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

DYNAMICKÁ PREZENTACE FOTOGRAFIÍ S VYUŽITÍM HLOUBKOVÉ MAPY

DYNAMIC IMAGE PRESENTATIONS USING DEPTH MAPS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. JIŘÍ HANZLÍČEK**
AUTHOR

VEDOUCÍ PRÁCE **Doc. Ing. MARTIN ČADÍK, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

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

Zadání diplomové práce

Student: **Hanzlíček Jiří, Bc.**
Program: Informační technologie Obor: Počítačová grafika a multimédia
Název: **Dynamická prezentace fotografií s využitím hloubkové mapy**
Dynamic Image Presentations Using Depth Maps
Kategorie: Počítačová grafika

Zadání:

1. Seznamte se s algoritmy řízené syntézy textur a odhadu pózy kamery.
2. Vytipujte metody vhodné pro dynamickou prezentaci statické fotografie a popište jejich vlastnosti.
3. Navrhněte a implementujte systém pro dynamickou prezentaci fotografií s využitím modelu terénu, simulujte pohyb virtuální kamery.
4. S navrženým systémem experimentujte a diskutujte možnosti budoucího vývoje.

Literatura:

- Dle pokynů vedoucího, datovou sadu dodá vedoucí práce.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3 zadání.

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

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

Abstrakt

Tato diplomová práce se zabývá problematikou, jak využít hloubkovou mapu k dynamické prezentaci statické fotografie. V práci je představen algoritmus sloužící k vytvoření prostorového modelu. Ten poslouží k vykreslení objektů na vstupní fotografii tak, aby při pohybu virtuální kamery v jeho okolí došlo k vytvoření paralax efektu díky hloubce v obrazu. Dále je v práci navržen postup, jak vyplnit chybějící obrazová data v modelu. Je navrženo využití techniky nazývané vedená syntéza textur, kde dochází k využití vykresleného modelu v roli vodítka syntézy. Doplněné obrazové informace do modelu následně umožňují větší svobodu pohybu virtuální kamery v okolí modelu, aniž by docházelo k vykreslení vizuálně rušivých fragmentů. Výsledný pohyb virtuální kamery je možné následně uložit do jednoduché video sekvence, která poslouží k prezentaci vstupní fotografie.

Abstract

This master's thesis focuses on the dynamic presentation of still photography using a depth map. This text presents an algorithm that describes the process of creating a spatial model which is used to render input photography so that the movement of virtual camera creates parallax effect due to depth in image. The thesis also presents an approach how to infill the missing data in the model. It is suggested that a guided texture synthesis is used for this problem by using rendering outputs of the model themselves as guides. Additional information in model allows the virtual camera to move more freely. The final result of the camera movement can be saved to simple video sequence which can be used for presenting the input photography.

Klíčová slova

Hloubková mapa, Výpočetní fotografie, Paralax efekt, Řízená syntéza textur

Keywords

Depth map, Computational photography, Parallax effect, Guided texture synthesis

Citace

HANZLÍČEK, Jiří. *Dynamická prezentace fotografií s využitím hloubkové mapy.* Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. Ing. Martin Čadík, Ph.D.

Dynamická prezentace fotografií s využitím hloubkové mapy

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana docenta Martina Čadíka. Další informace mi poskytl pan Ing. Ondřej Jamriška. Uvedl jsem všechny literární prameny a publikace ze kterých jsem čerpal.

.....
Jiří Hanzlíček
22. května 2019

Poděkování

Rád bych poděkoval vedoucímu mé diplomové práce Doc. Ing. Martinu Čadíkovi, Ph.D za pomoc při výběru tématu práce, odborné vedení a cenné rady při vyhotovování této diplomové práce. Dále pak pracovníkům katedry počítačové grafiky a interakce z Českého vysokého učení technického v Praze za poskytnutí odborných rad řešících problematiku syntézy textur. A v neposlední řadě skupině nejužších kamarádů, která podržela moji pracovní morálku v těch nejtemnějších chvílích a napomohli mi při korektuře mých češtinářských nešvarů.

Obsah

1 Úvod **3**

2 Výpočetní fotografie **4**
2.1 Barva 4
2.2 Zaznamenávání fotografií 5
2.3 Hloubková mapa 6
2.4 Vedená syntéza textur 8

3 Dynamická prezentace fotografií **9**

4 Návrh a technologie **11**
4.1 Struktura systému 12
4.2 Načítání zdrojových dat 14
4.3 Vykreslování modelu 14
4.4 Syntéza textur 17
4.5 Třídní rozložení aplikace 19
4.6 Uživatelské rozhraní aplikace 22

5 Práce s prostorovým modelem **27**
5.1 Struktura dat modelu 27
5.2 Vytvoření modelu 28
5.3 Perspektiva 32
5.4 Detekce děr v modelu 36
5.5 Vytváření záplat modelu 40

6 Dosažené výsledky **45**
6.1 Vytvořený model 47
6.2 Rozbor syntetických textur 48
6.3 Záplaty na modelu 51

7 Možnosti dalšího rozšíření **52**
7.1 Alternativní implementace 53

8 Závěr **54**

Literatura **55**

A Výstup vykreslovacího procesu **57**

B Váhové masky pro syntézu **59**

C Grafické uživatelské rozhraní **63**

D Testovací sada **66**

E Obsah DVD **70**

F Návod k použití aplikace **71**

Kapitola 1

Úvod

Motivací k vytvoření videa ze samostatné fotografie je posílení zážitku ze samotného sledování. Jelikož při zaznamenávání scenérie na fotografii může dojít ke ztrátě vjemu hloubky, tak dynamicky prezentovaná fotografie může například umocnit velkolepost některých staveb či skalních útvarů. Vytvoření přechodu podtrhujícího hloubku na fotografii může zpestřit prezentaci rozsáhlého fotoalba. Každý na sobě dokáže pozorovat jistý pokles pozornosti, když sleduje velké množství fotografií za sebou. Z tohoto důvodu jsou snímky, které zaznamenávají mohutné útvary, ideálními kandidáty k vytvoření vizuálně zajímavého video přechodu, který opět přiláká pozornost usínajícího diváka.

Tato diplomová se zabývá dynamickou prezentací statických fotografií za pomoci použití hloubkové mapy. Hlavní text je logicky rozčleněn do šesti kapitol. První teoretickou kapitolou je kapitola 2 zabývající se problematikou pořizování digitálních fotografií a hloubkových map. V jejím závěru se nachází teoretický úvod do principů řízené syntézy textur užívající obrazových informací z jediné fotografie. Na ni navazuje krátká kapitola 3 popisující problematiku dynamické prezentace statických fotografií a jak může taková fotografie být využita. Další již poněkud méně teoretická kapitola 4 se zabývá návrhem struktury programu, který slouží k vytvoření video sekvence prezentující předloženou fotografii a její hloubkovou mapu za pomoci prostorového modelu. U jednotlivých úkonů návrhu jsou představeny konkrétní technologie, které jsou použity k finální implementaci. Vzápětí následuje podrobnější popis technologie zvolené k syntéze textur, která poslouží k doplňování obrazových informací ve vytvořeném modelu. Na to následují implementační detaily finální aplikace této diplomové práce. Zde je popsáno, jakým způsobem je aplikace implementována a z jaký dílčích bloků se skládá. Závěrem této kapitoly je popis grafického uživatelského prostředí výstupní aplikace.

Samostatná kapitola 5 v textu je věnována vzniku a úpravám prostorového modelu. Tato kapitola je čistě experimentální povahy. Obsahuje popis jednotlivých dílčích úkonů, které je nutné vykonat k úspěšnému vytvoření modelu. Jsou zde diskutovány rozdílné postupy, které mají vliv na vzhled vytvořeného modelu. Většina zmíněných postupů je v textu graficky zobrazena. U postupů, se kterými bylo experimentováno, je uvedeno, proč nejsou vhodné pro finální navrženou implementaci. V rámci této kapitoly je popsán postup, jakým dochází k samostatnému vykreslení vzniklého modelu tak, aby byly zachovány proporce vstupní fotografie a zároveň vynikl prostorový efekt modelu při pohybu virtuální kamery. V průběhu této kapitoly je také diskutováno, jakým vhodným způsobem detekovat fragmenty v modelu, o kterých nejsou známy obrazové informace z původní fotografie. Závěr této kapitoly je věnován popisu postupu, jak tato místa vyplnit. Závěrem praktické části této práce je rozbor dosažených výsledků a ukázka výstupního modelu, který byl doplněn o data pomocí algoritmů dříve popsaných.

Kapitola 2

Výpočetní fotografie

Výpočetní fotografie je vědeckou disciplínou kombinující optickou fyziku a techniky zpracování obrazu [18]. Dochází zde k zabroušení do problematiky zabývající se optickými vlastnostmi objektivů, funkcionalitou digitálních senzorů, chybami zobrazení a dalšími. Výpočetní fotografie může přispět ke zlepšení kvality pořízených snímků. Příkladem může být odmlžení objektů na fotografii nebo korekce vinětace vzniklé použitím starších objektivů a mnohé další. V podstatě se výpočetní fotografie zabývá částečně vším spojeným s pořizováním a zpracováním fotografií, ať už se jedná o korekce optických vad nebo právě o jejich umělé vytvoření, které má za úkol stylizovat fotografii.

2.1 Barva

Barvou rozumíme individuální subjektivní vjem každého jedince, jakým způsobem interpretuje světlo dopadající do lidského oka [18]. K interpretaci barvy slouží v lidském oku přes 100 miliónů světločivých buněk na oční sítnici [8]. Ty dělíme na tyčinky a čípky. Tyčinky slouží k rozpoznávání kontrastů, tedy intenzitě dopadajícího světla do lidského oka. I když samotné barevné vidění neumožňují, lidský zrak primárně spoléhá na jejich funkci při rozpoznávání a orientaci. Nejen z tohoto důvodu je v lidském oku statisticky zhruba dvacetkrát více tyčinek než čípků. Schopnost rozpoznat barvu se u jednotlivých živočichů postupem času vyvinula různým způsobem. Čípky v oku oproti tyčinkám slouží k identifikaci barev. Konkrétně barev červené, zelené a modré. Pro každou barvu slouží jiný fotoreceptor. Fyzicky se od sebe nijak neliší, pouze chemickým složením obsahujících látek, které reagují na určité vlnové délky.

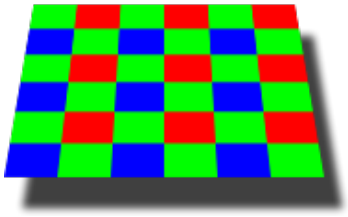
Jak čípky, tak tyčinky se nacházejí na sítnici, což je poslední článek optického aparátku zpracovávající lidské vidění [8], který odesílá signály do lidského mozku. Celý optický systém lidského oka se pak ještě skládá z rohovky, duhovky, čočky a sklivce. To jsou neméně důležité součásti, které také ovlivňují kvalitu lidského zraku, ale na vjemu barvy se zásadně nepodílejí.

Nyní, když bylo popsáno, jakým způsobem je interpretováno světlo vstupující do lidských očí, je dobré si také definovat barvu jako vlastnost materiálu. Zdrojem elektromagnetického vlnění je zrychlení pohybu elektrického náboje. Ve většině běžných objektů však nedochází k tak razantním změnám pohybů nabitých částic, aby emitovaly světlo. Proto také většina věcí sama od sebe nesvítí. Z toho vyplývá, že abychom mohli cokoliv spatřit, tak to musí být osvětleno, a to z důvodu, aby se světlo odrazilo od pozorovaného objektu směrem k pozorovateli. Barvou předmětu pak rozumíme tu část viditelného spektra, která

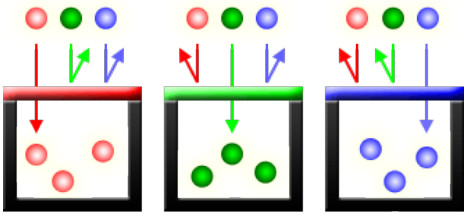
se od pozorovaného předmětu odrazila [18], tudíž nebyla předmětem pohlcena. Proto se například černé objekty na denním světle mnohem více zahřívají než jiné, protože většinu světelné energie pohltí.

2.2 Zaznamenávání fotografií

Fotografie je zachycením obrazového záznamu v čase [5]. Konkrétně se jedná o zachycení dopadajícího světla prostřednictvím světlocitlivého materiálu uvnitř fotoaparátu. U analogových fotoaparátů pak toto médium zpravidla bývá film a u digitálních senzor. Fotografii pak můžeme chápat jako dvourozměrný spojitý signál odpovídající intenzitě světla dopadajícího na konkrétní bod. Toto lze poměrně dobře interpretovat analogově pomocí chemických reakcí na filmu, nicméně digitální zachycení je poněkud komplikovanější, protože současné světlocitlivé součástky dokáží pouze měřit celkovou intenzitu dopadajícího světla. Proto bývají před jednotlivé fotocitlivé senzory předřazeny filtry, které propouští jen určitou část viditelného spektra. Konkrétně jsou inspirovány lidským zrakem, tedy na jednom čipu můžeme nalézt filtry propouštějící světlo náležící do spektra kolem barev červené, zelené a modré, viz 2.1b. To přináší ovšem problém, že v každém bodě čipu nemůžeme detekovat celé viditelné spektrum. Filtry jsou uspořádány v mřížce tak, aby bylo možné bilineární interpolací dopočítat barvu obrazu v každém bodě snímače. V současné době používané uspořádání filtrů se nazývá Bayerova mřížka. Jak je z uspořádání filtrů patrné na obrázku 2.1a, tak prvků propouštějících zelenou barvu je na čipu dvakrát více, což koresponduje s tím, že lidské oko je nejcitlivější na odstíny barev zelené. Pro dosažení vyšší přesnosti fotografií pro vědecké účely se pak používají fotosoustavy sestavené ze tří senzorů, kde každý má svůj vlastní barevný filtr, nebo z jednoho senzoru, který postupně zaznamená obraz pomocí různých filtrů.



(a) Bayerova mřížka [5]



(b) Demonstrace propustnosti určitých vlnových délek jednotlivými filtry [5]

Obrázek 2.1: Využití filtrů propouštějících určitý rozsah viditelného spektra před senzorem

2.2.1 Úhel zorného pole

Zorným polem rozumíme část prostoru, kterou je lidské oko nebo optický přístroj schopen zachytit [5]. Číselným vyjádřením zorného pole je zorný úhel[1]. Určuje se jako úhel od osy přístroje po nejzazší pozorovatelný bod v daném směru. K určení tohoto úhlu poslouží následující vzorec (2.1) doplněný ilustrací 2.2. V tomto jednoduchém schématu dochází k zanedbání vnitřních optických vlastností objektivu fotoaparátu. Toto si můžeme dovolit na základě toho, že ideální objektiv je definován pouze ohniskovou vzdáleností. Korelace

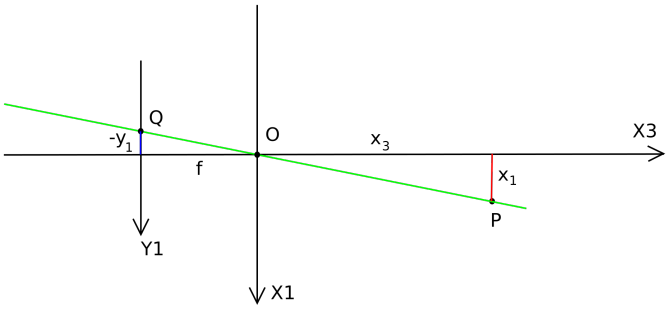
[1] V anglické literatuře se nazývá field of view, z toho je také odvozena zkratka *fov*

mezi těmito vlastnostmi je pak taková, že čím větší je úhel zorného pole, tím menší je ohnisková vzdálenost. Dalším ovliňujícím faktorem je velikost zaznamenávajícího média. U digitálních fotoaparátů se jedná tedy o velikost snímače. Ta je však u jednotlivých fotoaparátů konstantní a ke změně úhlu zorného pole dochází pouze výměnou objektivů nebo jejich přenastavením[2]. Na objektivech uváděné ohniskové vzdálenosti se vztahují k takzvanému plnoformátovému snímači, jehož rozměry jsou rovny velikosti plného políčka kinofilmu používaného u klasických fotoaparátů[3].

$$fov = 2 \cdot \arctan(\frac{y_1}{f}) \tag{2.1}$$

Na ilustračním obrázku, který ilustruje výpočet úhlu záběru (2.1) v jedné dimenzi, je zobrazeno:

- osa $X3$ objektivu kolmá na plochu snímače
- rovina $Y1$ snímače
- velikost y_1 je polovina velikosti snímače
- vzdálenost f je ohnisková vzdálenost objektivu od snímače
- vzdálenost x_1 je vzdálenost bodu P od osy objektivu
- bod O je ohnisko objektivu
- bod Q je krajním bodem snímače
- bod P je posledním zaznamenatelným bodem na fotografii ve vzdálenost x_3 od ohniska



Obrázek 2.2: Ilustrace ke vztahu (2.1) [23]

2.3 Hloubková mapa

Hloubková mapa zaznamenává vzdálenosti bodů v obraze od určitého bodu v prostoru. U statických fotografií pak takováto hloubková mapa může posloužit k získání třetího rozměru. Historicky se takovéto hloubkové mapy k fotografiím získávají pomocí numerických

[2]U objektivů typu zoom
[3]šířka:36 mm X výška: 24 mm

metod z oblasti stereovidění, kde typicky tyto metody vycházejí ze znalosti velikosti odsazení dvou snímků [17]. Pro malé vzdálenosti je vhodné mít odsazení snímku menší a naopak čím jsou měřené vzdálenosti větší, tím je nutnější odsazení dvou snímků zvětšit. V současné době hlavně díky rozvoji technik využití neuronových sítí začíná být možné odhadovat hloubku přímo z jednoho snímku.

2.3.1 Syntetická hloubková mapa

Jedním ze způsobů, jak získat znalost o hloubce objektů na fotografii, je možnost její lokalizace. Je-li známo místo pořízení fotografie a je-li k dispozici model okolního terénu, existují algoritmy, které dokáží na základě povahy horizontu zarovnat fotografii s modelem [24]. Z takto zarovnané fotografie s modelem je možné k fotografii změřit hloubku pro každý její pixel na základě výpočtu vzdálenosti od místa pořízení k místu dopadu na modelu. Takto pořízená hloubka bude výborně odpovídat skutečnosti například u horských panoramat. Bohužel, tato metoda vytváření hloubkové mapy nedokáže zohlednit objekty přítomné na fotografii, které nejsou v porovnávaném modelu. Příkladem aplikace takovéto metody je software LOCATE (Visual Localization in Natural Environments) [24] vyvíjený na Fakultě informačních technologií Vysokého učení technického v Brně.

Obrázek 2.3: Ilustrace lokalizované fotografie [24]

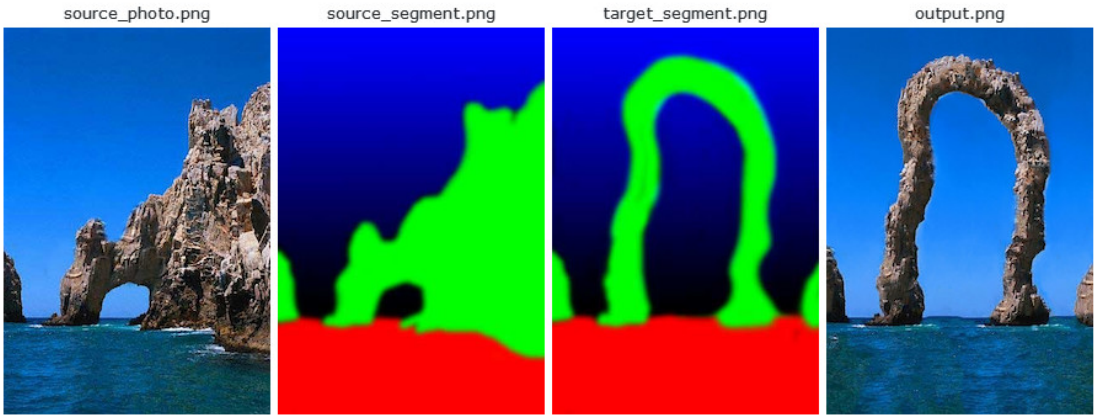
2.3.2 Hloubková mapa generovaná z obrazu

S rozmachem neuronových sítí a rostoucím výkonem výpočetních zařízení začíná být řešitelný problém detekce hloubky v obraze filtrační metodou v relativně rychlém čase. K výpočtu orientační hloubky k jednotlivým snímkům je možné natrénovat síť, která na poměrně široké variaci snímků dokáže hloubku odhadnout. Jedním takovým úspěšným pokusem o trénování neuronové sítě je projekt nazývaný MegaDepth [13].

Obrázek 2.4: Ilustrace výstupu hloubkových map [13]

2.4 Vedená syntéza textur

Vedená syntéza textur je postup vytváření nových textur z již existujících pomocí takzvaných vodítek [4]. Vodítkem rozumíme tradičně dvojici obrázků. První z dvojice slouží k segmentaci informací v původní textuře. Typicky se jedná o segmentaci dělící původní texturu podle povahy objektů. Příkladem může být jednoduchá segmentace na pozadí a popředí. Druhý obraz ve dvojici pak definuje povahu textury generované. Funkční algoritmy následně v rámci svého běhu skládají novou texturu ze segmentů textury původní tak, aby přechody plynule navázaly mezi jednotlivými segmenty.



Obrázek 2.5: Demonstrace funkce nástroje ebsynth [9]

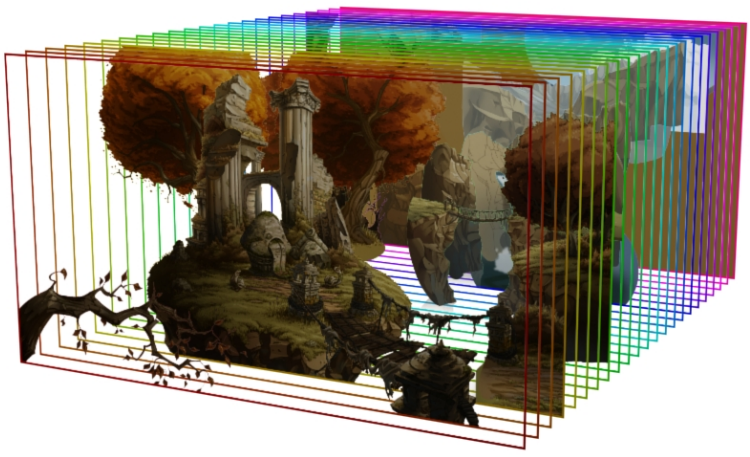
Na obrázku 2.5 je vyobrazena funkcionalita implementace takového syntetizátoru. Tato konkrétní implementace se nazývá A Fast Example-based Image Synthesizer, zkráceně Ebsynth, vyvíjená panem Ing. Ondřejem Jamriškou v rámci výzkumu na Českém vysokém učení technickém v Praze. V levé části se nachází zdrojová textura, ze které jsou brány zdrojové segmenty k vytvoření nové textury podle vodítka zobrazeného uprostřed. Jak je vidět, vodítko obsahuje tři rozdílné regiony. Barvou červenou je vybrán segment obsahující vodní hladinu. Barvu zelenou mají segmenty obrazu obsahující útesy. Posledním segmentem je vyplněno pozadí obsahující přechod barvy oblohy. Napravo je vyobrazena programem vygenerovaná nová textura.

