Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Paralelné filtrovanie štruktúrovaných mračien bodov vedené intenzitnou textúrou v CUDA

Bakalárska práca

2021
Martin Melicherčík

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Paralelné filtrovanie štruktúrovaných mračien bodov vedené intenzitnou textúrou v CUDA

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	RNDr. Martin Madaras, PhD.

Bratislava, 2021
Martin Melicherčík

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Martin Melicherčík
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Paralelné filtrovanie štruktúrovaných mračien bodov vedené intenzitnou textúrou v CUDA
Parallel Filtering of Structured Point Clouds Guided by Intensity Texture in CUDA

Anotácia: Pri 3D skenovaní pomocou skenerov so štruktúrovaným svetlom sú vypočítané hĺbkové mapy zo zachytených intenzitných obrázkov. Keď máme známu a nakalibrovanú kameru, vieme hĺbkové mapy konvertovať na usporiadané mračno bodov. Usporiadané mračná bodov sú podobné štandardným obrázkom, sú to 2D matice bodov, kde každá vzorka nesie informáciu o pozícii v 3D priestore. Tieto štruktúrované mračná bodov zvyčajne obsahujú šum, ktorý treba odfiltrovať. Napríklad bilaterálny filter vieme efektívne aplikovať na tieto dáta, ale filtrovanie nie je ideálne, ak je veľký uhol medzi normálou skenovaného povrchu a smerom kamery. Pre rýchle a robustné filtrovanie potrebujeme použiť iné prístup filtrovania, kde môže byť intenzitná textúra použitá ako nápomocná informácia.

Cieľ: Cieľom práce je navrhnúť a implementovať paralelný GPU filter na usporiadané mračná bodov v CUDA. Analyzujte metódy pre filtrovanie usporiadaných mračien bodov. Navrhnite a implementujte metódu na GPU pre filtrovanie usporiadaných mračien bodov, ktorá môže používať intenzitnú textúru ako nápomocnú informáciu pri filtrovaní. Porovnajte navrhnutú metódu s existujúcimi metódami a vyhodnoťte výsledky kvalitatívnym a kvantitatívnym spôsobom.

Literatúra:

1. Kaiming He et al. 2010, Guided Image Filtering
http://kaiminghe.com/eccv10/

2. Longquan Dai et al. 2017, Hardware-Efficient Guided Image Filtering for Multi-label Problem
https://arxiv.org/abs/1803.00005

3. Xian-Feng Han et al. 2018, Guided 3D point cloud filtering
https://www.researchgate.net/
publication/320599421_Guided_3D_point_cloud_filtering

4. Xujie Li et al. 2011, The research on parallelized fast trilateral filter on GPU acceleration
https://ieeexplore.ieee.org/document/6066411

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Kľúčové slová: paralelné spracovanie obrazu, CUDA, 3D skener, štruktúrované svetlo, Tegra TX2

Vedúci: RNDr. Martin Madaras, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 16.10.2020

Dátum schválenia: 25.10.2020 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Touto cestou by som rád poďakoval svojmu vedúcemu bakalárskej práce RNDr. Martinovi Madarasovi, PhD. za jeho ochotu, odborné rady a usmernenia. Vďaka patrí aj Ing. Andrejovi Fúsekovi, ktorý pre túto prácu generoval vstupné dáta a bol odbornou radou veľmi nápomocný pri evaluácii výsledkov.

Abstrakt

V práci sa zaoberáme filtrovaním štruktúrovaných mračien bodov vygenerovaných 3D skenerom na báze štruktúrovaného svetla. Cieľom filtrovania je vyhladiť povrch objektov skenovanej scény a pritom zachovať ich hrany ostré. Tento problém rieši známy *bilaterálny* filter, zlyháva však, ak je v scéne veľký uhol normály skenovaného povrchu objektu od kamery skenera. Vtedy povrch nevyhladí dobre. Takýto povrch zvládne vyhladiť *trilaterálny* filter, ktorý je však výpočtovo zložitejší. Chceme nájsť a implementovať rýchlu robustnú filtrovaciu techniku, ktorá takéto povrchy zvládne v čase podobnom bilaterálnemu filtru. Ako pomocnú informáciu pri filtrovaní môžeme použiť intenzitnú textúru scény z kamery skenera. Filtrovacou techikou, ktorú sme v tejto práci vybrali a implementovali na CPU a potom paralelne na GPU je *guided* filter. Tento filter používa pomocný vstupný obraz. Intenzitnú textúru treba pre guided filter upraviť, aby spĺňala jeho predpoklady. Po úprave intenzitnej textúry aplikujeme guided filter na dáta zo skenera a porovnáme jeho efektiviu a výsledky s bilaterálnym filtrom.

