 float2 uv : TEXCOORD0;
23             };
24
25             struct v2f
26             {
27                 float2 uv : TEXCOORD0;
28                 float4 vertex : SV_POSITION;
29             };
30
31             sampler2D _MainTex;
32             float4 _MainTex_ST;
33
34             v2f vert (appdata v)
35             {
36                 v2f o;
37                 o.vertex = UnityObjectToClipPos(v.vertex);
38                 o.uv = TRANSFORM_TEX(v.uv, _MainTex);
39                 return o;
40             }
41
42             fixed4 frag (v2f i) : SV_Target
43             {
44                 fixed4 col = tex2D(_MainTex, i.uv);
45                 return col;
46             }
47             ENDCG
48         }
49     }
50 }
```

Kód 1: Příklad jednoduchého Unlit shaderu. Pomocí „_MainTex“ v bloku „Properties“ lze v editoru nastavit texturu, která je poté ve fragment shaderu (funkce „frag“) aplikována na materiál objektu. Dále kód obsahuje definici struktur pro vertex („appdata“) a fragment („v2f“) shader a samotnou definici vertex shaderu.