Kapitola 3

Dynamická prezentace fotografií

Pohled na statické fotografie ne vždy dokáže podtrhnout rozsah zachycených snímků. Zároveň při demonstracích velkého množství fotografií může klesat pozornost diváků. Proto nejrůznější produkty prezentující fotografie vytváří animační přechody mezi snímky tak, aby zachovaly divákovu pozornost. Ve video produkci také mohou statické snímky působit rušivým dojmem. Proto nejen vědečtí pracovníci, ale i tvůrci v kinematografickém průmyslu se pokoušejí užití statických fotografií ve snímcích omezit nebo oživit dynamickým pohybem.

Jedním ze způsobů, jak oživit fotografii scenérie, je prezentace pohybu na pozadí. Klasicky takovéto snímky vznikají kombinací několika desítek fotografií pořízených na stejném místě a jejich následným spojením do jedné video sekvence. Tato video technika se nazývá Timelapse[1]. Typicky takto bývají prezentovány objekty na jejichž pozadí se míjejí mraky. Existují metody, které se snaží takovýto dynamický pohyb přinést do každé fotografie bez nutnosti zaznamenání velkého množství snímků. Jednou takovouto metodou je například metoda Animating Still Landscape Photographs Through Cloud Motion Creation [10]. Tato metoda detekuje oblast na fotografii na které se vyskytuje obloha. Následně z této oblasti získá informace o povaze mraků na fotografii a syntetizuje mraky, které si jsou vizuálně podobné. Takto vytvořené mraky poslouží k vytvoření Timelapse efektu z jedno statického snímku.



Obrázek 3.1: Demonstrace rozdělení obrazu do několika rovin [22]

[1] V české terminologii se můžeme setkat s pojmem časosběrný snímek

Další hojně využívanou metodou ve video průmyslu je vytvoření takzvaného paralax efektu na statické fotografii. Jedná se o umělé přidání hloubky do fotografie [6]. V současné době tento efekt bývá vytvářen ze statických fotografií ručně v nejrůznější paletě programů na tvorbu a střih videa. Princip vytvoření tohoto 2,5 dimenzionálního efektu primárně spočívá v separaci objektu zájmu a pozadí [20] a jejich vzájemným pohybem. Uměle vytvořený pohyb mezi objektem zájmu a pozadím může být buď zcela protichůdný, nebo ve stejném směru s rozdílnou rychlostí. Takto vytvořená sekvence se pozorovateli jeví jako prostorová. Problém při separaci objektu zájmu a pozadí je ten, že z původní fotografie není známo, jak vypadá pozadí za objektem zájmu. Tento úsek pozadí bývá z tohoto důvodu ručně retušován při vytváření paralax efektu. Princip takovéto prezentace snímku je možné umocnit rozdělením celého snímku do více než jedné hloubkové roviny, které se budou na výsledném snímku nezávisle pohybovat. Ilustrace rozdělení jedné scény do několika rovin je zobrazena na obrázku 3.1. Tento princip se také historicky využíval k vytvoření prostorového obrazu v animovaných filmech na začátku dvacátého století s využitím takzvané multiplátnové kamery[2] [22].

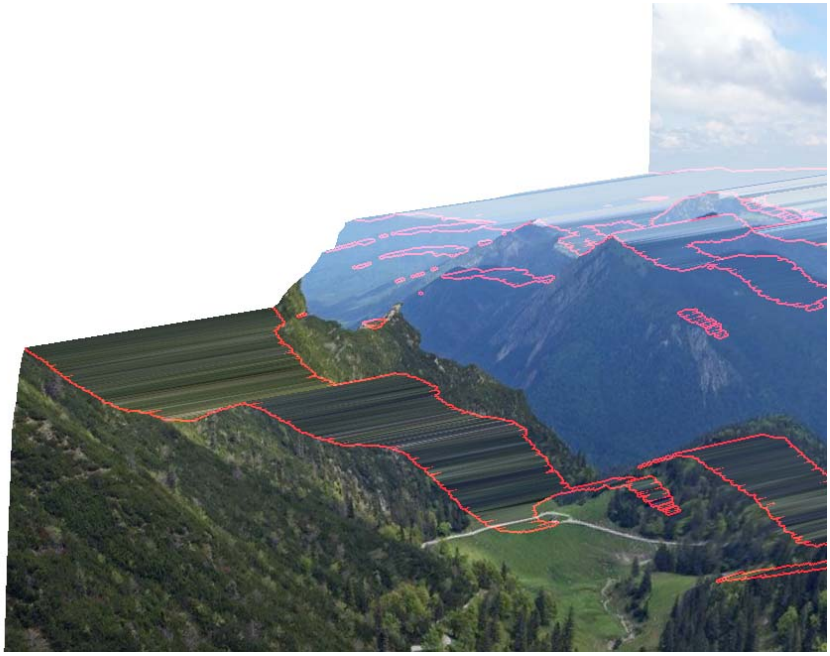
V roce 2018 na konferenci Adobe MAX firma Adobe představila ukázku jejich nové technologie, která se snaží výše zmíněný postup automatizovat. Tento pracovní projekt byl prezentován pod názvem Moving Stills [2]. Ze zveřejněných ukázek je patrné, že navržený systém prezentace fotografií pracuje s poměrně komplikovanějším rozložením hloubek než výše zmíněné návody. Je pravděpodobné, že zde dochází k využití hloubkové mapy při vytváření animačního přechodu.

[2] Z anglické multiplane camera využívané v animovaných filmech z dílen Walta Disneyho

Kapitola 4

Návrh a technologie

Tato kapitola se zabývá implementačními aspekty této diplomové práce. V této kapitole se nachází diskuse nad podobou životního cyklu programu, jehož úkolem je vytvořit ze statické fotografie a její hloubkové mapy video prezentaci. Hlavním úskalím při vytváření tohoto videa je vykreslení obrazu, který se schovává za objekty zobrazenými na fotografii. Implementace vede k vytvoření mračna bodů ze zdrojové fotografie a jejich následnému spojení pomocí indexů do trojúhelníků, jak je podrobně popsáno v kapitole věnující se vytváření prostorového modelu 5.2. Dojde-li k spojení bodů od sebe prostorově vzdálených do trojúhelníku, tak při jeho vykreslení vzniká roztažený fragment. Takovéto fragmenty působí vizuálně velmi nevzhledně a indikují, že v prostorovém modelu chybějí obrazové informace k této části modelu. Na tyto fragmenty bude dále poukazováno jako na díry v modelu. Na obrázku č. 4.1 jsou zobrazeny díry ohraničené červenou barvou. V textu této práce je dále popsáno, jak tyto díry v modelu detekovat a vyplnit. Těmto krokům a jejich popisu je věnována celá kapitola 5.5.



Obrázek 4.1: Vizualizace vzniklých fragmentů na pozici děr v modelu

4.1 Struktura systému

Aplikace je implementovaná jako desktopová. To znamená, že aplikace silně spoléhá na výpočetní výkon stroje na kterém je spuštěna. Primárně klade vysoké nároky na grafickou kartu, která je využita při vykreslování náhledu prostorového modelu, a následně i při akceleraci syntéz textur. Aplikace je rozdělena do několika tříd obhospodařujících jednotlivé dílčí problémy. Budeme-li postupovat hierarchicky, tak pro vytvoření videa ze statické fotografie je nutné provést tyto kroky:

Algoritmus 1: Návrh životního cyklu programu

```
1 Načtení a zpracování zdrojových dat
2 Vytvoření prostorového modelu
3 Nastavení perspektivy ve scéně
4 Určení počáteční a koncové pozice kamery
5 foreach snímek animace do
6     Vykreslit model a detekovat díry v modelu
7     Syntetizovat novou texturu scény
8     Uložit texturu jako snímek
9 Spojit uložené snímky do video sekvence
```

Z pohledu na cyklus v algoritmu č. 1 je na první pohled jasné, že pro vytvoření video sekvence ze statického snímku je nutné každý snímek při pohybu kamery syntetizovat. V ideálním případě by nově vytvořený snímek syntézou nebylo třeba nijak dále zpracovat. Dalším zpracováním se rozumí provádění výřezů, které by se algoritmicky spojily s vykresleným modelem obsahujícím díry. Teoreticky je možné takovéto nové textury generovat při vhodné kombinaci nastavení parametrů a zdrojových dat syntézy.

Primárním problémem tohoto přístupu je provázání návaznosti obrazu mezi snímky. Tento problém vzniká, protože jednotlivé spuštění syntézy je na sobě zcela nezávislé. Tudíž každý nově vytvořený snímek používá sice velmi podobná vstupní data, posunutá o krok kamery mezi snímky, ale chybí zde provázání s předchozími výsledky. Tato skutečnost může zapříčinit optickou nejednotnost zaplněných děr, což by ve výsledné video sekvenci mohlo působit problikáváním různých obrazců na pozicích děr. Tento problém je řešitelný tím, že by se místo vkresleného modelu s dírami použil jako vstup syntézy dalšího snímku předchozí snímek, nebo jejich kombinace. Toto řešení však není také zcela ideální, jelikož při snížení váhy vykresleného modelu při vstupu syntézy na úkor předchozímu snímku nebo jeho úplnému vynechání, začne docházet mezi jednotlivými snímky ke kumulaci chyb v obraze. Následně se může stát, že poslední snímek video sekvence vytvořené syntézou se bude výrazně lišit od pohledu kamery na původní model.

Dalším závažným nedostatkem této metody je její výpočetní složitost. Jelikož každé vyvolání syntézy je výpočetně a časově náročný úkon, vytvoření video sekvence tímto způsobem by zabralo nemalé množství času i přes skutečnost, že vyvolání syntézy je výpočetně akcelerováno paralelně na grafické kartě. Z důvodu výše zmíněných nevýhod algoritmu č. 1 nebylo s tímto návrhem více než teoreticky experimentováno.

Algoritmus 2: Implementovaný návrh životního cyklu programu

```
1  Načtení a zpracování zdrojových dat
2  Vytvoření prostorového modelu
3  Nastavení perspektivy ve scéně
4  foreach možný pohled na model do
5      Vykreslit model a detekovat díry v modelu
6      Syntetizovat novou texturu scény
7      Vytvořit záplatu na model ze syntetizované textury
8      Vložit záplatu do modelu
9  Určení počáteční a koncové pozice kamery
10 foreach snímek animace do
11     Vykreslit model
12     Uložit snímek
13 Spojit uložené snímky do video sekvence
```

Alternativou k návrhu č. 1 je návrh č. 2. Tento přístup se primárně odlišuje způsobem využití výsledků syntézy textur. Rozdíl spočívá v tom, že tento přístup vyplňuje díry přímo v modelu z textur vytvořených syntézou. Principiálně zde dochází k převodu fotografie na prostorový model a jeho následnému vykreslení na dvou-dimenzionální data, která jsou použita k syntéze. Dvou-dimenzionální výsledek je pomocí inverzních operací doplněn do prostorového modelu tak, aby zaplnil prostorovou díru. Takto je teoreticky možné díry v modelu vyplnit tak, aby v určitém pohledu vizuálně navazovaly na původní model. Principům operací nad prostorovým modelem a vytváření záplat je dále v textu věnovaná samostatná kapitola č. 5. Teoreticky by tedy měl jít prostorový model automatizovaně doplnit pomocí syntézy tak, aby při průletu kamery byly díry zaplněné obrazovými daty z výsledků syntézy.

Návrh č. 2 zároveň také nemá problém spojený s návazností jednotlivých snímků video sekvence, jelikož v průběhu jejich vytváření je model již neměnný a nad vyrasterizovanými snímky se neprovádějí žádné další úpravy. Další výhodou tohoto návrhu je také omezení počtu volání syntézy textur, což má za následek snížení doby trvání procesu vytváření celé video sekvence. Primárně z těchto důvodů výsledná implementace silně následuje tento návrh. Konkrétní rozdělení jednotlivých úkonů z tohoto návrhu do tříd je dále v textu popsáno v samostatné kapitole č. 4.5.

4.2 Načítání zdrojových dat

Při spuštění programu první modul zpracovává vstupní fotografii, její hloubkovou mapu a případně další dodatečné segmentační textury. Výstupem tohoto procesu je vytvořený prostorový model skládající se z mračna bodů a indexů propojující je v trojúhelníkovou síť. Tento celý proces je podrobně probrán v samostatné kapitole 5.2.

Za načítání a ukládání zdrojových obrazových dat je zodpovědná knihovna OpenCV. OpenCV (Open Source Computer Vision Library) [16] je svobodná a otevřená multiplatformní knihovna pro manipulaci s obrazem. Tato knihovna je uveřejněna pod otevřenou BSD licencí[1] k volnému užití jak v akademické, tak komerční sféře[2]. Má své rozhraní pro nejrůznější v současné době používané programovací jazyky jako jsou například C++, Java a Python. Je přenositelná napříč širokou škálou platforem z nichž nejvýznamnější jsou Windows, Linux, Mac OS a mobilní platformy iOS a Android. OpenCV bylo navrženo tak, aby co nejvíce využívalo veškeré výpočetní prostředky které jsou k dispozici. S prostředky se snaží hospodařit co nejefektivněji, a proto je vhodná i pro aplikace zpracovávající obraz v reálném čase.

Pro jednoduché načítání fotografií se může na první pohled jevit použití knihovny OpenCV jako značné mrhání jejím potenciálem. Tato knihovna byla zvolena právě pro svoji efektivitu a optimalizace. Základní operace nad obrazem jsou v knihovně zakořeněny od počátku, tudíž jsou stabilní a léty prověřené. Dalším bonusem využití této knihovny je to, že pro svoji rozšířenost má velice dobře napsanou dokumentaci, což usnadňuje práci na implementaci.

Knihovna OpenCV ukládá obrazová data do třídy `cv::Mat` od verze 2.0 [15]. Třída `Mat` je teoreticky rozčleněna na dvě datové části: hlavičku a ukazatel na obrazová data. Hlavička obsahuje informace o velikosti, dimenzích a datovém typu obrazu. Ukazatel na obrazová data pak ukazuje na místo v paměti, kde jsou obrazová data uložena v homogenní formě. Struktura uložených dat je principiálně znázorněna na obrázku 4.2. Tento konkrétní příklad ilustruje uložení tříkanálového obrazu o rozměrech šířky m a výšky n. Zajímavostí je, že knihovna standardně načítá třísložkové obrázky ve formátu BGR.

	Column 0			Column 1			Column ...			Column m		
Row 0	0,0	0,0	0,0	0,1	0,1	0,1	...	...	...	0, m	0, m	0, m
Row 1	1,0	1,0	1,0	1,1	1,1	1,1	...	...	...	1, m	1, m	1, m
Row ...	...,0	...,0	...,0	...,1	...,1	...,1	...	...	...	..., m	..., m	..., m
Row n	n,0	n,0	n,0	n,1	n,1	n,1	n,...	n,...	n,...	n, m	n, m	n, m

Obrázek 4.2: Ilustrace uspořádání obrazových dat v kontejneru `cv::Mat` [14]

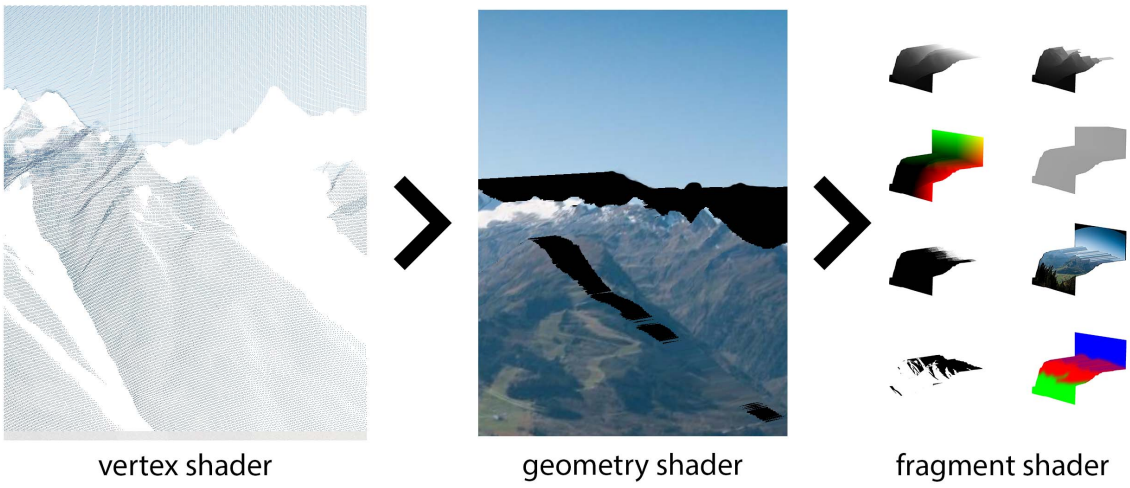
4.3 Vykreslování modelu

Pro vykreslování vytvořeného modelu ze vstupních fotografií bylo zvoleno API (Application Programming Interface) OpenGL (Open Graphics Library). OpenGL je [21] průmyslovým standardem na hardwarovou akceleraci vykreslovaní 2D a 3D geometrických útvarů na číslicových počítačích. Standard rozhraní OpenGL je navržen tak, aby API bylo nezávislé

[1] Dostupná online na: https://opencv.org/license/

[2] Jedná se o jednu z mnoha licenčních šablon pro otevřený komunitní software. Přehled typicky používaných lze nalézt: https://opensource.org/licenses/

na hardwaru, čímž je dosaženo přenositelnosti kódu na rozmanité platformy. V současné době spravuje standard konsorcium Khronos Group Inc v aktuální verzi 4.6. Samotné vykreslování je rozděleno do několika kroků, které popisuje standard OpenGL jako zobrazovací pipeline[3]. Tento řetězec zřetězeně zpracovává vstupní data podle aktuální konfigurace. Kromě konfigurace stavu řetězce programátor ovlivňuje výsledný vzhled zobrazovaných těles pomocí takzvaných shaderů.



Obrázek 4.3: Ilustrace průběhu dat ve vykreslovacím řetězci

Prvním programovatelným shaderem je vertex shader. Ten zpracovává jednotlivé vrcholy určující primitiva (vertexy). Zde je možné provést nejrůznější prostorové transformace s vykreslovanými objekty bez narušení původních dat. Dokáže však pouze modifikovat jednotlivé body, nikoliv celá primitiva. Jedná se o kód, který se vykonává paralelně pro každý vrchol nezávisle na ostatních. V implementaci je využit k pohybu kamery kolem modelu a perspektivním operacím popsaných v kapitole 5.3.

Dalším zcela volitelným[4] programovatelným krokem vykreslovacího řetězce je část nazývaná teselační shader. Ten se skládá ze dvou programovatelných částí kontrolního a vyhodnocovacího shaderu. Typicky se teselační shader využívá při vykreslování k rozdělení primitiv na více jemnější. Příkladem použití může být vyhlazení povrchu koule rozdělením trojúhelníků na jejím povrchu na více menších tak, aby opticky vypadala méně hranatá. Tato část vykreslovacího řetězce nebyla ve finální implementaci využita.

Následujícím využitým krokem v řetězci je geometry shader. Jedná se o nepovinnou součást grafického řetězce. Je to poslední krok před rasterizací primitiv. Pouští se pro každé primitivum paralelně. Na rozdíl tedy od Vertex shaderu má přístup ke všem vrcholům primitiva. V průběhu geometry shaderu je možné při zpracování primitiv měnit jejich typ tak, že je na výstupu odlišné primitivum než bylo původně na vstupu. Případně je možné některá primitiva úplně vyloučit z následného vykreslovacího procesu. V implementaci je využit k detekci děr v modelu. Princip této detekce je více diskutován ve vlastní kapitole č. 5.4.

[3]V české literatuře se setkáváme s názvem vykreslovací řetězec
[4]Programátor jej nemusí zcela využít

Poslední programovatelnou částí grafického řetězce je fragment shader. V tomto shaderu je definována barva jednotlivých bodů objektu po rasterizaci. Výslednou barvu je zde možné skládat z několika textur a ovlivňovat ji pozicí vůči světlu. Výsledky rasterizace se ukládají do takzvaného Framebuffer objektu, který si data drží v textuře. V základním nastavení se využívá tato textura k vykreslování výsledku rasterizace na obrazovku. V rámci aplikace je definován vlastní Framebuffer objekt, který umožňuje ukládání výstupů do několika textur najednou. Toho je využito pro generování několika informací z pohledu na model. Výstupem fragment shaderu jsou textury obsahující model s nanesenou původní texturou, s přidanými texturami, s původní hloubkovou mapou, s výsledkem detektoru fragmentů a s nanesenými původními souřadnicemi. Tyto výstupy v aplikaci jsou vizualizovány dalším vykreslovacím procesem, nebo je možné je uložit do složky export.

4.3.1 Podpůrné knihovny pro práci s OpenGL

OpenGL je pouze průmyslovým standardem zabývajícím se vykreslováním na grafické kartě. Standard neřeší zprávu oken v rámci operačního sytému. Pro tyto účely byla využita knihovna GLFW. Jedná se o otevřenou knihovnu napsanou v jazyce C vytvářející tenkou vrstvu mezi operačním systémem a kontextem API OpenGL. Knihovna vytváří pro programátora přístupový bod ke grafickému kontextu a zároveň obsluhuje vytváření systémových oken. K jednotlivým oknům dále umožňuje definovat obsluhu událostí ze vstupních periférií (jako například pohyb kurzoru myši, stisk klávesy na klávesnici a mnohé další) pomocí takzvaných callback funkcí. Tato knihovna je volně dostupná pod Zlib licencí [5].