Kľúčové slová: paralelné spracovanie obrazu, CUDA, 3D skener, štruktúrované svetlo, guided filter

Abstract

In this thesis we deal with the filtering of structured point clouds from a 3D scanner based on structured light. The purpose of filtering is to smooth surface of objects in the scanned scene while keeping their edges sharp. This problem is solved by a well-known *bilateral* filter, but this technique fails if there is a large angle between normal of scanned surface and the scanner camera. Then the surface will not smooth well. Such a surface can be smoothed out by *trilateral* filter, which is computationally more complex. We want to find and implement a fast, robust filtration technique that can handle these surfaces in time similar to bilateral filter. We can use intensity texture of scenes from the scanner camera as an auxiliary information. The filtering technique that we have selected and implemented in this work on the CPU and consequently using parallelism on the GPU, is a *guided* filter. This filter uses an auxiliary input image. The intensity texture needs to be corrected for guided filter to meet its requirements. After correcting the intensity texture, we apply a guided filter to the data from the scanner and compare its effectiveness and results with a bilateral filter.

Keywords: parallel image processing, CUDA, 3D scanner, structured light, guided filter

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

V tejto práci sa budeme venovať problematike filtrovania 3D dát. Virtuálna reprezentácia 3D objektov má v dnešnej dobe veľa aplikácií, ako napr. validácia vyrobených komponentov alebo analýza archeologických nálezov. Existuje veľa zariadení založených na rôznych techologických princípoch, ktoré dokážu naskenovať 3D objekty a reprezentovať ich pomocou takzvaného štruktúrovaného mračna bodov. Štruktúrované mračná bodov majú podobnú štruktúru ako štandardné obrázky. Sú to 2D matice bodov, kde každá vzorka nesie informáciu o pozícii v 3D priestore [15]. Zariadenia vedia 3D objekty naskenovať s istou presnosťou. Povrch naskenovaných objektov býva zašumený. Aby objekty čo najviac zodpovedali realite je potrebné ich zašumený povrch vyhladiť. Pri vyhladzovaní povrchu však nesmieme vyhladiť ich ostré hrany, tým by sme stratili dôležité detaily skenu.

Tento problém riešia takzvané hrany-zachovávajúce filtre. Známym príkladom takéhoto filtru je *bilaterálny* filter [16], ktorý má na 3D dátach veľmi dobré výsledky, ale v niektorých situáciach zlyháva. Ak je uhol normály povrchu objektu v skenovanej scéne od kamery skenera veľký, bilaterálny filter ho nevyhladí dobre. Problém rieši *trilaterály* filter [4], ktorý je založený na tom istom matematickom modeli ale je výrazne výpočtovo náročnejší. Ak chceme dobre filtrovať 3D scénu obsahujúcu problémové povrchy objektov rýchlejšie ako trilaterálnym filtrom, mohli by sme využiť filter založený na inom matematickom modeli. Skenovacie zariadenia vedia okrem 3D dát reprezentujúcich skenovanú scénu urobiť aj jej fotku. Táto fotka by nám mohla slúžiť ako pomocná informácia pri filtrovaní. Populárny filtrovací prístup, ktorý využíva pomocný obraz je *guided* filter [8]. Tento filter by mal byť, tak ako je definovaný, rýchlejší ako bilaterálny filter a na povrchoch s veľkým uhlom normály od kamery skenera by nemal zlyhať. V tejto práci budeme implementovať guided filter. Chceme overiť, či v čase porovnateľnom s bilaterálnym filtrom vieme na jeho problémových oblastiach obrazu dosiahnuť lepšie výsledky. Guided filter budeme implementovať aj paralelne na GPU, pomocou platformy CUDA. Chceme tým docieliť čo najväčšiu rýchlosť a efektivitu filtrovania.