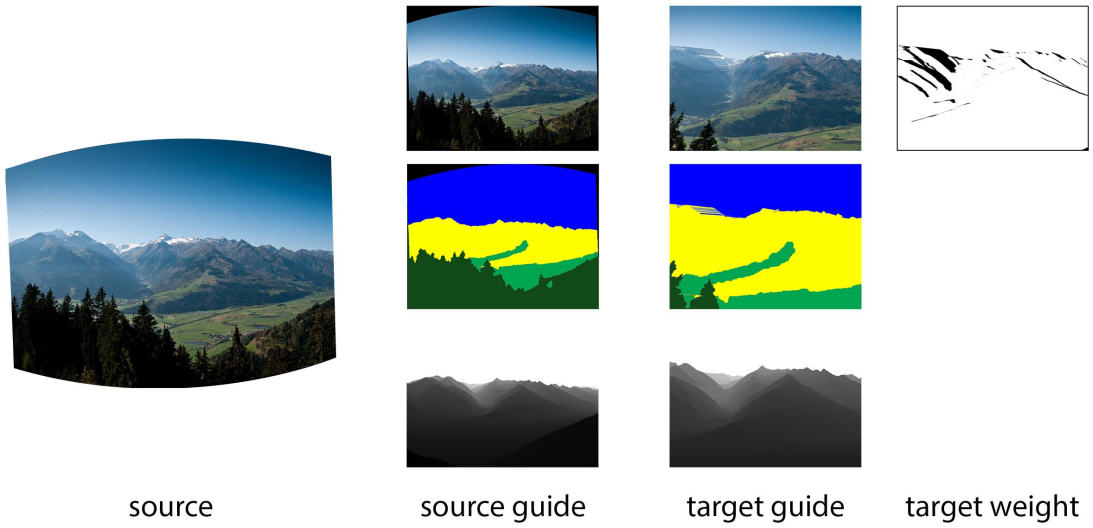
Knihovna OpenGL Extension Wrangler Library, zkráceně GLEW, zpřístupňuje programátorovi OpenGL funkce a jejich rozšíření pomocí ukazatelů na funkce v OpenGL API. Samotná knihovna GLEW slouží k načtení funkcí OpenGL API za běhu aplikace (při jejím spuštění). Proto se využívá jako komunikační rozhraní, jelikož většina funkcí OpenGL API není známa při překladu aplikace a k jejich načtení dojde až při spuštění grafického kontextu. V kombinaci s dříve zmíněnou knihovnou GLFW vytváří minimální mezivrstvu pro programování grafické aplikace při použití OpenGL API. Tato knihovna je zdarma dostupná pod BDS licencí[6].

OpenGL Mathematics, zkráceně GLM, je otevřenou knihovnou napsanou v jazyce C++. Knihovna se skládá pouze z hlavičkových souborů, a proto se přikládá k implementaci vždy při překladu. Knihovna GLM implementuje datové typy definované ve standardu OpenGL Shading Language (GLSL), což je podmnožina standartu OpenGL. Tím tato knihovna přináší datové typy (například vektory a matice) využívané grafickými kartami na procesor počítače. Při využíti těchto typů je zaručeno pomocí maker, že překladač neprovede prostorovou optimalizaci nad těmito daty a vždy budou v aplikaci uložena přesně tak, jako na grafické kartě. Další výhodou použití této knihovny je, že obsahuje bohatou funkcionalitu nad těmito datovými typy. V grafických aplikacích je také ceněna možnost vytváření transformačních matic, jejichž využití je popsáno v kapitole 5.3. Celá knihovna je k dispozici pod Happy Bunny licencí[7].

[5] Dostupné na: https://www.glfw.org/license.html
[6] Více informací na: https://github.com/nigels-com/glew#copyright-and-licensing
[7] Dostupné online na: http://glm.g-truc.net/copying.txt

4.4 Syntéza textur

Jak je popsáno v kapitole zabývající se problematikou prostorového modelu 5, tak při pohybu kamery mimo osu Z dochází k vykreslení fragmentů na pozici děr. Proto je nutné buď celý obraz syntetizovat, nebo nahradit konkrétní fragmenty v obraze. K vypočtení nové textury poslouží API Ebsynth. Princip jeho funkcionality již byl popsán v kapitole 2.4.



Obrázek 4.4: Návrh systému syntézy textur

Jako vstupní textura syntézy je použit kolmý přímý pohled na model. Ta je vyobrazena na obrázku 4.4 vlevo. Teoreticky je možné využít přímo vstupní fotografii, ale při použití tohoto způsobu je zaručeno, že všechny použité vodící textury jsou na pixel přesně zarovnány se zdrojem, jelikož k jejich rasterizaci byly použity stejné vstupní body. To napomáhá zlepšení přesnosti syntetizovaných výsledků. Jako primární vodící kanál poslouží hloubková mapa původní fotografie nanesená na model. Z tohoto vodícího kanálu získá syntetizační API povědomí o horizontech a pozici pozadí na fotografii. Druhým vodítkem syntézy může být samotný model s nanesenou původní texturou, toto zobrazení lze nazývat reprodukce obrazu. V podstatě se jedná od tři závislé vodící kanály vytvořené každou barevnou složkou. S tímto vodítkem je možné experimentovat, zda nepostačí využít například kanál pouze jeden, nebo zkombinovat všechny barevné složky do jedné intenzitní. Toto vodítko samozřejmě obsahuje také fragmenty, které je chtěné syntézou nahradit. Proto je vodítko doplněno druhou texturou obsahující výsledek detektoru děr, zobrazeno 4.4 vpravo. Váhy jsou vysoké v místech, kde dochází k vykreslení původních bodů modelu. Oproti tomu hodnota 0 je v místech fragmentů. Tato technika by teoreticky měla značně vylepšit kvalitu syntézy tím, že poznačí body, které není nutné přepočítávat. Jelikož každý bod modelu má své texturovací koordináty, viz 5.1, tak je možné syntetizační API podpořit dalšími informacemi o původní fotografii. Na obrázku 4.4 je ilustrováno užití segmentace obrazu. Teoreticky je možné takovýchto textur doplnit i více. Optimální podoba parametrů na spuštění syntézy byla konzultovaná přímo s autorem API Ebsynth. Finálně použité hodnoty jsou k nahlédnutí na výpisu č. 4.1.

```
int patchSize = 7;
float uniformityWeight = 1000;
int numSearchVoteIters = 12;
int numPatchMatchIters = 4;
int stopThreshold = 5;
int extraPass3x3 = 0;
```

Výpis 4.1: Parametry syntézy použité v implementaci

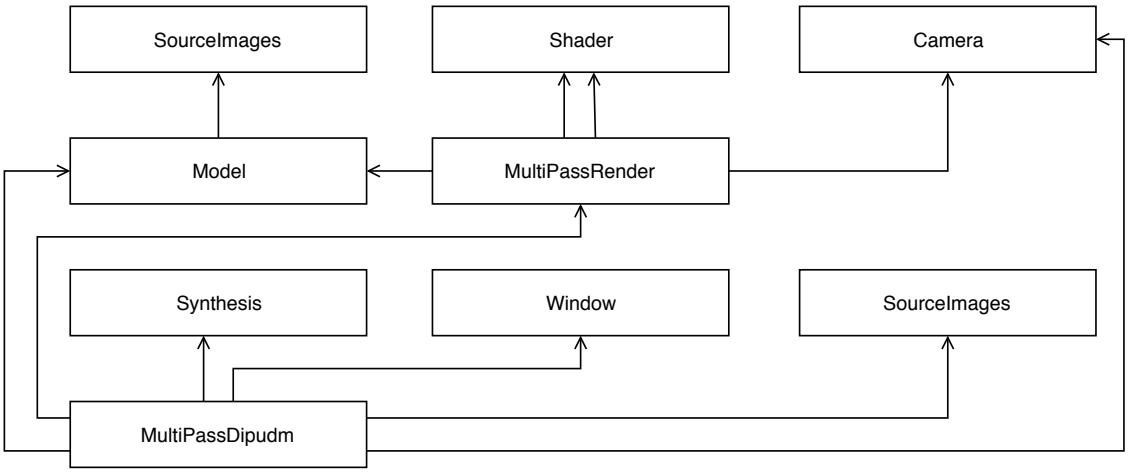
Jak již bylo zmíněno dříve, syntetizační API vytváří novou texturu z fragmentů původního obrazu. Tyto fragmenty se nazývají záplaty (z anglického Patch) [3]. API v rámci běhu syntézy algoritmicky vybírá tyto záplaty hierarchicky ve stromové struktuře od největších k nejmenším tak, aby v nově vytvořené textuře byly přítomny pokud možno co nejucelenější segmenty z původního obrazu. K výběru těchto záplat dochází systematickým hledáním nejvhodnější záplaty ze vstupního obrazu pomocí zmíněných vodítek. Do výsledného obrazu se pak vkládá záplata mající nejmenší chybové ohodnocení vyjádřené níže zobrazeným vzorcem (4.1). Ze vzorce je patrné, že výběr záplaty je možné parametricky ovlivnit váhou jednotlivých vodítek. Například ve výsledné implementaci je vhodné nejvyšší váhu přidělit vodítku obsahujícímu hloubkovou mapu. Výhodou zvoleného syntetizačního API je přítomnost váhové textury pro cílové vodítko. Tato textura bývá označovaná jako váhová nebo také modulační. Výstup detektoru děr je ideální texturou, kterou lze naplnit tato modulační data.

$$Err(x_s, y_s, x_t, y_t) = \sum_{c_i=0}^{C} \left[W(c_i) * \left(Target(x_t, y_t, c_i) - Source(x_s, y_s, c_i) \right)^2 * M(x_t, y_t, c_i) \right] \tag{4.1}$$

Rovnice (4.1) vyjadřující chybu záplaty v syntetizovaném obraze:

- x_s a y_s jsou souřadnice záplaty ve zdrojovém vodítku,
- x_t a y_t jsou souřadnice záplaty v cílovém vodítku,
- C je počet kanálů použitého vodítka,
- $W(c_i)$ vrací hodnotu váhy vodícího kanálu,
- $Target(x_t, y_t, c_i)$ vrací hodnotu cílového vodítka kanálu c_i na pozici x_t a y_t,
- $Source(x_s, y_s, c_i)$ vrací hodnotu zdrojového vodítka kanálu c_i na pozici x_t a y_t,
- $M(x_t, y_t, c_i)$ vrací hodnotu váhy cílového vodítka kanálu c_i na pozici x_t a y_t.

4.5 Třídní rozložení aplikace



Obrázek 4.5: Diagram tříd

Window

Třída `Window` zajišťuje obsluhu GLFW knihovny. Stará se tedy o úspěšné otevření a zavření hlavního aplikačního okna obsahujícího OpenGL kontext. S tím je spojena i přítomnost obsluhy ukončení běhu hlavního vykreslovacího cyklu. Dále se zde nachází obsluha vstupu z klávesnice. Klávesnice se v aplikaci využívá k ovládání volného pohybu kamery v prostoru kolem modelu. Výsledek aktuálně stisknutých kláves se ukládá v podobě dvou vektorů určujících směr a rotaci pohybu. Třída `Window` při svém vytvoření obdrží informace důležité k vytvoření hlavního okna aplikace. Primárně se jedná o počáteční velikost okna a jeho nadpis. Dále jsou pak přidány parametry využívající funkcionalitu knihovny GLFW.

Shader

Třída `Shader` slouží k vytvoření shader programu spustitelného na grafické kartě sloužícího k vykreslování. Tento výsledný program překládá ze zdrojových kódu jednotlivých shaderů. Implementace vyžaduje dva standardně povinné shadery vertex a fragment. Volitelně je možné přiložit i zdrojový kód geometry shaderu. Alternativně je také možné u třídy `Shader` vyvolat konstruktor obdržující pouze cesty k jednotlivým souborům obsahující zdrojové kódy. V rámci obsluhy shader programu je v třídě `Shader` zakomponováno asociativní pole obsahující indexy uniformních proměnných tak, aby bylo co nejvíce eliminováno volání funkce `glGetUniformLocation`, které značně zpomaluje vykreslovací cyklus.

Camera

Jak název třídy `Camera` napovídá, tato třída obsluhuje parametry virtuální kamery pohybující se v okolí vytvořeného prostorového modelu. Obsahuje informace nutné k vytvoření pohledových transformačních matic, které jsou použity při vykreslování. Konkrétní popis využitých transformací je v textu rozepsán v kapitole 5.3. Nejdůležitější informace, které si třída `Camera` uchovává, je pozice a směr natočení virtuální kamery. Z těchto informací je možné za použití funkcí knihovny GLM vytvořit odpovídající transformační matice. Dále

jsou zde přítomny interpolační metody, které dokáží lineárně zanimovat její pohyb mezi dvěma body.

SourceImages

V rámci vytvoření třídy `SourceImages` dochází k načítání zdrojových obrazových informací a jejich zpracování. Podrobnější popis předzpracovaní obrazových informací před vytvořením prostorového modelu je k dispozici na začátku kapitoly zabývající se vytvořením modelu č. 5.2. Veškeré výsledky zpracování obrazových dat jsou uložené ve veřejně přístupných třídních proměnných typu `cv::Mat`. Kromě operací spojených se zpracováním vstupních dat se nachází v této třídě také metoda sloužící k načítání souborů ve formátu `.pfm`. Dále je zde také velmi užitečná statická metoda `onMouseReadPixel` využitelná jako callback při poklikání na obraz vizualizovaný knihovnou OpenCV, která vypisuje při stisknutí levého tlačítka myši na souřadnicích pixelu jeho barevnou hodnotu do terminálu. Tato funkce byla primárně využívána při vývoji samotné implementace.

Model

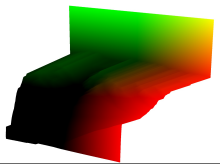
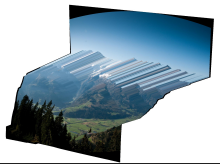
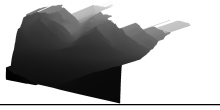
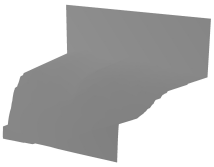
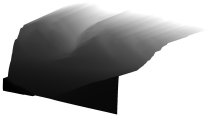
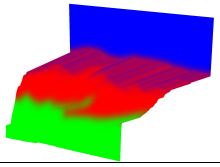
Primárním účelem třídy `Model` je uchovávání prostorových informací modelu. Obsahuje metody, které dokáží ze zpracovaných obrazových informací ze třídy `SourceImages` vytvořit prostorový model skládající se z mračna bodů a indexy propojující je do trojúhelníkové sítě. Proces vytváření modelu je podrobněji popsán dále v textu v kapitole č. 5.2. Dále také třída `Model` obsahuje metody sloužící k vytvoření záplat modelu, jejichž princip je popsán v kapitole 5.5. K ukládání dat jednotlivých záplat dochází odděleně od původního modelu, přičemž ale data záplat využívají stejnou datovou strukturu jako data modelu, aby je bylo možné zpracovávat stejným způsobem při vykreslování. Tato struktura je zmíněna na začátku kapitoly č. 5.1 věnující se prostorovému modelu.

MultiPassRender

Třída `MultiPassRender` slouží k vykreslování prostorového modelu z třídy `Model`. Jelikož pro přesnější výsledky syntézy je možné na model nanést dodatečné informace, je vykreslování modelu implementováno pomocí techniky Multiple Render Targets tak, aby bylo možné u syntézy využít vodítko s více než čtyřmi kanály jedné textury. Implementace využívá všech osm možných výstupních textur při vykreslování modelu. Jejich obsah by se jistě dal zredukovat, ale pro snazší manipulaci s daty byla data v nich uspořádána podle tabulky č. 4.2. K samotnému vykreslení dat pro uživatele následně dochází v druhém zřetězeném cyklu, který využívá na vstupu oněch osm textur. K vykreslování jsou využity dva shader programy z třídy `Shader`.

Synthesis

Primární úlohou třídy `Synthesis` je příprava dat pro vyvolání syntézy nové textury z API Ebsynth a následné zpracování výsledků tak, aby bylo možné z nich vytvořit záplaty modelu na model. Třída vytváří multikanálová vodítka buď podle předpřipravené konfigurace, nebo konfigurace určené uživatelem z grafického rozhraní. Nově vytvořené textury syntézou je možné samostatně za běhu aplikace ukládat.

Obrázek	`TextureType`	Popis
	`TARGET_COORDINATES`	Jediná textura datového typu float, která slouží k uložení původních koordinát jednotlivých bodů mračna bodů. Využití této textury je v textu dále popsáno v kapitole č. 5.5.3
	`TARGET_TEXTURE`	Textura obsahující barevné informace z modelu. Dále se v tomto textu můžeme také setkat s pojmem reprodukce obrazu. Tento pojem bývá použit k referenci na takovéto vykreslení modelu.
	`TARGET_DEPTH_MAP`	Textura obsahující hloubkovou mapu po operacích popsaných v kapitole č. 5.4.3.
	`TARGET_DEPTH_MAP _NORMALIZED`	Jedná se o texturu obsahující normalizovaná data hloubky z Z-bufferu vykreslovacího řetězce OpenGL. Prvotní experimenty se syntézou textur však nevedly k úspěšným výsledkům, a proto tato textura nemá v programu další hlubší využití. Neúspěchy experimentů byly zapříčiněny tím, že hloubka uložená v textuře je vypočítána v závislosti na vzdálenosti modelu od kamery.
	`TARGET_DEPTH_MAP _ORIGINAL`	Jedná se o texturu obsahující původní normalizovanou hloubku modelu. V podstatě je to Z složka z textury `TARGET_COORDINATES`. Tato informace byla přidána do samostatné textury proto, aby mohla být lehčeji využita jako vstup syntézy. V kapitole č. 5.4.3 je popsán primární rozdíl od textury `TARGET_DEPTH_MAP` a jak se tento rozdíl projevuje na syntéze textur.
	`TARGET_HOLES`	V této textuře je uložený výsledek detektoru děr v modelu. Princip detekce děr je dále v textu popsán v kapitole č. 5.4.
	`TARGET_EXTRA_1`	Zde se nachází výsledek reprodukce první přidané textury, která poslouží jako pomocné vodítko syntézy textur.
	`TARGET_EXTRA_2`	Slouží stejně jako předchozí textura k uložení druhé přidané textury.

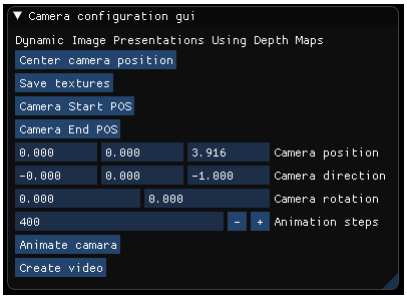
Tabulka 4.2: Přehled výstupních textur.
Výstupní textury jsou k nahlédnutí v příloze A.

MultiPassDipudm

Třída `MultiPassDipudm` slouží k propojení celého programu. Obsahuje rutiny k přenášení informací mezi jednotlivými třídami, jako jsou například metody na aktualizaci pohybu kamery (`Camera`) podle vektoru stisknutých kláves (`Window`), nebo přenos dat k vyvolání syntézy z výsledků vykreslování. Třída `MultiPassDipudm` je využita k vytvoření celého aplikačního kontextu a řídí celý životní cyklus. Pro své vytvoření vyžaduje minimálně dvě cesty ke zdrojovým souborům, a to ke vstupní fotografii a její hloubkové mapě. V metodě `run()` je implementována hlavní programová smyčka, která je ukončena přerušením z událostí grafického okna aplikace. A v poslední řadě třída `MultiPassDipudm` obsahuje metody vykreslující uživatelské grafické prostředí, jemuž je věnována celá následující kapitola č. 4.6.

4.6 Uživatelské rozhraní aplikace

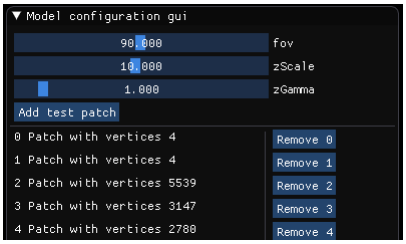
Pro vykreslování grafického uživatelského prostředí v rámci okna OpenGL byla použita knihovna Immediate Mode Graphical User interface (zkráceně ImGui). Jedná se o velice jednoduchou knihovnu fungujcí v takzvaném okamžitém módu[8]. To se implementačně liší tím, že knihovna ImGui obsluhuje akce vyvolané uživatelem okamžitě, nikoliv jak je obvyklé u knihoven na grafické uživatelské prostředí pomocí takzvaných callback funkcí. Knihovna je napsána v jazyce C++ a do implementace se vkládá přímo s celými zdrojovými kódy za překladu. Programátor pouze musí vybrat vhodné adaptéry na použité technologie vytváření grafického kontextu. V implementaci se jedná koknrétně o adaptér na OpenGL a GLFW. Knihovna je volně k dispozici pod MIT licencí [9].



Obrázek 4.6: Vizualizace grafického uživatelského rozhraní z výsledné aplikace obhospodařující parametry kamery

Na obrázku č. 4.6, jehož nezmenšená verze je k nalezení v příloze C.1, je zobrazeno výsledné rozložení ovládacích prvků na panelu navázaného na třídu `Camera`. Jsou zde tedy tlačítka sloužící k nastavení animace virtuální kamery. K tomu slouží tlačítko určující počáteční a konečnou pozici kamery. Tyto hodnoty jsou převzaty z aktuální pozice virtuální kamery v prostředí. Hodnoty pozice a směru natočení je možné sledovat v jednotlivých políčcích k tomu určených. Manuální změna hodnoty v nich zobrazených je zpětně navázaná na virtuální kameru, takže je možné kameru v prostoru absolutně pozicovat, i když to není zcela intuitivní řešení. Tato funkcionalita je nutná, pokud chceme přesně porovnávat stejnou animaci kolem rozdílných modelů.

[8] Z anglického Immediate Mode
[9] Dostupné online na: https://github.com/ocornut/imgui/blob/master/LICENSE.txt

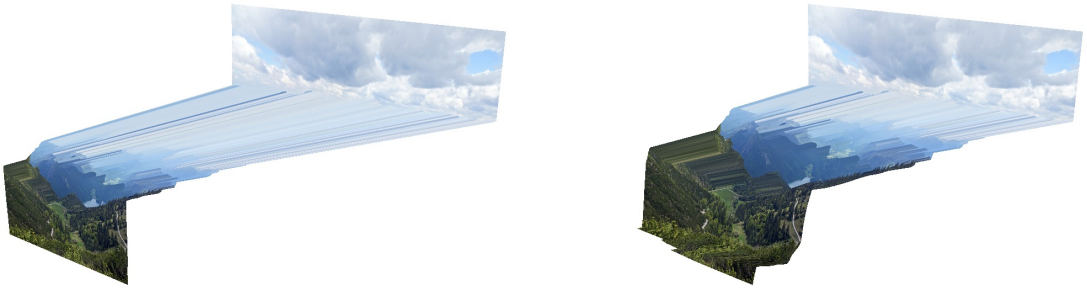


Obrázek 4.7: Vizualizace grafického uživatelského rozhraní z výsledné aplikace obhospodařující prostorový model

Obrázek č. 4.7, jehož plná verze je k nalezení v příloze C.2, zobrazuje výsledné rozložení ovládacích prvků na panelu navázaného na třídu `Model` a její vykreslení. Ve vrchní části panelu Model se nacházejí posuvníky nastavující perspektivní vlastnosti modelu. Více o problematice vytvoření perspektivního dojmu při pohybu kamery kolem modelu je k dispozici v kapitole věnující se rekonstrukci perspektivy 5.3. Jedním z těchto posuvníků je i posuvník na takzvanou gamma korekci hloubky. Principiálně se jedná o převzatou techniku používající se na korekci fotografií aplikovanou na Z souřadnici modelu podle vzorce (4.2).

$$z_{nov} = z_{orig}^{\frac{1}{gamma}} \tag{4.2}$$

Pomocí této techniky je možné mnohem lépe využít rozsah hloubky modelu <0;1> tak, aby bylo možné pozorovat rozdíly hloubky mezi jednotlivými horizonty vrcholků na modelu. K normalizaci rozměrů modelu dojde v průběhu jeho vytváření na úrovni zpracování vstupní hloubkové mapy, jak je dále v textu popsáno v kapitole č. 5.2. Aplikace této transformace probíhá až za běhu programu dynamicky pouze v průběhu vykreslovacího procesu. Na obrázku 4.8a je zobrazen model s nastavením gamma korekce na hodnotu 1. Tedy jak vyplývá ze vzorce (4.2), nedochází v tomto případě při vykreslení modelu k žádné modifikaci hloubky zapříčiněné hodnotou gamma. Obrázek č. 4.8b ukazuje mnohem lepší využití rozsahu hloubky po nastavení hodnoty gamma na hodnotu 2,5.



(a) Původní hloubka modelu

(b) Zvýraznění profilu hloubky modelu pomocí gamma korekce

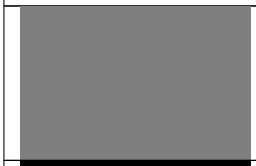
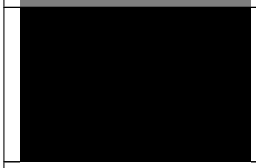
Obrázek 4.8: Rozdílné využití rozsahu hloubky

Obrázek 4.9: Vizualizace grafického uživatelského rozhraní z výsledné aplikace obhospodařující nastavení syntézy textur

Největším panelem grafického uživatelského prostředí, viz obrázek č. 4.9, je panel obsahující prvky, které nastavují parametry použitých vodících kanálů syntézy. Obrázek 4.9 je nezmenšený k nahlédnutí v příloze C.3. V horní části panelu se nachází nastavení použitých vodítek generovaných v průběhu vykreslování. Jednotlivé kanály výstupních textur (viz tabulka 4.2) se dají nastavit jako vodítka syntézy. K těmto vodítkům je možné nezávisle nastavit váhy a přiřadit modulační váhovou masku. Princip, jakým tyto váhy ovlivňují výsledky syntézy, je popsán dříve v textu vzorcem (4.1). Pro hladší navázání modulačních vah je na výstup detektoru děr aplikována široká paleta morfologických operací. Mezi použité morfologické operátory se řadí gaussovské rozmazání, dilatace a eroze. Všechny tyto operátory byly aplikovány ve dvou velikostech jádra 7 a 17 pixelů. Dále se ve výběru váhových masek nachází vhodné kombinace dříve zmíněných operací. Kompletní přehled implementovaných masek je níže v tabulce 4.3. Ilustrativní obrázky z této tabulky jsou k nahlédnutí v příloze B. Tato sada různých váhových masek vznikla primárně proto, aby bylo možné experimentovat se syntetizačním API. Tak aby bylo možné co nejplynuleji navázat nové fragmenty na pozici děr v reprodukci modelu.

Kromě syntézy nových obrazových informací k zaplnění je možné experimentovat přímo i se syntézou nové hloubkové mapy z různých pohledů na model. Tato možnost byla přidána primárně z experimentálních důvodů, aby bylo možné otestovat další možnosti využití technik syntézy textur.

Ve spodní části panelu se nacházejí tlačítka obsluhující jednotlivé kroky syntézy, jako je nastavení vodítek z aktuálního pohledu kamery na model a spuštění samostatné syntézy. Výsledná implementace umožňuje jednotlivé výsledky syntézy ukládat pro případné další zkoumání či užití.

Obrázek	`Synthesis::Mask`	Popis
	`WHITE`	Maska obsahující samé maximální hodnoty. Tato maska je využita u vodících kanálů, u kterých není požadováno brát v úvahu díry v modelu.
	`SCALAR_127`	Podobně jako předchozí maska je i tato vyplněna jednolitou hodnotou. Tato maska vznikla jako kontrolní tak, aby mohla být ověřena funkčnost syntetizačního API.
	`BLACK`	Stejně jako předchozí maska, je i tato vyplněna jednolitou hodnotou. Tato maska vznikla jako kontrolní tak, aby mohla být ověřena funkčnost syntetizačního API.
	`HOLES`	Jedná se přímý výstup detektoru děr implementovaného podle 5.4.3.
	`HOLES_INVERSE`	Tato maska obsahuje binární inverzi masky `HOLES`.
	`BLUR_7X7`	Jedná se o masku `HOLES` upravenou gaussovským rozmazáním o velikosti jádra 7 pixelů.
	`BLUR_17X17`	Jedná se o masku `HOLES` upravenou gaussovským rozmazáním o velikosti jádra 17 pixelů.
	`DILATED_7X7`	Tato maska obsahuje výsledek morfologické operace dilatace provedené nad maskou `HOLES` pomocí kruhového jádra o průměru 7 pixelů.
	`DILATED_17X17`	Tato maska obsahuje výsledek morfologické operace dilatace provedené nad maskou `HOLES` pomocí kruhového jádra o průměru 17 pixelů.
	`ERODED_7X7`	Tato maska obsahuje výsledek morfologické operace eroze provedené nad maskou `HOLES` pomocí kruhového jádra o průměru 7 pixelů.
	`ERODED_17X17`	Tato maska obsahuje výsledek morfologické operace eroze provedené nad maskou `HOLES` pomocí kruhového jádra o průměru 17 pixelů.

Obrázek	`Synthesis::Mask`	Popis
	`BLUR_AFTER _DILATATION_7X7`	Jedná se o masku `DILATED_7X7` upravenou gaussovským rozmazáním o velikosti jádra 7 pixelů.
	`BLUR_AFTER _DILATATION_17X17`	Jedná se o masku `DILATED_17X17` upravenou gaussovským rozmazáním o velikosti jádra 17 pixelů.
	`BLUR_AFTER _EROSION_7X7`	Jedná se o masku `ERODED_7X7` upravenou gaussovským rozmazáním o velikosti jádra 7 pixelů.
	`BLUR_AFTER _EROSION_17X17`	Jedná se o masku `ERODED_17X17` upravenou gaussovským rozmazáním o velikosti jádra 17 pixelů.

Tabulka 4.3: Přehled použitelných váhových masek při syntéze textur.
Detalnější pohled na jednotlivé masky je k nahlédnutí v příloze B.

Kapitola 5

Práce s prostorovým modelem

V této kapitole je podrobně popsán proces vytváření prostorového modelu ze zdrojového obrazu a jeho hloubkové mapy. V textu jsou diskutovány rozdílné postupy pro vytvoření modelu tak, aby bylo z následujícího textu patrné, proč je ve finální implementaci zvolen konkrétní postup u všech důležitých operací. Následně tato kapitola popisuje princip, jakým je model vykreslen tak, aby vynikla hloubka ve vytvořeném modelu. Poslední část této kapitoly se věnuje postupu vytváření prostorových záplat modelu z výsledků syntézy obrazu.

5.1 Struktura dat modelu

Prostorový model se skládá z mračna bodů, které vznikne ze zdrojové fotografie, jak je popsáno v následující kapitole 5.2. Mračno bodů je definováno vrcholy, a ty jsou uloženy v následující struktuře Vertex 5.1. Model je následně zobrazován pomocí vykreslovacího řetězce OpelnGL. Proto každý vrchol obsahuje svoji pozici. Pro uložení pozice byl zvolen čtyřrozměrný vektor, i když pro definování bodu v třírozměrném prostoru postačuje vektor třírozměrný. Přidání čtvrté položky v hodnotě 1 je nutné k prostorovým transformacím a není zde důvod ji rovnou nemít připravenou vzhledem k faktu, že aplikace celkově nevyužívá veškeré množství paměti grafické karty. Další výhodou tohoto řešení je zarovnání paměti na bloky o velikosti násobku č. 4, což je výhodné z hlediska architektur většiny standardních grafických karet. Ze stejných důvodů má čtyři složky i barva bodu v podobě RGBA. Složka barvy u jednotlivých bodů by mohla být zcela vynechána, jelikož každý bod má své texturovací koordináty, ale pro usnadnění práce a experimentování s modelem bylo výhodnější barevnou složku ponechat. Texturovací koordináty slouží ve výsledné implementaci primárně k mapovaní vodících textur na model. Poslední dvě položky `depth` a `holeColor` slouží pro uložení výstupních hodnot při zpracovaní v grafickém řetězci, jak je podrobněji popsáno dále v kapitolách 5.4 a 5.3.

```
using Vertex = struct
{
    float pos[4];
    float color[4];
    float texCord[2];
    float depth;
    float holeColor;
};
```

Výpis 5.1: Data jednoho bodu ze souboru model.h

5.2 Vytvoření modelu

Vstupem programu je fotografie a její hloubková mapa. Pro zlepšení kvality syntézy textur v modelu, které vyplňují takzvané díry v modelu, je možné přidat až dvě dodatečné obrazové textury. Typicky se může například jednat o segmentaci obrazu, která slouží jako další vodítko pro syntézu. Přidáním takovéto další informace může dojít ke zlepšení kvalit syntézy. Vytvoření samotného prostorového modelu předchází načtení a následné zpracování zdrojových obrazových informací. Pro úspěšné vytvoření modelu je nutné provést následující kroky:

Algoritmus 3: Zpracování vstupních obrazových informací

1 Načtení vstupní fotografie

2 Převedení formátu vstupní fotografie

3 Načtení hloubkové mapy

4 Extrakce pozadí

5 Korekce hloubkové mapy

6 Změna velikosti hloubkové mapy

7 Načtení dodatečných textur

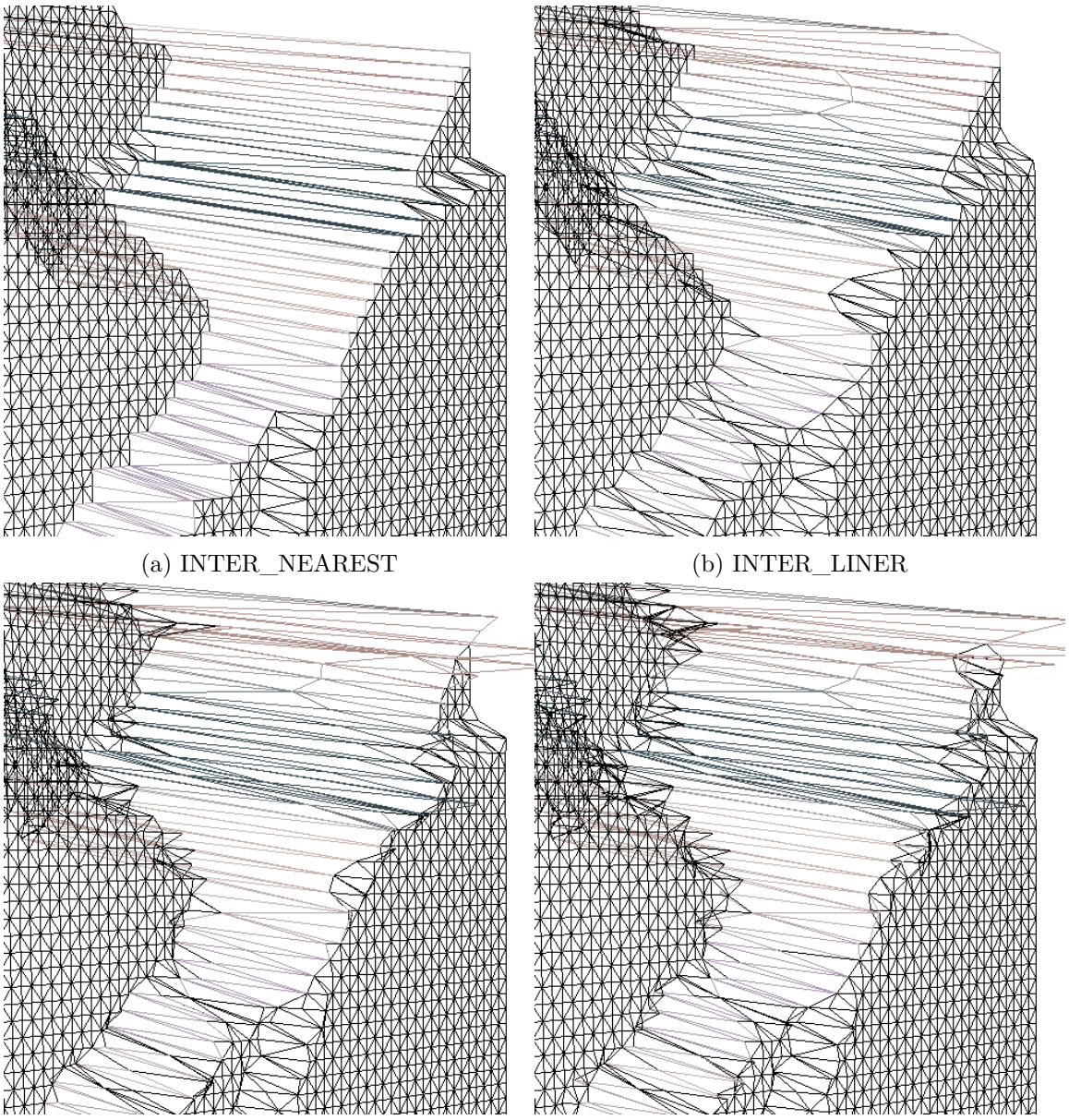
Vstupní fotografie je načtena programem jako typický třísložkový obrázek. Pokud se jedná o obrázek jednosložkový, dochází k rozkopírování složek obrazu tak, aby programem dále zpracovávaná fotografie měla tři typické složky (RGB). Jelikož vykreslovací řetězec OpenGL pracuje s barevnou hodnotou v rozsahu $\langle 0; 1\rangle$, je zdrojová obrazová matice převedena do formátu float. Teoreticky je tedy možné jako zdrojovou fotografii použít HDR snímek, který bude lineárně namapován na barevnou paletu v rozsahu $\langle 0; 255\rangle$ až při samostatném vykreslování.

U hloubkové mapy implementace očekává šedotónový obraz. U formátů nativně podporovaných knihovnou OpenCV dochází k načtení hloubkové mapy přímo vestavěnou funkcí. Vzhledem ke skutečnosti, že synteticky generovaná hloubková mapa ze softwaru LOCATE je uložená ve formátu `.pfm`, byla doimplementována metoda na načítání tohoto formátu. Pokud je obrazová informace v tomto případě tříkanálová, dochází k převedení do jednokanálové.

Načtená hloubková mapa nutně nemusí velikostně odpovídat vstupní fotografii. To může být typicky zapříčiněno podvzorkováním při zpracování obrazu. Proto je nutné změnit velikost hloubkové mapy tak, aby byla stejně velká jako vstupní fotografie. Teprve až poté může dojít k vytvoření prostorového modelu. Použitá konvence při vytváření modelu je, že tmavé odstíny značí kratší vzdálenost od kamery a světlé je pozadí. Tento model byl převzat primárně díky použití zdrojových dat ze softwaru LOCATE. Inverze hloubkové mapy se provede po načtení ostatních vstupních formátů.

Prvním krokem převzorkování hloubkové mapy je extrakce pozadí a detekce chyb vstupních dat. Z původní hloubkové mapy je vytvořena bitová maska pozadí. Ta je vytvořena na základě maximální hodnoty u standardních vstupů a u zdrojových dat ze softwaru LOCATE na základě hodnoty -1[1]. V tomto kroku také dochází ke korekci hodnot. V ojedinělých přípa-

[1] značící pozadí



(a) INTER_NEAREST

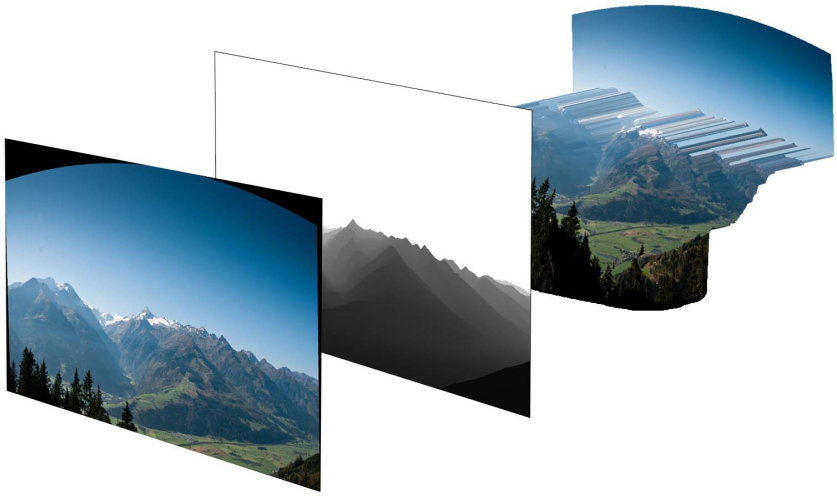
(b) INTER_LINER

(c) INTER_CUBIC

(d) INTER_LANCZOS4

Obrázek 5.1: Vliv interpolační metody při změně rozměru hloubkové mapy na výsledný model

dech totiž exportované hloubkové mapy ze softwaru LOCATE obsahují nevalidní hodnoty. V některých pixelech je uložena hodnota `NAN` nebo hodnota -1 v oblastech, kde by se neměly vyskytovat. Typicky se jedná o osamostatněný pixel a tyto body jsou záplatovány pomocí metody bilineární interpolace z okolních hodnot. Záplatovaní je mnohem jednodušeji proveditelné na vstupních datech rovnou oproti komplikacím s kumulováním chyb při zvětšování. U nově záplatovaných dat nejprve dochází ke zvětšení masky pozadí na rozměry vstupní fotografie. Převzorkování probíhá pomocí interpolační metody nejbližšího souseda tak, aby byl zachován ostrý tvar pozadí, ilustrováno na 5.1a. Oproti tomu samotné převzorkování hloubkové mapy je výhodnější provést libovolnou metodou, která zjemní přechody v modelu. Bohužel jak ukázaly různé pokusy, tak použití pokročilejších interpolačních metod může vést k vytvoření špiček na hranách a ostrých přechodů v modelu, jak je zobrazeno na modelech 5.1c a 5.1d. Z tohoto důvodu byla na převzorkování použita metoda bilineární interpolace. Dříve extrahovaná a převzorkovaná maska pozadí v posledním kroku zvětšení hloubkové mapy poslouží ke zkosení přechodu mezi pozadím a modelem tak, aby zde byla zachována ostrá hrana. Posledním krokem zpracování obrazových vstupů je načtení přídavných textur. U těchto textur nedochází k žádnému předzpracování a převzorkování samotných textur probíhá v případě potřeby přímo až při samotném renderovacím procesu pomocí texturovací jednotky na grafické kartě.



Obrázek 5.2: Ilustrace vytváření ortogonálního modelu

Nyní, když fotografie a hloubková mapa mají stejné rozměry, může dojít k vytvoření samotného prostorového modelu. Prostorový model lze vytvořit dvěma zcela odlišnými přístupy. Pokud známe k původní fotografii úhel zorného pole, je možné z každého pixelu fotografie sestavit konkrétní bod v prostoru. Získání úhlu zorného pole je podrobněji popsáno v předcházející kapitole 2.2.1. Samotný výpočet vychází ze znalosti velikosti čipu fotoaparátu a ohniskové vzdálenosti objektivu. Protože ne u všech fotografií jsou tyto informace k dispozici a protože také výpočet zanedbává další faktory[2], nebyl tento způsob vytváření modelu využit a ani s ním nebylo více do hloubky experimentováno.

[2]redukční kroužky, crop faktor, distorze objektivu

Druhým a zároveň použitým způsobem vytváření mračna bodů je vytvoření ortogonálního modelu přímo z fotografie. Mračno bodů je vytvořeno v normalizovaném modelovém prostoru $\langle -1; 1 \rangle$ na osách X, Y a v rozsahu $\langle -1; 0 \rangle$ na ose Z značící hloubku, kde hodnota 0 je blíže ke kameře. Princip vytvoření mračna bodů je podrobněji rozepsán pseudokódem v algoritmu č. 4. Při ortogonálním vykreslení takto vytvořeného modelu dochází ke ztrátě paralaxy v obraze. Proto je nutné pro perspektivní virtuální kameru model deformovat, jak je dále popsáno v nadcházející kapitole 5.3, aby došlo k zachovaní proporcí původního vzhledu obrazu.

Algoritmus 4: Vytvoření mračna bodů

Data: výška a šířka jsou rozměry vstupního obrázku, hloubková mapa

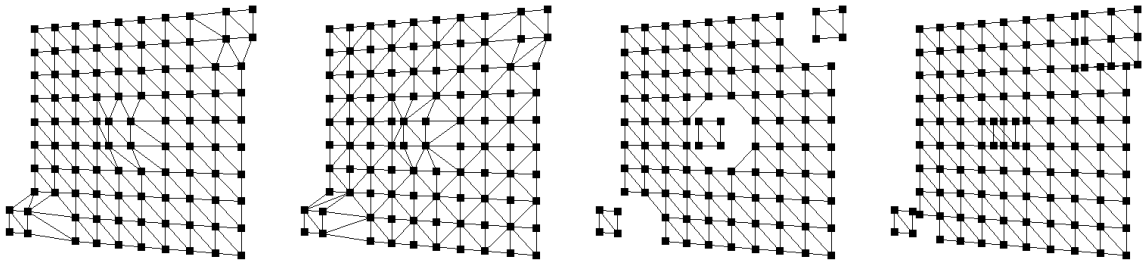
Result: mračno bodů

```
1  for y = výška - 1; y >= 0; y– do
2      yPos = y / (výška - 1);
3      vertex.pos[y] = (yPos - 0.5) * 2.0) * -1.0f;
4      vertex.texCord[y] = yPos;
5      for x = 0; x < šířka; x++ do
6          xPos = x / (šířka - 1);
7          vertex.pos[x] = (xPos - 0.5) * 2.0;
8          vertex.texCord[x] = xPos;
9          vertex.pos[z] = hloubkováMapa[x,y];
10         Uložení vertexu do mračna bodů
```

Takto vytvořený model mračna bodů není problém vykreslit přímo v kolmém pohledu ve větších vzdálenostech. Problém nastává v okamžiku, kdy se kamera přiblíží k modelu nebo se znatelně nakloní. V těchto případech zde začnou být viditelné mezery mezi body. Proto je vhodné propojit body pomoci indexů do trojúhelníků. Trojúhelníky jsou vždy vytvářeny nad čtveřicí bodů. Toto řešení velice elegantně vyplní malé díry v modelu vznikající přiblížením kamery a umožní tak volnější pohyb kamery kolem modelu bez vzniku viditelných artefaktů. Modely typicky bývají spojeny jednoduchou sítí, jak je znázorněno na 5.3a. Mračno bodů není komplikované takto spojit, jelikož je známé pořadí bodů, ve kterém byly přidány. Na základě znalosti sousednosti jednotlivých bodů bylo experimentováno i s dalšími metodami jejich propojení. Jako velice slibný přístup se jevila možnost nevytvářet trojúhelníky v oblastech obrazu, kde je v hloubkové mapě skok mezi body větší než určitý práh. Realizace takto vzniklé sítě indicií je ilustrována na obrázku 5.3c. Hlavní nevýhodou tohoto modelu je existence otvorů (děr) o velikosti jednoho pixelu v kolmém pohledu na model. Tento problém je vyřešen na návrhu metody znázorněné na ilustraci 5.3d. V této metodě v rámci vytváření indexů modelu dochází ke zdvojení bodů. Body jsou zdvojeny v místech překročení prahu rozdílu hloubky, neboli jsou do mračna bodů přidány další body, které mají identické vlastnosti jako body původní[3] a liší se jen jejich Z souřadnice. Při du-

[3] Barvu, texturovací koordináty a souřadnice X a Y

plikaci vždy dochází k přidání od pozorovatele vzdálenějšího bodu. Ideální implementací by bylo, kdyby hloubka nově vzniklého bodu byla dopočítána pomocí roviny definované zbylými třemi body ve čtverci. Experimenty v rámci testování tohoto modelu byly provedeny na modelu, kde duplikovaný bod měl Z souřadnici rovnu vzdálenějšímu mediánu ve čtveřici původních bodů. Přidáním těchto dodatečných trojúhelníků dojde k zaplnění děr v modelu, které byly viditelné při kolmém pohledu na model u metody znázorněné na obrázku 5.3c. Ve finální implementaci se využívá indexů vytvářejících trojúhelníkovou síť znázorněnou na obrázku 5.3b, protože jak je popsáno v následující kapitole č. 5.4, je výhodnější model vykreslovat celistvý včetně artefaktů na pozici děr. Takto vytvořený model neobsahuje žádné díry a nevyžaduje přidání dalších bodů do půdního mračna bodů. Hlavní výhodou tohoto modelu oproti modelu 5.3a je, že díky křížovému vzoru dochází k rozbití pravidelnosti a zjemnění modelu. Toto je možné hlavně proto, že se model skládá z velkého množství malých trojúhelníků.

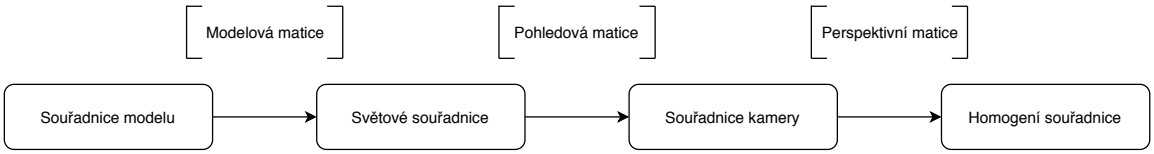


(a) (b) (c) (d)

Obrázek 5.3: Rozdílné způsoby vytváření indexů v modelu

5.3 Perspektiva

Jak již bylo dříve zmíněno v kapitole 5.2, při vykreslení normalizovaného modelu ortogonální projekcí dochází ke ztrátě paralaxního efektu, tudíž při takovémto vykreslení nevynikne hloubka vytvořeného modelu. Proto je v implementaci model vykreslován pomocí grafické techniky nazývané Model-View-Projection (MVP). Principiálně se jedná o sérii transformačních matic, kterými jsou vynásobeny všechny vrcholy modelu tak, aby došlo k perspektivnímu vykreslení modelu z pohledu virtuální kamery [11]. Důležité je mít na paměti, že na pořadí násobení transformačních matic s jednotlivými vrcholy modelu záleží.



Obrázek 5.4: Ilustrace významů jednotlivých transformačních matic MVP

Modelová matice

Modelová transformační matice slouží ke změně tvaru vykreslovaného modelu. K těmto změnám lze využít libovolnou kombinaci transformačních matic vyjádřujících nejrůznější ortogonální operace [12]. Nejčastěji se používají matice rotační, translační a matice měnící

velikost. Kombinace zmíněných matic vzniká násobením, přičemž pořadí násobení matic odpovídá pořadí provedených operací. Vynásobením všech vrcholů modelu výslednou modelovou maticí je model umístěn na svoji pozici ve virtuální scéně. Tato transformace bývá označována jako přechod z modelového souřadnicového systému do systému světového. Jednodušeji řečeno, modelová matice je užívaná k umístění modelu do virtuálního světa.

Ve finální implementaci dochází k vykreslení pouze jediného modelu, a proto je hodnota modelové matice konstantní odpovídající jednotkové matici. To znamená, že model je umístěn do počátku světového souřadného systému.

Pohledová matice

Jedním z možných způsobů vytvoření transformační pohledové matice je její utvoření z tří vektorů určujících pozici a otočení kamery v prostoru. Tyto vektory určují pozici kamery, bod zájmu kamery a vektor směřující vzhůru od kamery [19]. Z těchto tří prostorových informací je možné sestrojit pohledovou matici podle vzorce (5.1).

$$Pohled = \begin{bmatrix} kolmy_x & vzhuru'_x & smer_x & -cil_x \\ kolmy_y & vzhuru'_y & smer_y & -cil_y \\ kolmy_z & vzhuru'_z & smer_z & -cil_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.1}$$

$$smer = \frac{cil - kamera}{\|cil - kamera\|} \tag{5.2}$$

Kde:

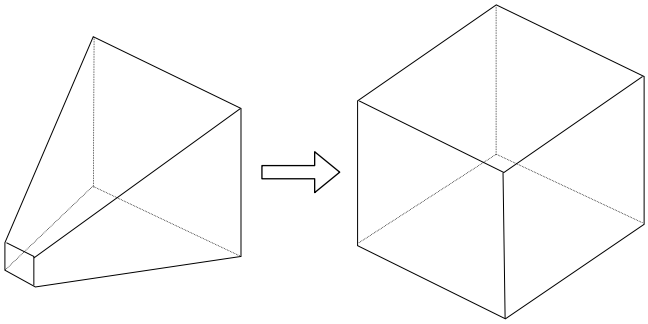
- $kamera$ je třírozměrným vektorem určujícím pozici kamery,
- cil je třírozměrným vektorem určujícím bod zájmu kamery,
- $vzhuru$ je třírozměrným vektorem určujícím směr kolmo vzhůru od kamery,
- Výpočtem (5.2) je určen z $kamera$ a cil normalizovaný směr pohledu kamery,
- $kolmy$ je kolmým vektorem na $smer$ a $vzhuru$ vyjádřeným vztahem: $kolmy = smer \times vzhuru$,
- $vzhuru'$ je vektorem určujícím směr vzhůru vůči směru pohledu získaným ze vztahu: $vzhuru' = kolmy \times smer$.

Násobením všech bodů scény touto transformační maticí se přesouvají tyto body ze souřadnicového systému světového do souřadnicového systému kamerového, jehož počátek je přesně ve středu zorného pole kamery. Na výpočet této matice byla v implementaci využita knihovna GLM, která je kompatibilní s vektorovou konvencí standardu OpenGL. V implementaci je vektor směřující vzhůru od kamery konstantní. To znamená, že kamerou není možné rotovat v ose Z.

Perspektivní matice

Projekční matice slouží k finální transformaci všech vrcholů po transformaci pohledové a modelové. Primární funkce tohoto posledního transformačního kroku je vymezení zobrazované oblasti. Vymezuje maximální a minimální vzdálenosti vykreslovaných objektů.

V anglické literatuře se tento vymezený prostor nazývá clip-space [7]. Standardně se využívají dva typy perspektivních transformačních matic. Matice kompozičně jednodušší pouze ořezává vykreslovaný prostor podle vstupních limitů a zachovává propozice vykreslovaných objektů po předchozích transformacích. Tato matice se nazývá ortogonální neboli také pravoúhlá. Nejčastěji se využívá v programech na vytváření výkresů.



Obrázek 5.5: Ilustrace principu perspektivní matice

Druhou variantou je matice vymezující clip-space v podobě komolého čtyrbokého jehlanu. Ten je vůči kameře orientovaný v prostoru tak, že jeho základna je kolmá na pohledový vektor kamery a jednotlivé hrany jehlanu se rozcházejí směrem od kamery tak, jak je zobrazeno na obrázku č. 5.5. Sestavit transformační matici, která dokáže převést prostor komolého čtyrbokého jehlanu na krychli se standardně dělá podle nadcházejícího vzorce (5.3).

$$Perspektiva = \begin{bmatrix} \frac{1}{aspekt*\tan(\frac{fov}{2})} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(\frac{fov}{2})} & 0 & 0 \\ 0 & 0 & -\frac{z_{vz}+z_{bl}}{z_{vz}-z_{bl}} & -\frac{2*z_{vz}*z_{bl}}{z_{vz}-z_{bl}} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{5.3}$$

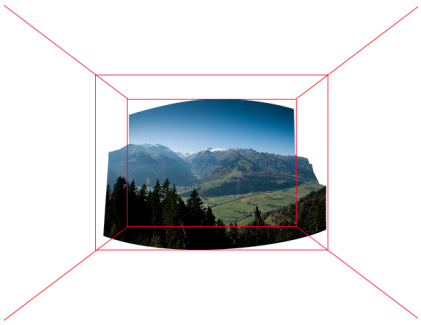
Kde:

- *aspekt* je poměrem šířky ku výšce okna, do kterého se vykresluje,
- fov je úhlem zorného pole kamery,
- z_{bl} určuje vzdálenost bližší ořezové plochy,
- z_{vz} určuje vzdálenost vzdálenější ořezové plochy.

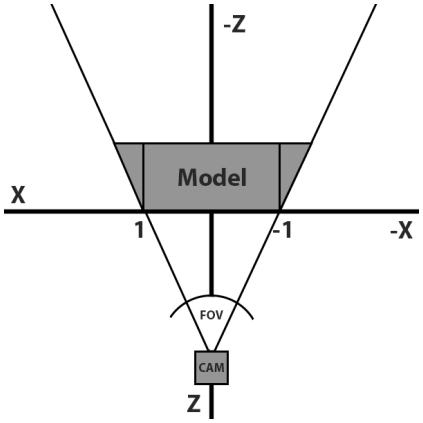
Alternativním přístupem je určení výřezového jehlanu pomocí jeho vrcholů. Jak je na ilustraci č. 5.5 vidět, šíře zorného pole se se vzdáleností od kamery postupně rozšiřuje. Tento jev ve virtuálním ortogonálním prostoru simuluje chování skutečné kamery. Tato skutečnost má za následek vznik paralax efektu a jevu sbíhavosti svislic ve vykreslované scéně. V implementaci se k vytváření perspektivní matice stejně jako u matice pohledové využívá knihovny GLM.

5.3.1 Rekonstrukce původního obrazu

Při aplikaci MVP transformací dochází k vytvoření efektu sbíhavosti obrazu a ten podtrhuje prostorový vjem při vykreslení modelu. Nevýhodou takto vykresleného modelu je to, že při přímém pohledu na model zepředu dochází ke zkreslení původní fotografie. Tento jev je viditelný na obrázku 5.6. Pro rekonstrukci původního obrazu je nutné v modelovém prostoru provést operaci inverzní vůči perspektivnímu zkreslení. Principiálně je nutné model deformovat rozbíhavostí v závislosti na vzdálenosti od kamery. Rovnice (5.4) vyjadřuje změnu polohy jednotlivých bodů v závislosti na hloubce z a úhlu zorného pole fov. Je-li úhel rozbíhavosti modelu a sbíhavosti perspektivy stejný, tak při přímém kolmém pohledu na model zepředu je možné vypočíst pomocí vztahu (5.5) pozici kamery, kde projekce modelu bude vykreslena v identické podobě jako původní vstupní fotografie.



Obrázek 5.6: Zkreslení ortogonálního modelu perspektivní projekcí

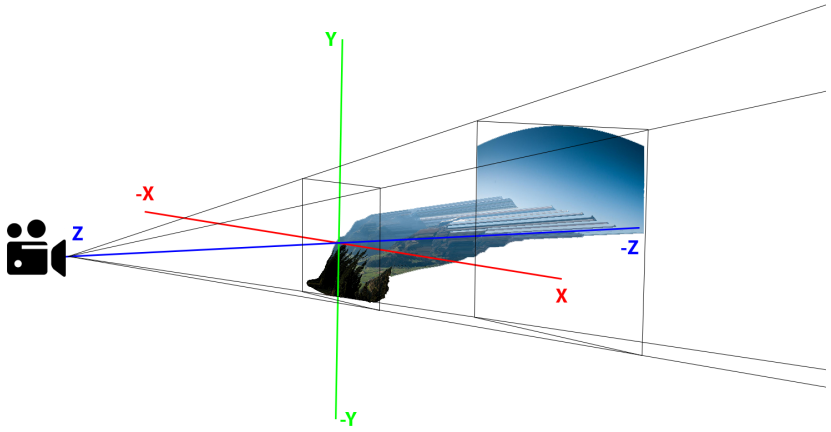


Obrázek 5.7: Nákres k rovnicím (5.4) a (5.5)

$$\begin{aligned} x' &= x \cdot \tan(fov/2) \cdot |z\ | \\ y' &= y \cdot \tan(fov/2) \cdot |z\ | \end{aligned} \tag{5.4}$$

$$cam_z = \frac{1.0}{\tan(fov/2)} \tag{5.5}$$

Proces deformace modelu rozbíhavosti je sice možné provést přímo při vytváření modelu, ale přístup transformace v modelovém prostoru v průběhu umožňuje dynamicky měnit nastavení scény. Toho může být využito při dynamické prezentaci fotografie, jelikož je možné kromě pohybu kamery měnit její perspektivní nastavení za běhu aplikace. V rámci modifikací souřadnic bodů modelu je také možné měnit parametricky vykreslovanou hloubku modelu. Na Z souřadnici bodů je například možné aplikovat gamma korekci, která může potlačit či zvýraznit určitý rozsah hodnot hloubek v modelu podobně jako u fotografií. Další možnou transformací hloubky je vynásobení konstantou, která zvětší její rozsah, a tím dojde k umocnění prostorového efektu při pohybu kamery.



Obrázek 5.8: Výchozí nastavení scény

Výchozí nastavení scény při spuštění programu je znázorněno na obrázku 5.8. Úhel záběru virtuální kamery je shodný s úhlem rozbíhavosti modelu tak, aby zobrazení modelu odpovídalo vzhledově vstupní fotografii. Na obrázku je černou barvou vyznačeno zorné pole kamery (dříve zmíněný clip-space) a barevně vyznačený střed souřadného systému.

5.4 Detekce děr v modelu

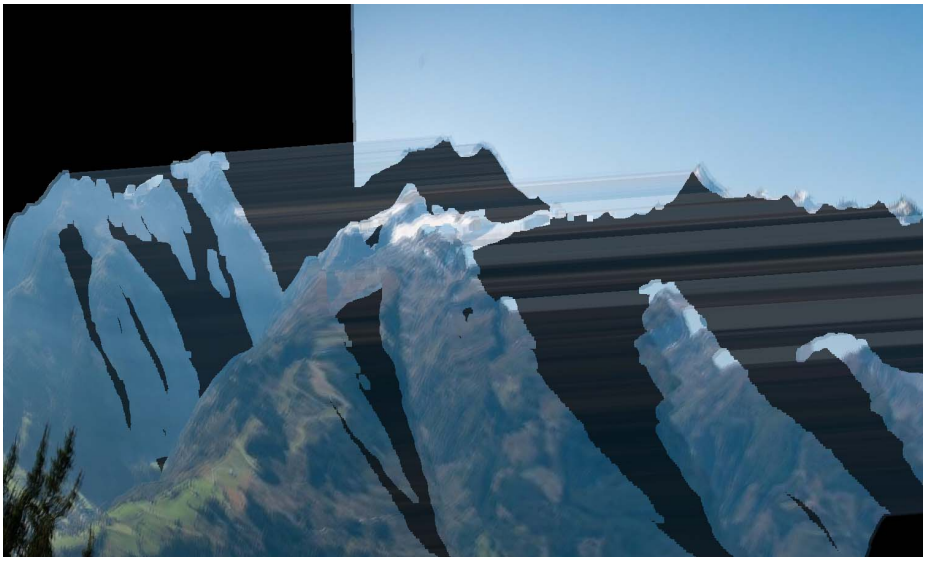
Model skládající se z mračna bodů vytvořeného z jedné zdrojové fotografie a její hloubkové mapy lze kvalitně vykreslit při kolmém pohledu zepředu na model. Prostorový vjem z obrazu lze pak docílit pohybem virtuální kamery v ose Z. Problém ovšem nastává v okamžiku, kdy se kamera vychýlí při svém pohybu z osy Z. Mračno bodů neobsahuje body schované za body z fotografie. Zde jsou v modelu takzvané díry. Jak již bylo zmíněno v kapitole 5.2, malé díry v modelu jsou zahlazeny a schovány trojúhelníkovou sítí. Trojúhelníková sít 5.3b zvolená pro vykreslování modelu obsahuje propojení všech bodů v mračnu. To znamená, že v modelu jsou přítomny i trojúhelníky na hranách horizontů v hloubkové mapě. To má za následek to, že při renderování modelu z většiny úhlů pohledů dojde k vykreslení děr, kde chybějí body v modelu. Cílem této kapitoly je navrhnout vhodný způsob detekce fragmentů na pozicích děr v obraze tak, aby mohly být zaplněny.

5.4.1 Detekce děr pomocí vykreslování mračna bodů

První metoda, se kterou bylo experimentováno, využívá samotných bodů mračna. Idea této metody spočívá ve využití stencil bufferu vykreslovacího řetězce. Tato metoda detekce vykreslí samotné mračno bodů bez indexů a v místech, kde není nic vykresleno, je detekována díra v obraze. Pro zvětšení efektivity bylo experimentováno s rozdílnými velikostmi vykreslovaných bodů do stencil bufferu tak, aby došlo skutečně k detekci větších jednolitých děr v obraze. Fixní velikost vykreslovaných bodů při použití této metody není příliš ideální. Vhodnější by bylo dynamicky měnit velikost v závislosti na vzdálenost kamery od modelu. Implementace takovéto funkcionality není nikterak komplikovaná, ale k její realizaci nikdy

nedošlo, jelikož tato metoda má jiné a mnohem závažnější nedostatky. U děr detekovaných tímto způsobem bylo často vhodné výstup detektoru ještě poupravit pomocí morfologických operací tak, aby došlo ke spojení děr do větších shluků.

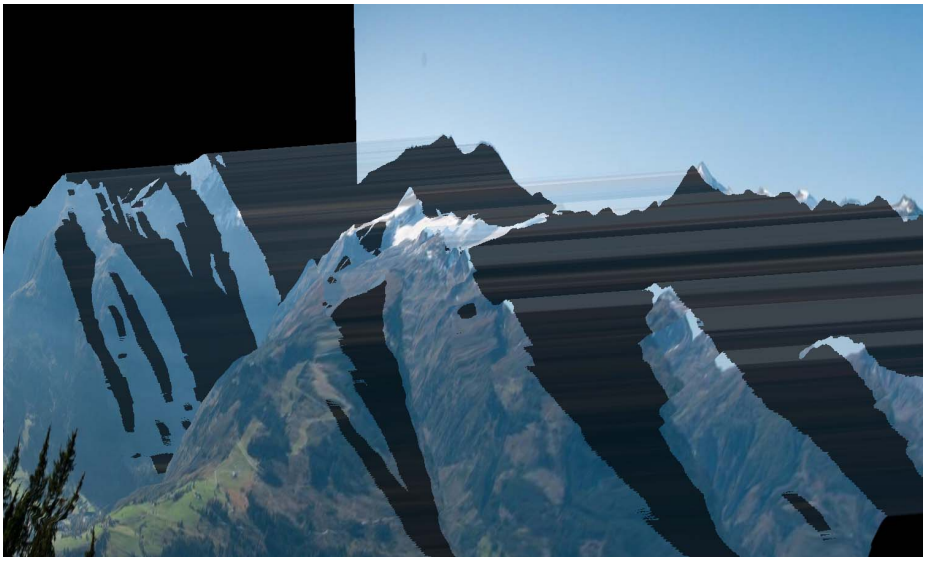
Asi nejzásadnější problémem této metody je vznik odsazení vykreslených bodů vůči modelu vykresleného pomocí trojúhelníkové sítě. Toto odsazení vzniká při zvětšování vykreslovaných bodů. Rozdíly v pozicích jednotlivých bodů jsou zobrazeny na obrázku 5.9 níže. Rozdíl je roven polovině velikosti jednotlivých bodů. Protože tedy tato metoda v okrajových podmínkách detekuje díry v obraze na místech, kde nejsou artefakty a naopak, nebylo shledáno použití této metody vhodným. Navíc tato metoda vyžaduje pro detekci děr další samostatný renderovací cyklus, což není efektivní využití výpočetních zdrojů grafické karty.



Obrázek 5.9: Překrytí modelu vykreslenými zvětšenými body

5.4.2 Detekce děr pomocí vykreslování děravého modelu

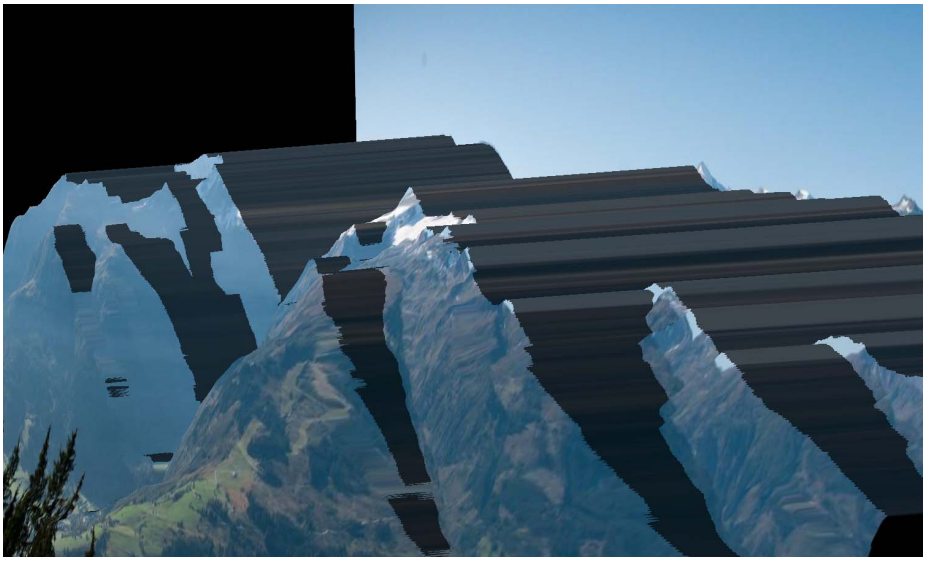
Druhou navrženou metodou k detekci děr v modelu je využití vykreslení děravého modelu 5.3d. Výhodou tohoto přístupu oproti přístupu předchozímu je to, že je možné detekovat díry v obraze v rámci jednoho vykreslovacího cyklu, kde dochází k vykreslení modelu a zároveň vykreslení děr. Díry jsou v tomto případě získány inverzí stencil bufferu. Takto získané díry se shodují na pixel přesně s renderovaným obrazem. K chybné detekci díry může dojít v okamžiku, kdy ve směru od kamery je za dírou vykreslená další část modelu. Takto vykreslené části lze vidět na obrázku 5.10. Při použití vykreslovací techniky, která nevykresluje trojúhelníky čelem odvrácené od kamery, se tato chyba značně zmenšuje. U některých modelů by tato metoda detekce mohla být zcela dostatečná. Pokud je ovšem model velmi členitý, je velmi pravděpodobné, že dojde k zaplnění děr vzdálenějšími částmi modelu. To v praxi znamená to, že skrze díru v modelu je vidět další část modelu. Tato situace může nastat například u členitějších údolí v jednom snímku.



Obrázek 5.10: Překrytí modelu modelem s dírami

5.4.3 Detekce děr pomocí vykreslovacího řetězce

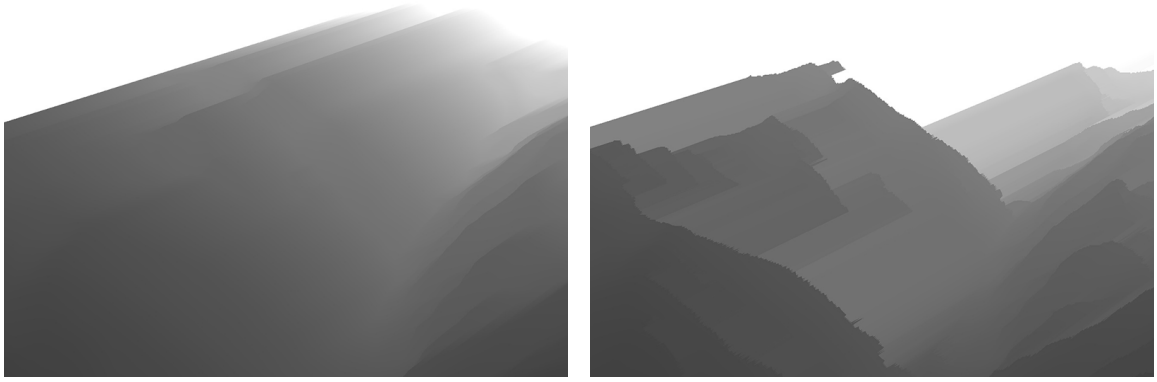
V konečné implementaci použitou metodou k detekci děr v modelu je metoda zvýrazňující trojúhelníky, které mají hranu větší než určený práh. Tyto trojúhelníky jsou vyznačeny až při samotném procesu vykreslování modelu v geometry shaderu. Tento geometry shader pouze zpracovává trojúhelníky na vstupu a mění některé jejich parametry na výstupu. Nedochází zde k žádné modifikaci počtu bodů či trojúhelníků ve vykreslovaném modelu. Tento přístup detekce děr v modelu řeší všechny dříve popsané problémy a zároveň dynamicky reaguje na parametry měnící vzhled modelu. To znamená, že dochází skutečně k detekci děr v částech modelu, kde se vyskytují fragmenty vzniklé vykreslením roztažených trojúhelníků.



Obrázek 5.11: Zvýraznění detekovaných děr na vykresleném modelu

Výhodou tohoto řešení je již zmíněná dynamičnost detekce, jelikož dokáže detekovat díry nově vzniklé změnou parametrů. Příkladem může být změna gamma korekce hloubky. Toto řešení je také využitelné u dalších deformací modelu. Příkladem může být simulování objektivu rybího oka. U takto extrémní deformace může dojít například k detekci díry na okraji modelu, kde bylo výhodné doplnit více bodů tak, aby byl výsledek detailnější. Je-li tento efekt nechtěný, stačí správně nastavit velikost prahu detekce.

V rámci zpracování detekovaných trojúhelníků v geometry shaderu také dále dochází k modifikaci vlastnosti `depth`. U trojúhelníků, které jsou označené jako díry, dochází k modifikaci složky `depth` u jednotlivých vrcholů za účelem vytvoření nové hloubkové mapy `TARGET_DEPTH_MAP`, jak je ilustrováno na obrázku č. 5.12b. Účelem této hloubkové mapy je podpoření syntézy textury tím, že simuluje hloubku obrazu za hranou z daného pohledu. Toho je docíleno tím, že parametr `depth` je u všech vrcholů detekovaných trojúhelníků roven hloubce nejvzdálenějšího bodu od počátku v ose Z.



(a) Původní hloubka (b) Hloubka po transformaci geometry shaderu

Obrázek 5.12: Rozdílné vstupní hloubky pro syntézu

Původní hloubka nanesená na model vizualizovaná na obrázku č. 5.12a je u roztažených trojúhelníků mezi jednotlivými vrcholy interpolována v rámci zpracování trojúhelníků ve fragment shaderu, jelikož je zpracována jako barva. Takto zobrazená hloubka může značně omezit možnosti výběru záplat syntetizačního API tak, jak je vidět na obrázku č. 5.13a, kde za horizontem došlo k několikanásobnému nakumulovaní stejné záplaty na celé délce díry. Oproti tomu na obrázku č. 5.13b je patrné, že syntetizační API mělo mnohem větší volnost při použití záplat utvářejících horu v pozadí.

(a) Výstup při použití vodítka 5.12a (b) Výstup při použití vodítka 5.12b

Obrázek 5.13: Rozdílné výstupy syntézy, kde hloubková mapa byla jediným použitým vodítkem

5.5 Vytváření záplat modelu

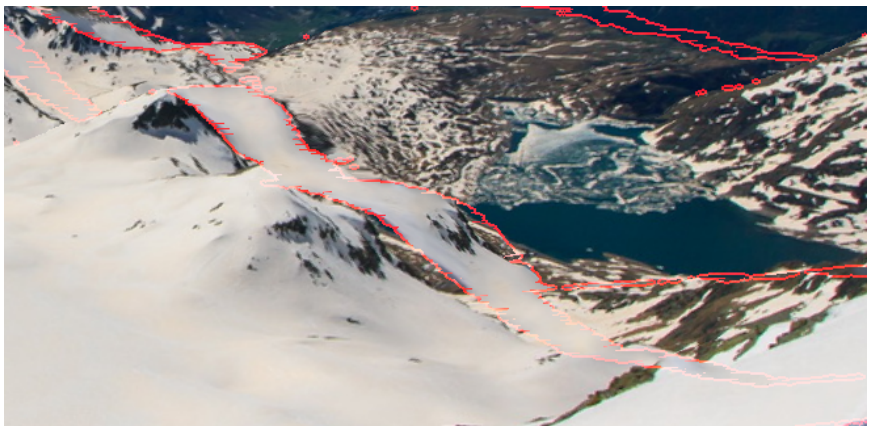
Z návrhu funkčnosti programu 2 vyplývá, že syntéza textur je v implementaci využita k vytváření dalších bodů prostorového modelu. Problémem je, že syntetizační API pracuje pouze s obrazovými informacemi. To znamená, že dokáže vyplnit pouze detekované díry v reprodukovaném obraze. Tato kapitola se proto zabývá problematikou, jak ze syntézou vytvořené textury vytvořit záplaty modelu, které překryjí (vyplní) díry v modelu. Tento úkon má dvě hlavní úskalí. Prvním je jakým způsobem určit pozici syntetizovaných bodů v původním modelu při libovolné konfiguraci scény. Druhým problémem je jakým způsobem tyto body spojit do trojúhelníků, aby mohly být následně vykresleny pomocí stejného grafického řetězce jako model.

5.5.1 Výběr záplaty

Prvním úkonem, který je třeba provést při vytváření modelové záplaty, je výběr bodů ze syntetizované textury, které budou použity při jejím vytvoření. Jako maska bodů poslouží výstup detektoru děr odpovídající syntetizované textuře. Ve finální implementaci existují dva způsoby, jak z výstupu detektoru děr vytvořit masky záplaty. První možnost je manuálně řízená uživatelem a druhá automatizovaná.

Automatizovaná metoda využívá funkce `findContours` z knihovny OpenCV k nalezení jednotlivých ucelených množin bodů. Každý takto detekovaný celistvý shluk je následně zpracován jako maska ohraničující jednu záplatu. Toto volání probíhá cyklicky pro každou díru velikostně překračující určitý práh, aby došlo k vyloučení záplat obsahujících příliš malé množství bodů. Příliš malé záplaty by zbytečně zpomalovaly vykreslovací řetězec za cenu minimálního dopadu na vylepšení vizuálního vzhledu modelu.

Metoda ruční může být použita uživatelem, který chce ze syntézou vytvořené textury aplikovat pouze jednu záplatu na model. Užití tohoto přístupu se předpokládá v případě, kdy uživatel specificky nastavil parametry syntézy pro potřeby překrytí určité konkrétní díry v modelu. Při ručním výběru je uživateli zobrazeno dedikované okno, které obsahuje obraz reprodukce obrazu, kde jsou díry vyplněny částmi ze syntetické textury. Tohoto zobrazení je docíleno pomocí vymaskovaného sčítání obrazu, kde maskou je výstup detektoru děr, přičemž okraje záplat jsou obražené červenou barvou, viz obrázek č. 5.14. Vytvoření finální masky se spouští po stisku ukazatele myší nad záplatou. Maska vzniká rozdílem dvou téměř totožných masek, kde rozdílem je, že na jedné byl proveden algoritmus semínkového vyplňování spuštěný od souřadnic poklepu.



Obrázek 5.14: Zvýraznění syntetických záplat

5.5.2 Inverzní transformace

Jelikož vykreslovací proces je v podstatě matematické zobrazení třídimenzionálního prostoru do dvoudimenzionálního, je teoreticky možné provést sérii inverzních operací tak, abychom v modelovém prostoru získali pro každý pixel z obrazu vektor počínající z pozice kamery. Výpočtem průsečíků těchto vektorů je tak možné získat pozice původně zobrazených bodů modelu. Rozvinutím této myšlenky vzniká níže vypsaný algoritmus 5.

Algoritmus 5: Výpočet souřadnice syntetizovaného pixelu v modelu

Data: souřadnice bodu v reprodukci

Result: souřadnice bodu v modelu

```
1  Vytvořit třírozměrný vektor v homogenním krychlovém prostoru ze souřadnic bodu
2  Vynásobit vektor inverzní perspektivní transformační maticí
3  Vynásobit vektor inverzní pohledovou transformační maticí
4  Vynásobit vektor inverzní modelovou transformační maticí
5  Vynásobit vektor inverzní perspektivní transformační maticí
6  Provést inverzi operací prováděných ve vertex shaderu nad vektorem (5.4)
7  foreach trojúhelník modelu do
8      Vypočíst průsečík vektoru s trojúhelníkem
9      if nalezen průsečík then
10         return souřadnice průsečíku
11 return průsečík nenalezen
```

Použití tohoto algoritmu však není zcela optimální, a to z následujícího důvodu. Primárním problémem je efektivita vyhledávání průsečíku, jelikož model se skládá z nemalého množství trojúhelníků. Přesněji se v modelu nachází zhruba tolik trojúhelníků, kolik má vstupní obrázek pixelů. To znamená, že přístup vyhledávání průsečíku v naivním cyklu je značně neefektivní. Způsob vyhledávání by se jistě dal optimalizovat pomocí technik využívaných při vykreslování. Jednou takovouto použitelnou technikou je například uspořádání trojúhelníků do hierarchické stromové struktury. Toto uspořádání při algoritmickém prohledávání značně zredukuje počet výpočtů průsečíků, než dojde k nalezení toho hledaného. Další možností, jak tento algoritmus urychlit, je že vyhledávání průsečíku bude probíhat systematicky v okolí posledního nalezeného, jelikož se předpokládá, že postupně vyhledávané body jsou vůči sobě prostorově blízké.

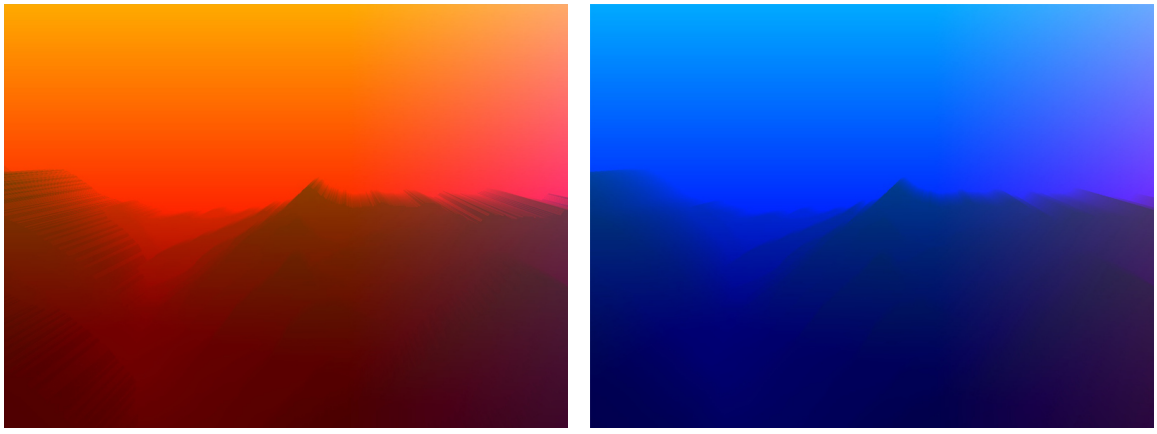
5.5.3 Využití souřadnicové textury

Protože v průběhu vykreslování modelu již dochází k nalezení průsečíků pohledových vektorů z kamery s modelem, nabízí se možnost tyto průsečíky rovnou ukládat. V herním průmyslu se tato technika například užívá k identifikaci objektů ve scéně, kde každý objekt má svůj unikátní identifikátor reprezentovaný barvou. Tyto identifikátory se vykreslují do speciální textury pomocí techniky Multiple Render Targets. Jelikož tato technika je v im-

plementaci již využita pro vytváření vodítek pro syntézu textur, není problém přidat další výstupní texturu do implementovaného frame bufferu.

Jednou z možností, jak tuto techniku adaptovat v implementaci, by bylo přidělit jednoznačné identifikátory jednotlivým bodům mračna. Ale protože se ve vykreslované scéně nachází pouze jeden model, který má normalizované rozměry, nabízí se možnost vykreslovat přímo souřadnice jednotlivých bodů. Dochází zde tedy k vykreslování hodnot v rozsahu $\langle -1; 1 \rangle$ pro všechny tři osy. Každá osa je uložena do rozdílné barevné složky. Cílová textura musí být datového typu float, stejně jako jsou uloženy souřadnice jednotlivých bodů. Původní souřadnice jednotlivých vrcholů jsou v průběhu rasterizace trojúhelníku zpracovány jako barva v rozsahu $\langle -1; 1 \rangle$. Výhodou tohoto řešení je, že dochází k lineární interpolaci u bodů vykreslených uvnitř trojúhelníku.

Důležité je, aby souřadnice vykreslené do reprodukce skutečně přesně na pixel odpovídaly modelu. V prvotní fázi vývoje byla na jednotlivé trojúhelníky nanesena textura obsahující souřadnice odpovídající jednotlivým bodům pomocí texturovacích koordinát s tím, že reprodukce tohoto zobrazení poslouží k akceleraci algoritmu hledajícího průsečíky. Jak je ale vidět na obrázku č. 5.15a, tak textura je na model nanesena nepřesně. Přesněji se jedná o posunutí o půl pixelu. Toto je zapříčiněno procesem vytváření modelu z obrázku. Proto je ve finální implementaci datová struktura jednotlivých bodů 5.1 zachována v průběhu celého vykreslovacího procesu. Informace uložené v jednotlivých bodech jsou použity k přesnějšímu vykreslení barevných informací na modelu. Konkretně se tedy jedná o následující informaci pro každý bod: původní pozici v homogenním modelu, texturovací koordináty, výstup detektoru děr a modifikovanou hloubku. Při pohledu na obrázek č. 5.15b je vidět, že problém projevující se jako varhánkovité pruhy za horizonty při použití informací přímo z původních souřadnic bodů zcela vymizel. Problém, že texturovací koordináty nepřesně sedí na model, není zásadní, jelikož jsou využity pouze při nanášení přidaných textur na model. Tyto textury mohou být výstupy nejrůznějších detektorů a segmentorů. Tyto obrázky mohou být oproti původním podvzorkovány, nebo obsahovat pouze obalové obrysy detekovaných objektů. Existují i metody, které dokáží vytvářet bitové masky detekovaných objektů [1]. Ale i takovéto výstupy mají jistou vůli a benevolenci a nemusejí přesně na pixel sedět s původním obrazem. Tudíž posun textury o půl pixelu není zase tak znatelný a neovlivní tak kvalitu syntézy.

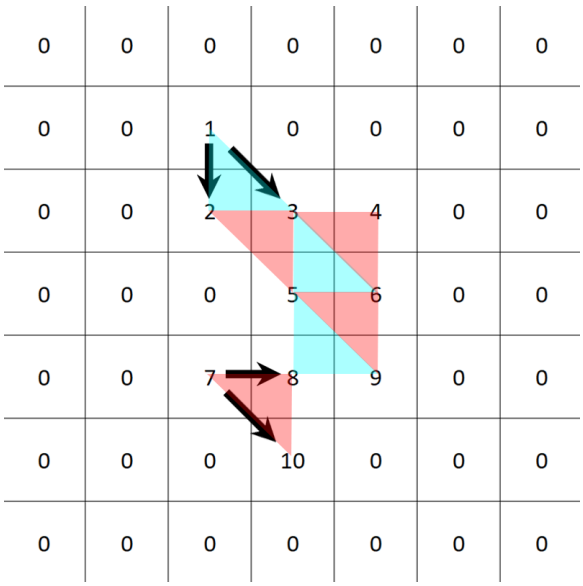


(a) Souřadnicová textura (b) Využití souřadnic původních bodů

Obrázek 5.15: Rozdílné přístupy k mapování souřadnic v modelu

5.5.4 Kompletace záplaty

Po vyřešení problémů, ze kterých bodů syntetické textury vytvořit nové body modelu a jaké mají souřadnice, zbývá poslední krok k dokončení vytvoření prostorové záplaty, aby mohlo dojít k jejímu vykreslení pomocí stejného vykreslovacího řetězce. Tímto úkonem je propojení bodů do trojúhelníků. Princip vytvoření indexů trojúhelníků z bodů záplaty podobně jako u vytvoření modelu vychází z předpokladu souslednosti jednotlivých bodů. Jinými slovy se předpokládá, že dva body syntetické záplaty jsou umístěny v prostoru vedle sebe. Problémem tentokrát je, že záplata není čtvercem ani obdélníkem, ale obecným segmentem obrazu určeným maskou. Proto při vytváření jednotlivých bodů záplaty dochází k plnění takzvané indexové mapy. Tou se rozumí nová textura velikostně odpovídající velikosti textury syntetické, přičemž při vytvoření nového bodu záplaty dojde k uložení jeho pořadového čísla do indexovací textury. Platí tedy, že tato textura obsahuje hodnoty pouze na pozicích korespondujících s maskou záplaty. V ostatních bodech obsahuje hodnotu nula. Vizualizace výřezu takovéto indexové mapy je k vidění na obrázku č. 5.16.



Obrázek 5.16: Propojení bodů záplaty do trojúhelníku

Na tom samém obrázku č. 5.16 je také ilustrovaný algoritmus vytváření indexů trojúhelníků pomocí šipek. Algoritmus je navržený tak, že postupně prostupuje indexovou mapu z levého horního rohu po řádcích směrem doprava dolů. V okamžiku, kdy algoritmus při průchodu narazí na nenulovou hodnotu, dochází ke kontrole sousledných bodů podle zobrazených šipek. Kontrolují se vždy dvojice sousledných bodů, které lze spojit do trojúhelníku podle vzoru červených nebo modrých trojúhelníků. Pokud oba sousledné body obsahují také nenulové indexy, dochází k vytvoření trojúhelníku záplaty. Tímto způsobem je možné záplatu kompletovat. Tato metoda má i své problémy. Jak je vidět na obrázku č. 5.17a, tak při pohledu na tento konkrétní model z boku se dvě detekované díry v porostou překrývají, protože ke kameře bližší horizont je nižší, respektive došlo při závěrečné rasterizaci výsledku detektoru děr k jejich spojení.

Z algoritmu vytvářejícího indexy trojúhelníků vyplývá, že v tomto případě dojde k vytvoření záplaty, která přemostí údolí mezi horizonty. Takovéto přemostění je ilustrováno na obrázku č. 5.17b. Aby došlo k zabránění vytváření takovýchto mostů, byla do algoritmu spojujícího body do trojúhelníků zakomponována podmínka detekující roztažené trojúhelníky. Principiálně se jedná o stejnou podmínku, kterou využívá detektor děr v modelu. Tentokrát však dochází k detekování prostorově roztažených trojúhelníků přímo při jejich vytváření. Takto detekované trojúhelníky nakonec nejsou vytvořeny a dojde k rozdělení záplaty. Práh detekce roztažených trojúhelníků se dynamicky přizpůsobuje konkrétnímu modelu. Konkrétně hodnota prahu je odvozena z velikosti původních trojúhelníků propojujících původní model.

(a) Pohled na dvě překrývající se díry

(b) Pohled na výslednou záplatu

Obrázek 5.17: Záplata přemosťující údolí

Kapitola 6

Dosažené výsledky

Výsledná implementace byla vyvíjena tak, aby byla využitelná na reálných datech a dokázala prezentovat potenciál užití hloubkové mapy při prezentaci fotografií. Primárním zdrojem vstupních dat byl data set[1] generovaný nástrojem LOCATE zmíněným v kapitole 2.3.1. Proto jsou některé kroky v procesu vytváření modelu specificky přizpůsobeny těmto vstupním datům, primárně v části načítání a zpracování obrazových informací. To ovšem neznamená, že finální implementace nedokáže prostorový model sloužící k vytvoření video sekvence utvořit z libovolných dat. Je nutné podotknout, že dále uvedené fotografie ze zmíněného datasetu byly pečlivě vybrány a nejedná se o náhodný výběr. Účelem každé fotografie je demonstrovat rozdílné vlastnosti implementace. Všechna zmíněná testovací data jsou přiložena na digitálním médiu, jež je přiloženo k této praci. Při výběru jednotlivých ukázek byl kladen důraz na přesnost hloubkové mapy na vizuálně dominantních horizontech tak, aby nedocházelo k jejich přenesení do pozadí. Takto nepřesná hloubková mapa může následně negativně ovlivnit výsledky syntézy, nebo může působit opticky rušivým dojmem.

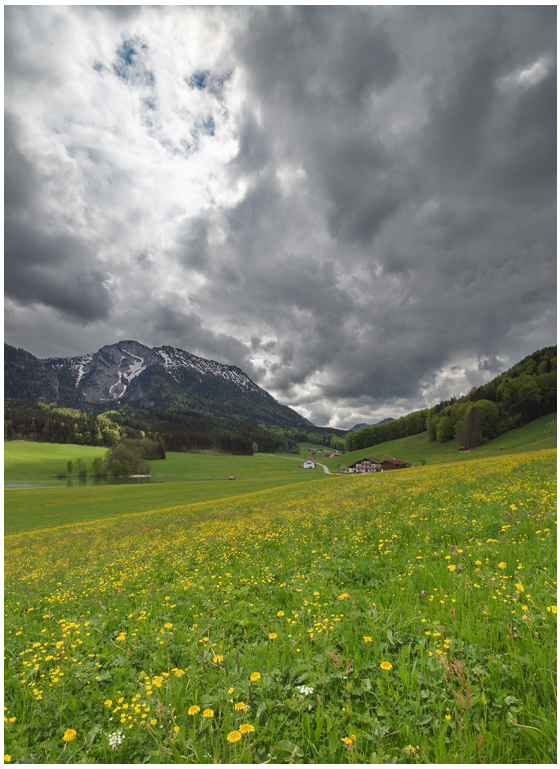
Pro demonstraci funkčnosti programu i na jiných hloubkových datech byly vygenerovány k testovacím fotografiím hloubkové mapy za použití projektu MegaDepth, který byl zmíněn v kapitole 2.3.2. Při generování hloubkových map byla využita referenční natrénovaná síť[2]. Tato předtrénovaná neuronová síť byla primárně učena na datech z prostředí městské zástavby, proto výsledné hloubkové mapy nemusejí být zcela optimální. Dalším omezením takto získané hloubkové mapy je její rozlišení, které je omezeno v demo aplikaci na rozměry 512 x 384 pixelů. I na toto omezení bylo při implementaci přihlíženo, a proto před vytvořením prostorového modelu dochází k zarovnání velikosti vstupní fotografie a hloubkové mapy, jak je popsáno v kapitole 5.2.

V rámci testování došlo i k ručnímu vytvoření hloubkové mapy. Tato mapa poslouží jako simulace uživatele nemajícího možnost vygenerovat si hloubkovou mapu jiným způsobem. Tato možnost byla vzata v potaz, jelikož inspirace k implementaci této práce také částečně vychází z postupů používaných ve video průmyslu, tak, jak je popsáno v kapitole č. 3. Ručně vytvořená hloubková mapa vyobrazená na obrázku č. 6.1d obsahuje pouze čtyři odstíny šedé reprezentující skokové rozdíly v hloubce. Jednotlivé odstíny opisují horizonty tak, jak by je uživatel oddělil pro potřeby vytvoření prostorového vjemu v grafickém editoru.

Na ukázku testovací sady je možné se podívat na následující stránce na obrázku č. 6.1. U zobrazení hloubkové mapy generované softwarem LOCATE 6.1b byla provedena normalizace a inverze barev tak, aby všechny hloubkové mapy používaly stejnou barevnou konvenci reprezentace hloubky. Kompletní přehled použitých fotografií je v příloze D.

[1] Dostupný na: http://cphoto.fit.vutbr.cz/geoPose3K/

[2] Dostupná na:
http://www.cs.cornell.edu/projects/megadepth/dataset/models/best_generalization_net_G.pth



(a) Původní fotografie

(b) LOACATE

(c) MegaDepth

(d) Ručně vytvořená

Obrázek 6.1: Rozdílné zdroje hloubkové mapy

6.1 Vytvořený model

Jak již bylo zmíněno, kvalita modelu koreluje s přesností vstupní hloubkové mapy. Pro vytvoření vizuálně příjemného videa je proto zapotřebí, aby dominantní linie horizontu byla pokud možno co nejlépe zarovnána na hloubkové mapě s fotografií. Pozorovatel snímku si pak nemusí povšimnout nedokonalostí jinde na modelu. Dalším důvodem, proč je důležité mít dobře ohraničený horizont, je fakt, že dobře zarovnaná hloubková mapa podpoří kvalitu výstupu syntézy při generování záplat. Toto je hlavně důležité na pomezí pozadí a horizontu. Na obrázku č. 6.2c je ilustrován dopad špatně zarovnaného horizontu na vzhled modelu. V okamžiku, kdy bude uživatel chtít syntézou překrýt vyobrazenou díru, nastane opravdový problém, jak nastavit parametry syntézy. Primárním zdrojem problému je to, že nepatrná část hory se nachází na pozadí modelu. Tento roh hory se může v rámci běhu syntézy rozdistribuovat do libovolné záplaty, která se bude pokoušet vytvořit obraz pozadí.

(a) Detail horizontu

(b) Překrytí fotografie hloubkovou mapou

(c) Projev posunuté hloubkové mapy na modelu

Obrázek 6.2: Ukázka špatně zarovnané hloubkové mapy na modelu

Nejlepším příkladem pro demonstraci výhod prostorového modelu k vytvoření paralax efektu je určitě fotografie č. D.5, protože je přesně lokalizována v softwaru LOCATE a obsahuje detailní hloubku jednotlivých úrovní horizontů na pozadí. Tato fotografie je ideálním kandidátem na vytvoření video sekvence obsahující přelet fotografie z jedné strany na druhou díky svému velkému rozlišení. Při vytvoření videa s rozlišením 1080p tak nedojde k zásadní ztrátě kvality původní fotografie a bude působit mnohem zajímavěji než přelet kamery nad plochou fotografií. Bohužel takovéto hloubkové rozmanitosti nedosahuje hloubková mapa generovaná neuronovou sítí z projektu MegaDepth. Primárním důvodem, proč tomu tak je, je nízký kontrast u jednotlivých horizontů. Z tohoto důvodu je pro vytvoření stejného efektu nepoužitelná.

Oproti tomu hloubková mapa generovaná stejnou neuronovou sítí k obrázku D.4 má jedinečnou výhodu, kterou v současnosti systém LOCATE nedokáže zreprodukovat, protože využívá pouze geologického modelu, který neobsahuje porost ani budovy. Proto u modelu vytvořeného pomocí hloubkové mapy 6.1b dojde k optickému rozseknutí lesa, který se nachází v pravé části fotografie. K tomuto jevu však nedojde v případě použití hloubkových map 6.1c a 6.1d.

Zajímavostí je fotografie č. D.3 kde, i když jsou na první pohled vstupní hloubkové mapy silně rozdílné, má výsledný model podobné optické kvality. A to z toho důvodu, že na normalizovanou hloubkovou mapu ze softwaru LOCATE je vhodné pro optimální podobu modelu aplikovat gamma korekci hloubky hodnotou zhruba 3.5. Obecně se dá říci, že u všech modelů sestavených za použití hloubkové mapy generované neuronovou sítí MegaDepth jsou přechody mezi horizonty jemnější a díky tomu ve spoustě míst v modelu pak nedochází k vytvoření tak velkých děr.

6.2 Rozbor syntetických textur

První pokusy se syntézou obrazu za použití jediného vodítka `ARGET_DEPTH_MAP_ORIGINAL` generovaly slibné výsledky. Jak je vidět na výstupu pokusu na obrázku č. 6.3a, tak nově vytvořená textura je celistvá a neobsahuje zásadní grafické nedostatky. Tato konkrétní textura vznikla při pokusu reprodukovat sebe samu, neboli vstupní a výstupní vodítko bylo totožné a obsahovalo pouze hloubku. Ovšem v okamžiku, kdy experimenty začaly využívat váhovou masku, nastaly problémy, jak je vidět na výstupu 6.3b. Tato syntetická textura vznikla při kombinaci vodítka obsahujícího hloubkovou mapu a modulační masky obsahující výstup detektoru děr. Z pohledu na syntetickou texturu je na první pohled patrné, že se výstup zcela jistě nesetkává s teoreticky popsaným očekáváním. Postupným laděním vstupních parametrů a vodítek se kvalita syntetických textur značně zlepšila, ale fragmenty vyplňující původní díry zcela nebyly navázány na reprodukci obrazu. Ukázkou takto nedokonale navázané záplaty je obrázek č. 6.3d.

(a) Použití hloubky jako vodítka

(b) Použití bílé váhy

(c)
První použití detektoru děr

(d) Viditelný dopad detektoru děr
na syntetickou texturu

Obrázek 6.3: Snímky pořízené k analýze chyby v knihovně Ebsynth

Pro kontrolu, zda-li vše funguje, jak má, vznikla sada testovacích vodítek v široké paletě kombinací. Hlavním indikátorem, že syntéza špatně využívá modulační masku, byl výstup zobrazený na obrázku č. 6.3b. Syntetizačnímu API byla podána stejná vstupní data, jako v případě vytváření obrázku č. 6.3a s tím rozdílem, že součástí vstupních dat byla modulační

maska obsahující jednolitou bílou barvu. Očekávaný výstup by tedy měl být silně podobný, ale výstup 6.3b se značně odlišuje od předcházejícího 6.3a. Z výsledků testovací sady vznikl krátký přehled nekorektního chování syntetizační knihovny Ebsynth, který byl poskytnut jejímu autorovi. Analýzou poskytnutých dat následně došlo k objevení skutečné chyby v API Ebsynth. Následně byla chyba úspěšně opravena. Změny provedené opravou jsou k dispozici na repozitáři API Ebsynth[3] [4].

Po opravení této chyby se na výsledcích syntézy skutečně začaly projevovat výhody užití modulační textury tak, aby při syntéze textury došlo skutečně k plynulému napojení záplat na reprodukci modelu. Ukázkou je obrázek č. 6.4. Díry v reprodukci modelu 6.4a viditelné díky pohledu virtuální kamery na bok modelu jsou zcela zaplněny záplatami. Vizuálně nejpovedeněji působí záplaty vytvořeny v okolí jezera na obrázku 6.4b.

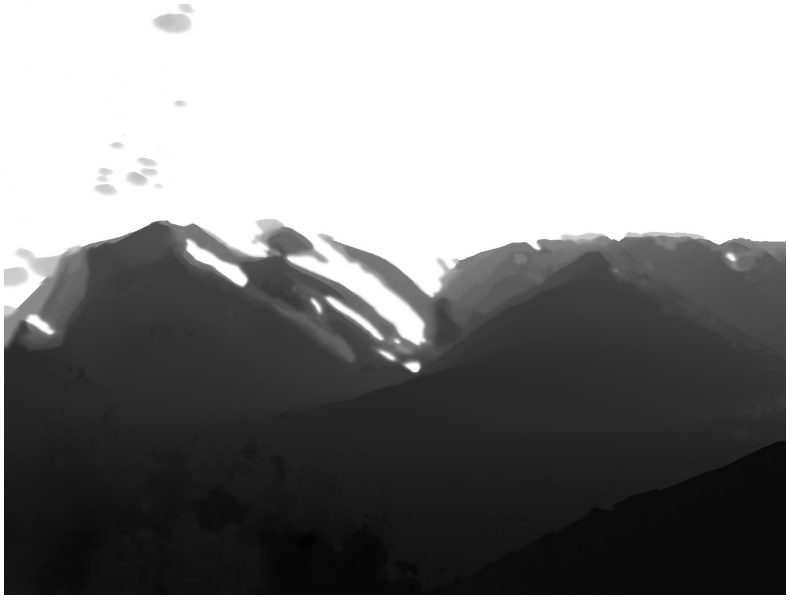
(a) Reprodukce modelu obsahující díry

(b) Nová textura vytvořená syntézou

Obrázek 6.4: Ukázka úspěšné syntézy

[3] Dostupné na: https://github.com/jamriska/ebsynth/commit/d77e03742f29b335f1e9dd43b6fa5d0c82ee7a72
[4] Dostupné na: https://github.com/jamriska/ebsynth/commit/2f5c97c0c21a86bb7334dee61453623e6a3d41c3

Pro rozsáhlejší ukázku schopností využití vedené syntézy byla do implementace přidána možnost syntetizovat i novou hloubkovou mapu. Jedná se čistě o experimenty poukazující na potenciál technologie k alternativnímu využití. Na obrázcích č. 6.5 je jasně vidět potenciál syntézy rozpoznat jednotlivé horizonty. Bohužel, jak jde vidět na obrázku č. 6.5a, tak v místech, kde se nachází modrý opar nad krajinou, dojde k mylné detekci modrého pozadí. Snímek č. 6.5a vznikl na konci procesu utváření prostorového modelu z fotografie č. D.1. Model byl doplněn o velké množství záplat tak, aby při pohledu na něj z bočního úhlu (zhruba 30°) nebyly viditelné žádné velké díry. Z tohoto pohledu byla syntetizována nová hloubková mapa za pomocí tříkanálového vodítka obsahujícího reprodukci modelu. Oproti tomu výstup č. 6.5b byl vytvořen pouze za pomoci přiblížení kamery k modelu sestaveného z fotografie č. D.5.



(a) Synteticky generovaná hloubka z reprodukce pootočení modelu

(b) Synteticky generovaná hloubka z reprodukce přiblížení modelu

Obrázek 6.5: Ukázka syntézy hloubky z reprodukce modelu

6.3 Záplaty na modelu

Kvalita jednotlivých záplat modelu se značně odvíjí od faktu, jak moc se zdařilo syntéze při sestavování syntetické textury navázat na reprodukci obrazu. Příkladem může být obrázek č. 6.6, který byl pořízen jako výsledek automatizovaného doplnění všech záplat z výsledku syntézy zobrazeného dříve na obrázku č. 6.4b. Po aplikaci těchto záplat na model získává virtuální kamera mnohem větší manévrovací prostor při pohybu kolem modelu při vytváření video sekvence prezentující fotografii.

Finální implementace sice obsahuje metodu plnící syntetizační API podle předem předpřipravených parametrů, ale jejich použití nemusí vždy ideálně fungovat. Tyto parametry byly určeny experimentálně tak, aby pokud možno univerzálně fungovaly na všech modelech a využily potenciál dat dostupných z modelu. Jedná se ale tedy o kompromis, který ne vždy musí produkovat optimální výsledky. Z tohoto důvodu je největší komponentou grafického uživatelského rozhraní panel sloužící k nastavení jednotlivých vodítek syntézy, aby mohl uživatel případně doladit parametry specifickým potřebám konkrétního modelu, nebo jediné záplatě.

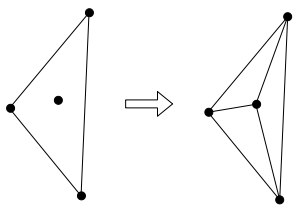
Obrázek 6.6: Záplaty aplikované přímo na modelu
z výsledku syntézy zobrazené na obrázku č. 6.4b

Kapitola 7

Možnosti dalšího rozšíření

Finální aplikace této diplomové práce demonstruje možnosti, jak doplnit prostorový model vytvořený z jediné fotografie a její hloubkové mapy. Z aplikace je možné ukládat video sekvenci, která je vytvořena pohybem virtuální kamery, nebo ukládat jednotlivé snímky poskytující jiný perspektivní pohled na původní fotografii. Vytvořené záplaty jsou ve své podstatě téměř optický klam, což ale není důležité. Důležité je, že opticky nepůsobí rušivým dojmem. Jelikož syntéza záplat modelu je částečně manuální činností, bylo by vhodné mít možnost z aplikace model ukládat a zpětně načítat. Zajímavé výsledky by také určitě mohlo přinést detailnější zpracování hloubkové mapy před procesem vytvoření samotného modelu. Například se pokusit zkombinovat dvě hloubkové mapy z různých zdrojů tak, aby model obsahoval to nejlepší z obou. Alternativou tohoto přístupu by mohla být integrace detektorů objektů, která by vymaskovala hloubky jednotlivých objektů tak, aby nemohlo dojít k rozpůlení objektů, popřípadě osob, o kterých nejsou žádné informace v hloubkové mapě. Takováto integrace by navíc mohla i potenciálně zlepšit kvalitu syntézou vytvořených textur, jelikož její výstup integrovaného detektoru by se dal použít jako další vodící kanál.

Jednou z dalších možností, jak prezentovat vytvořený model, je dedikovaný prohlížeč simulující rozhlížení hlavou. Pro potřeby takovéto aplikace by bylo vhodné prostorové záplaty přímo integrovat do modelu tak, aby došlo ke spojení všech bodů ve scéně do jedné sítě. To by ovšem vyžadovalo rozbití sítě původní, jak je ilustrováno na obrázku č. 7.1. Druhou variantou by mohla být aplikace algoritmů zabývajících se rekonstrukcí povrchů z mračna bodů. Vhledem k tomu, že již původní model obsahoval nemalé množství trojúhelníků, a to bylo ještě rozšířeno o trojúhelníky záplat, bylo by vhodné také zaintegrovat algoritmus sloužící k jejich decimaci. Zde je vhodné podotknout, že při decimaci by bylo třeba brát v potaz i barevnou informaci uloženou v jednotlivých bodech. Toto řešení by pravděpodobně vedlo k vytvoření nové modelové textury, která by byla vytvořena ze všech bodů modelu, a následně by tato textura byla použita k otexturování zdecimovaného modelu. Takto zjednodušený model by bylo reálné vykreslovat například i na mobilních zařízeních bez větších problémů.



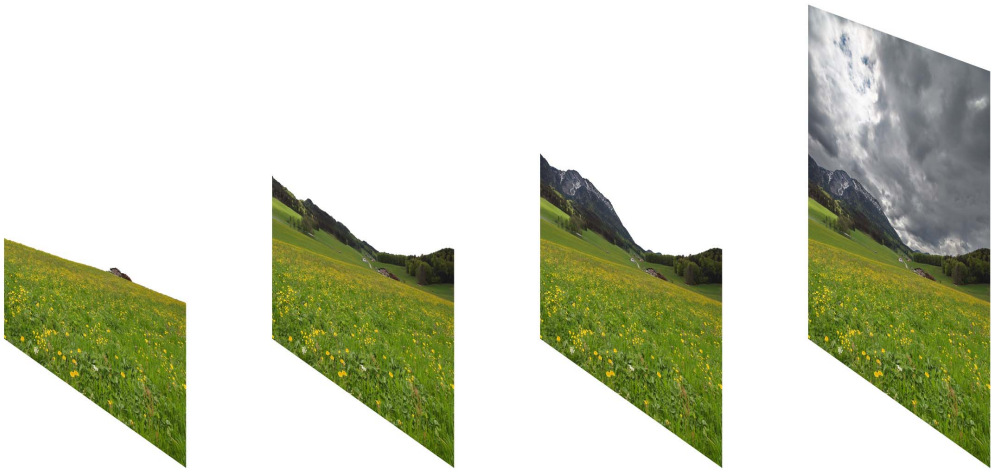
Obrázek 7.1: Integrace bodů záplat do modelu

7.1 Alternativní implementace

Inspirací k alternativnímu řešení byla více plátnová kamera zmíněná v kapitole 3. Jak je vidět na dříve zmíněných obrázcích 5.12 a 5.13 demonstrujících zvýšení pestrosti aplikovaných záplat za pomoci úpravy hloubky na modelu, tak syntetizační API poměrně zdatně dokáže i z malého množství zdrojového vzorku vytvořit rozmanitou texturu. Toho může být využito k alternativnímu vytvoření obrazu skrytého za horizonty. V alternativním návrhu by tedy nedocházelo k vytvoření prostorového modelu, jak je tomu ve finální implementaci, ale k vytvoření několika virtuálních pláten. Obraz na plátnech by byl při vytvoření totožný, přičemž by se ke každému plátnu vytvořila maska v podobě alfa kanálu. Vytvoření těchto masek by vycházelo ze vstupní hloubkové mapy podle prahů. Tyto prahy by bylo nutné pečlivě volit tak, aby původní fotografie byla rozdělena v místech horizontů. Ilustrace takového rozdělení je na obrázku č. 7.2. Takovéto rozdělení vstupní fotografie by bylo klíčové a algoritmicky nejsložitější, a to především z toho důvodu, aby nedošlo k rozpůlení vizuálně důležitých částí.

Nad takto rozdělenými plátny by byla následně provedena syntéza textur u částí, které jsou překryty plátny předchozími. Tím by došlo k vytvoření několika souvislých pláten, které jsou již opticky rozdělené. Pro vytvoření prostorového vjemu by následně posloužila technika popsaná v kapitole č. 5.3.1. Výhodou takovéto implementace by bylo, že by nebylo třeba složitě nastavovat parametry syntézy. Pravděpodobně by jednoduše stačilo využít hloubkové mapy jako jediného vodítka. Případně by se dal využít i původní vstupní obraz doprovázený maskou ohraničující předěl plátna.

V případě, že by výsledný prostorový efekt při pohybu kamery mezi jednotlivými plátny nebo pohyb pláten samotných nepůsobil dostatečně prostorově, bylo by možné jednotlivá plátna částečně prostorově deformovat pomocí původní hloubkové mapy. Tím by se podoba celé této alternativy značně přiblížila ke stávající implementaci s využitím modelu obsahujícího díry, dříve zmíněného v kapitole 5.2.



Obrázek 7.2: Vizualizace rozdělení fotografie do několika rovin

Kapitola 8

Závěr

Tato diplomová práce se zabývala problematikou dynamického prezentovaní statické fotografie za užití hloubkové mapy. Vysvětluje principy, jakými byla fotografie a hloubková mapa pořízena. Dále na základě analýzy, jakým způsobem je možné fotografie dynamicky prezentovat, je navržen algoritmus. Cílem tohoto algoritmu je řízení programu, který poslouží k vytvoření videosekvence z jedné fotografie a její hloubkové mapy. Prakticky získané informace z experimentování s fotografií a její hloubkovou mapou byly využity k vyhotovení algoritmu, na základě kterého lze sestavit prostorový model. Text v této části také pojednává o problémech spojených s vytvářením takového modelu. Jednotlivé úkony vedoucí k úspěšnému sestrojení prostorového ortogonálního modelu jsou v textu diskutovány a zdůvodněny. Nachází se zde i rozbor alternativních přístupů, u kterých se zpravidla vyskytuje zdůvodnění, proč nebyly použity ve finální implementaci.

Jedním z hlavních výsledků této práce je princip popisující rekonstrukci původního obrazu fotografie při pohledu na model perspektivní kamerou. Kombinace ortogonálního modelu a perspektivní kamery byla zvolena z důvodu vzniku paralax efektu při pohybu kamery v okolí modelu tak, aby vynikla hloubka objektů na původní fotografii. Z experimentů s prostorovým modelem vyplývá, že kombinací fotografie a hloubkové mapy není problematické vytvořit věrohodný virtuální model, který je při kolmém pohledu zepředu vizuálně shodný se zdrojovou fotografií. Dále je z experimentů patrné, že při pohybu virtuální kamery kolem modelu může dojít k vykreslení nevzhledných fragmentů vznikajících v oblastech modelu, které jsou však zakryté objekty v popředí na původní fotografii. Tyto fragmenty jsou označovány jako díry. Ty je možné detekovat a následně nahradit novou synteticky vytvořenou záplatou.

Výstupem praktické části je aplikace implementovaná v jazyce C++ využívající vykreslovací řetězec OpenGL. Aplikace umožňuje vytvoření video sekvence zaznamenávající lineární pohyb virtuální kamery v okolí prostorového modelu vytvořeného ze zdrojové fotografie a její hloubkové mapy. Za běhu aplikace je možné dynamicky měnit parametry vhledu vykreslovaného modelu stejně jako dráhu pohybu virtuální kamery. Tento model může být poté vyplňován dalšími obrazovými informacemi za pomocí vedené syntézy textur tak, aby při pohybu virtuální kamery v okolí modelu nedošlo k vykreslení takzvaných děr. Parametry syntézy jsou do značné míry uživatelem konfigurovatelné tak, aby výsledky syntetických záplat dosahovaly co nejlepších vizuálních kvalit.

Literatura

[1] Abdulla, W.: Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow. https://github.com/matterport/Mask_RCNN, 2017.

[2] Adobe Creative Cloud: *#MovingStills: Adobe MAX 2018*. [Online; navštíveno 13.01.2019].
URL https://www.youtube.com/watch?v=dM-lX9c3Pqw

[3] Barnes, C.; Shechtman, E.; Finkelstein, A.; aj.: PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, ročník 28, č. 3, Srpen 2009.

[4] Bonneel, N.; Van De Panne, M.; Lefebvre, S.; aj.: Proxy-Guided Texture Synthesis for Rendering Natural Scenes. 05 2010, doi:10.2312/PE/VMV/VMV10/087-095.

[5] Cambridge in Colour: *DIGITAL PHOTOGRAPHY TUTORIALS*. [Online; navštíveno 13.01.2019].
URL https://www.cambridgeincolour.com/tutorials.htm

[6] Gardiner, M.: *How to Make an Animated Parallax Video from a Photo*. [Online; navštíveno 13.01.2019].
URL https://photography.tutsplus.com/tutorials/how-to-animate-a-still-photograph-in-photoshop--cms-29237

[7] Gregory, J.: *Game engine architecture*. Wellesley, Mass: A K Peters, 2009, ISBN 978-1568814131.

[8] Ivan Dylevský, P. J.: *Základy funkční anatomie člověka*. [Online; navštíveno 13.01.2019].
URL http://vos.palestra.cz/skripta/anatomie/15a2a5.htm

[9] Jamriska, O.: Ebsynth: Fast Example-based Image Synthesis and Style Transfer. https://github.com/jamriska/ebsynth, 2018.

[10] Jhou, W.-C.; Cheng, W.-H.: Animating Still Landscape Photographs Through Cloud Motion Creation. *IEEE Transactions on Multimedia*, ročník 18, 01 2015: s. 1–1, doi:10.1109/TMM.2015.2500031.

[11] Lengyel, E.: *Mathematics for 3D game programming and computer graphics*. Boston: Course Technology PTR, 2012, ISBN 978-1-4354-5886-4.

[12] Lengyel, E.: *Foundations of game engine development : Volume 1: mathematics*. Lincoln, California: Terathon Software LLC, 2016, ISBN 978-0-9858117-4-7.

[13] Li, Z.; Snavely, N.: MegaDepth: Learning Single-View Depth Prediction from Internet Photos. In *Computer Vision and Pattern Recognition (CVPR)*, 2018.

[14] OpenCV dev. team: *How to scan images, lookup tables and time measurement with OpenCV*. [Online; navštíveno 13.01.2019].
URL https://docs.opencv.org/3.0-beta/doc/tutorials/core/how_to_scan_images/how_to_scan_images.html

[15] OpenCV dev. team: *Mat - The Basic Image Container*. [Online; navštíveno 13.01.2019].
URL https://docs.opencv.org/3.0-beta/doc/tutorials/core/mat_the_basic_image_container/mat_the_basic_image_container.html

[16] OpenCV dev. team: *OpenCV*. [Online; navštíveno 13.01.2019].
URL https://opencv.org/

[17] Raab, F.: Hloubková mapa z dvojice obrazů. 2013.
URL http://hdl.handle.net/11012/54950

[18] Raskar, R.: *Computational photography : mastering new techniques for lenses, lighting, and sensors*. Natick, Mass: A.K. Peters, 2009, ISBN 978-1568813134.

[19] Sellers, G.; Jr., R. S. W.; Haemel, N.: *OpenGL Superbible: Comprehensive Tutorial and Reference (7th Edition)*. New York: Addison-Wesley Professional, 2015, ISBN 978-0-672-33747-5.

[20] Spoon Graphics: *2.5D Parallax Photo Effect Photoshop Tutorial*. [Online; navštíveno 13.01.2019].
URL https://www.youtube.com/watch?v=XdYEzui3Ttc

[21] The Khronos Group Inc.: *The Industry's Foundation for High Performance Graphics*. [Online; navštíveno 13.01.2019].
URL https://www.khronos.org/opengl/

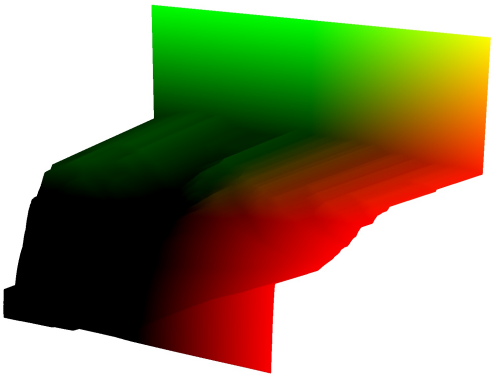
[22] Wikipedia: *Parallax scrolling*. [Online; navštíveno 08.05.2019].
URL https://en.wikipedia.org/wiki/Parallax_scrolling

[23] Wikipedia: *Pinhole camera model*. [Online; navštíveno 13.01.2019].
URL https://en.wikipedia.org/wiki/Pinhole_camera_model

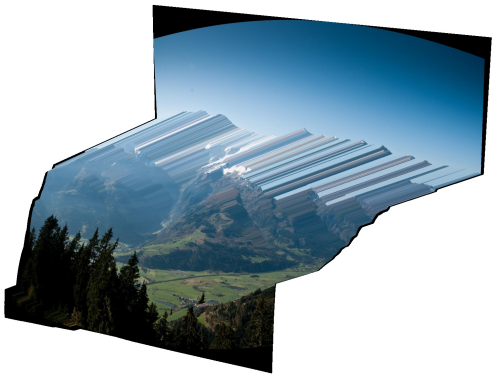
[24] Čadík, M.: *LOCATE - Visual Localization in Natural Environments*. [Online; navštíveno 13.01.2019].
URL http://cadik.posvete.cz/locate/

Příloha A

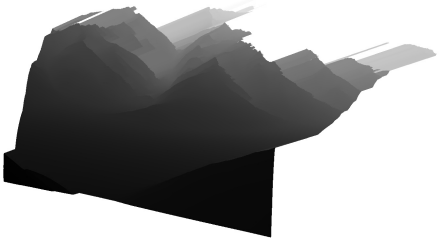
Výstup vykreslovacího procesu



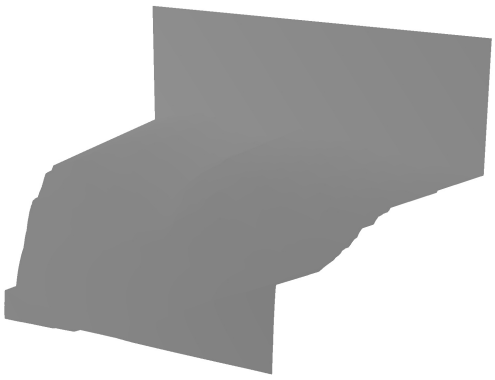
Obrázek A.1:
TARGET_COORDINATES



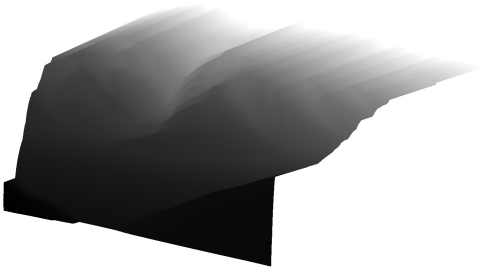
Obrázek A.2:
TARGET_TEXTURE



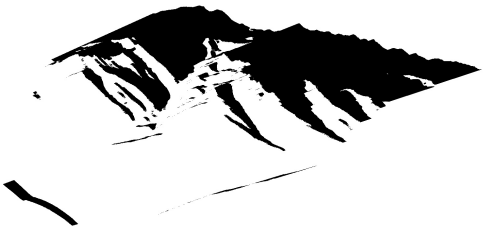
Obrázek A.3:
TARGET_DEPTH_MAP



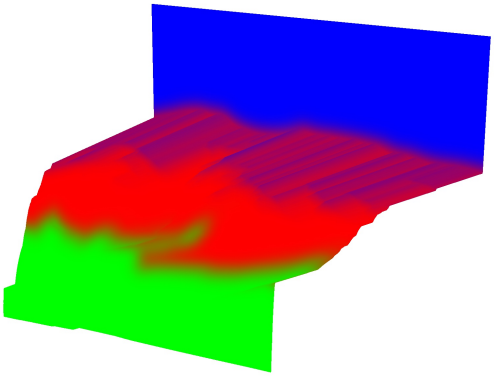
Obrázek A.4:
TARGET_DEPTH_MAP_NORMALIZED



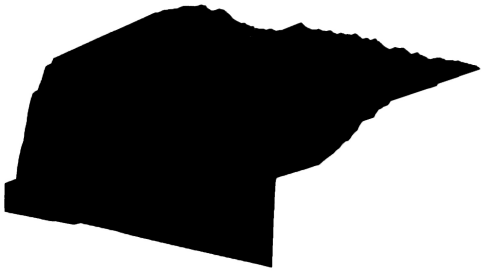
Obrázek A.5:
TARGET_DEPTH_MAP_ORIGINAL



Obrázek A.6:
TARGET_HOLES



Obrázek A.7:
TARGET_EXTRA_1



Obrázek A.8:
TARGET_EXTRA_2

Příloha B

Váhové masky pro syntézu

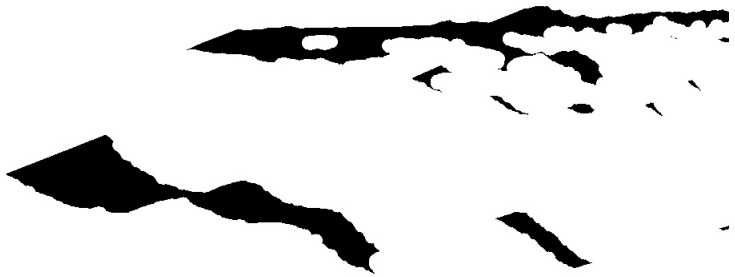
Obrázek B.1: HOLES

Obrázek B.2: HOLES_INVERSE

Obrázek B.3: BLUR_7X7

Obrázek B.4: BLUR_17X17

Obrázek B.5: DILATED_7X7



Obrázek B.6: DILATED_17X17

Obrázek B.7: ERODED_7X7

Obrázek B.8: ERODED_17X17

Obrázek B.9: BLUR_AFTER_DILATATION_7X7

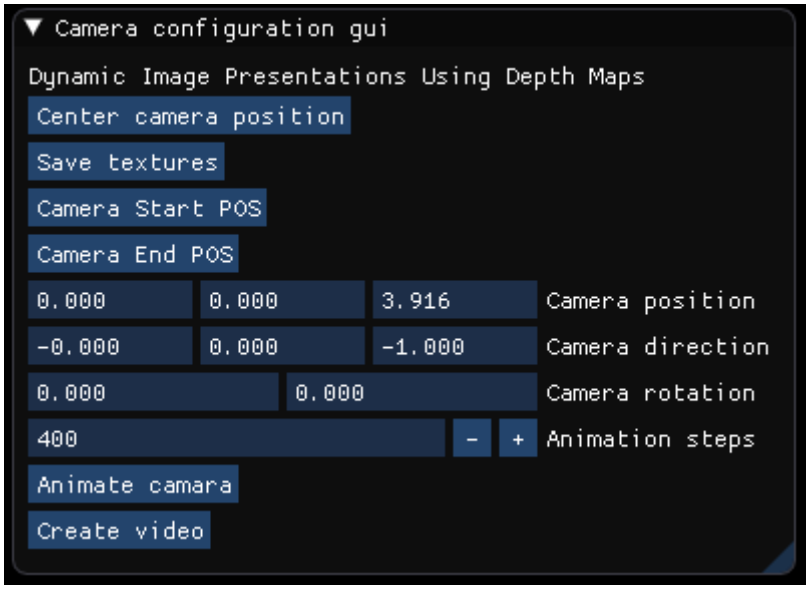
Obrázek B.10: BLUR_AFTER_DILATATION_17X17

Obrázek B.11: BLUR_AFTER_EROSION_7X7

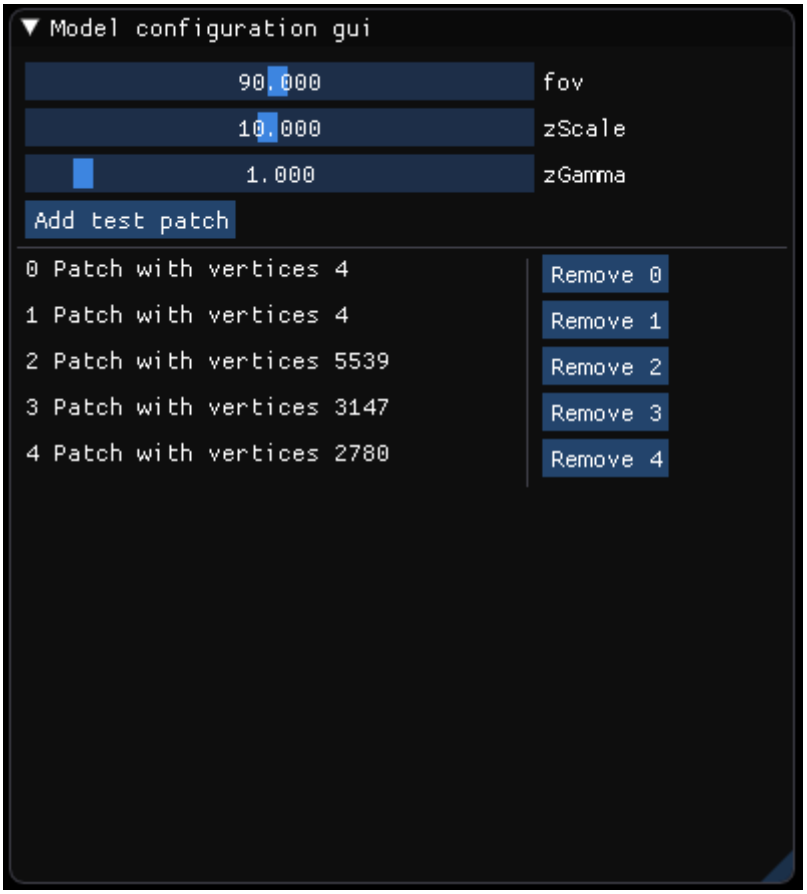
Obrázek B.12: BLUR_AFTER_EROSION_17X17

Příloha C

Grafické uživatelské rozhraní



Obrázek C.1: Vizualizace grafického uživatelského rozhraní z výsledné aplikace obhospodařující parametry kamery



Obrázek C.2: Vizualizace grafického uživatelského rozhraní z výsledné aplikace obhospodařující prostorový model

Obrázek C.3: Vizualizace grafického uživatelského rozhraní z výsledné aplikace obhospodařující nastavení syntézy textur

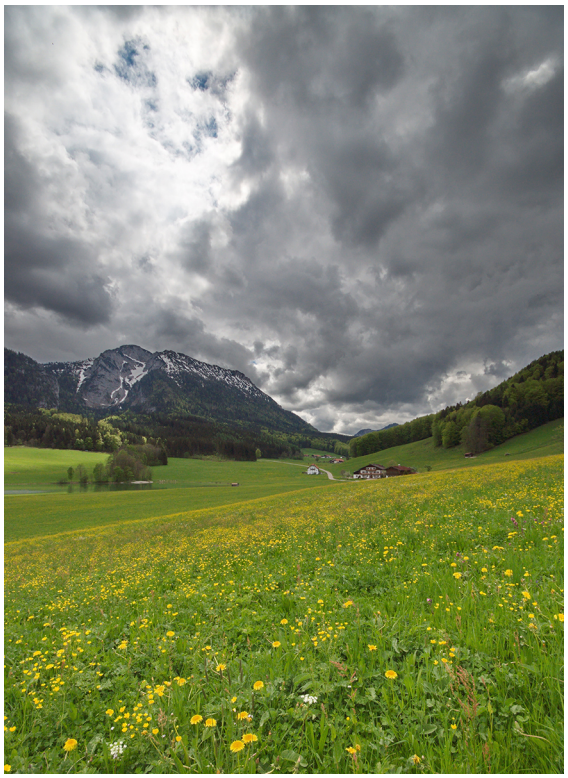
Příloha D

Testovací sada

Obrázek D.1: Fotografie na která byla primárně používána při vývoji, většina snímků demonstrujících vlastnosti výsledné implementace byla pořízena na základě tohoto snímku.

Obrázek D.2: Test1

Obrázek D.3: Test2



Obrázek D.4: Test3

Obrázek D.5: Test4

Obrázek D.6: Test5

Příloha E

Obsah DVD

Příloho této diplomové práce je i DVD obsahující výslednou implementaci, zde následuje rozpis jednotlivých složek:

- `build` je složka obsahující přeloženou výslednou aplikaci se všemi potřebnými dll knihovnami tak, aby bylo možné ji rovnou spustit. Je vhodné připomenou, že aplikace pro svůj běh vyžaduje grafickou kartu podporující grafický standart OpenGL 4.5. Dalším omezením je knihovna Ebsynth, ta pro běh syntézy textur vyžaduje prostředí CUDA. Obsahuje sice i implementaci určenou na CPU, ale tato implementace nedokáže využít modulačních dat. Ve složce se dále nachází složka `shaders` obsahující zdrojové kódy k shader programům. Jejich umístění je relativní k samotnému spuštění aplikace.
- `src` je složka obsahující zdrojové kódy aplikace. Jedná se o přenesení z vývojového repositáře. Ve složce se dále nachází soubor `CMakeLists`, který slouží k vytvoření projektu v prostředí Visual Studio za použití nástroje cmake. Samozřejmostí jsou přiložené všekré potřebné knivny ve složece `externals`.
- `video` je složka obsahující demostrační video výstupy výsledné aplikace.
- Tato textová část diplomové práce

Příloha F

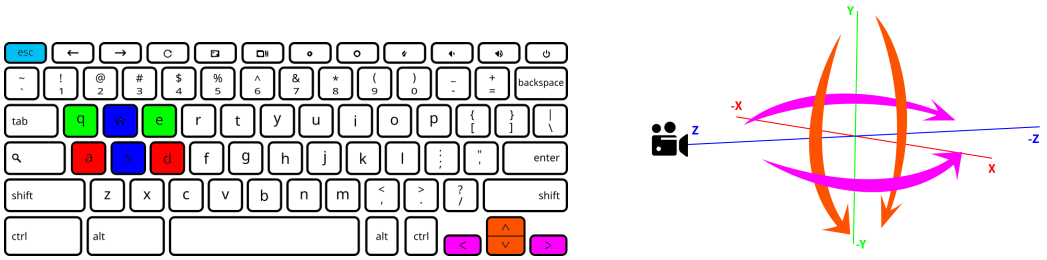
Návod k použití aplikace

Demonstrační aplikace se spouští pomocí příkazové řádky. Není-li zadán žádný argument aplikace, program se pokouší vytvořit model s ukázkových dat z fotografie D.1. Při zadáni jedno jediného argumentu je možné spustit program na poskytnutých testovacích datech, kde tvar argumentu je písmeno 't' následované číslem zvolené fotografie. Přidáním písmene 'm' dojde k užití hloubkové mapy generované neuronovou sítí MegaDepth. Poslední možností spuštění programu je pomocí až čtyř cest ke zdrojovým obrázkům ve tvaru vstupní fotografie, hloubková mapa a následují dvě volitelné textury.

```
DIPUDM.exe t1
DIPUDM.exe t1m
DIPUDM.exe t2
DIPUDM.exe t2m
DIPUDM.exe t3
DIPUDM.exe t3m
DIPUDM.exe t3man
DIPUDM.exe t4
DIPUDM.exe t4m
DIPUDM.exe t5
DIPUDM.exe t5m

DIPUDM.exe cesta_k_fotografii cesta_k_hloukoveMape
           [cesta_k_texture1] [cesta_k_texture2]
```

Výpis F.1: přehled všech argumentů programu



(a) Mapa kláves ovládání

(b) Mapování kláves na ovládání os

Obrázek F.1: Vizualizace ovládání virtuální kamery