V nasledujúcej kapitole budeme analyzovať dáta vygenerované pre túto prácu a opíšeme zariadenie, ktoré ich vytvorilo. Potom konkrétnejšie opíšeme vyššie uvedené filtrovacie techniky a zdôvodníme výber guided filtra ako filtrovacej techniky, ktorú budeme implementovať. Následne guided filter podrobne rozoberieme a preskúmame

možnosti jeho aplikácie na vstupné dáta. Opisovať budeme aj našu implementáciu filtra a následne jej paralelizáciu. Nakoniec uvedieme výsledky aplikácie guided filtra na naše vstupné dáta a zhodnotíme efektivitu jeho paralelizácie na GPU. Výsledky aj čas behu filtrovania porovnáme s bilaterálnym filtrom. Nakoniec budeme aplikovať bilaterálny a nami implementovaný guided filter na scénu, kde bilaterálny filter zlyháva a preskúmame, či na nej bude mať guided filter lepšie výsledky.

Kapitola 1

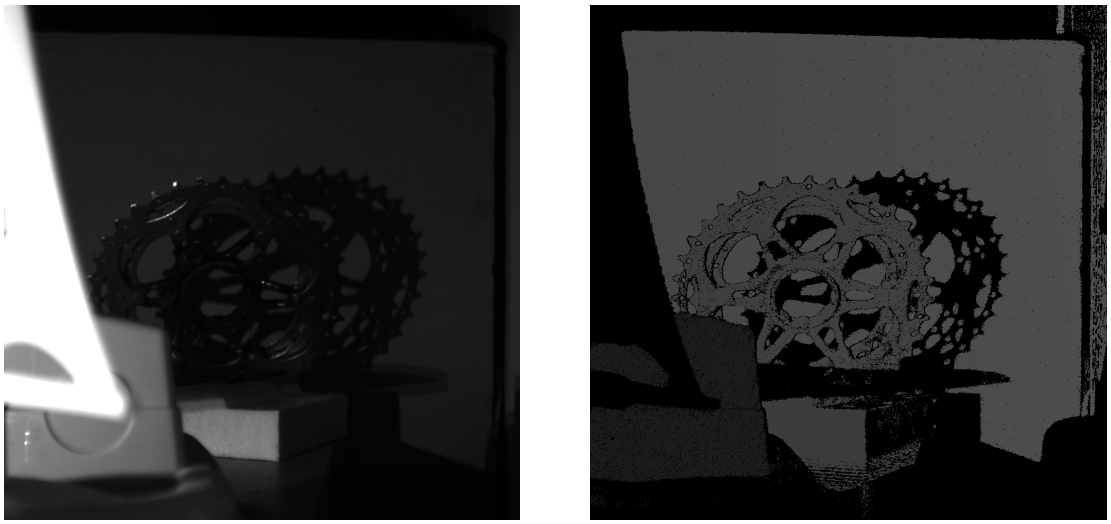
3D skenovanie a mračná bodov

Cieľom tejto práce je vybrať a aplikovať vhodnú filtrovaciu techniku na 3D dáta. Vstupným obrazom pre filtrovanie teda bude 3D sken scény obsahujúcej viac objekov. Existuje veľa skenerov založených na rôznych technických princípoch, ktoré takéto dáta dokážu generovať. Z dát vygenerovaných skenerom vieme vypočítať štruktúrované mračno bodov reprezentujúce danú scénu. Štruktúrované mračná bodov majú rovnakú štruktúru ako štandardné obrázky akurát v sebe nesú 3D informáciu. Sú to 2D matice bodov, kde každá vzorka nesie informáciu o pozícii v 3D priestore [15]. V tejto kapitole sa zoznámime so skenerom, ktorý pre túto prácu vygeneroval vstupné dáta, uvedieme typy a fromát dát, ktoré nám skener poskytol a nakoniec budeme tieto dáta analyzovať.

1.1 3D skener na báze štruktúrovaného svetla

Naše vstupné dáta budú generované PhoXi 3D skenerom od firmy Photoneo [12]. Hlavnými komponentami tohto skenera sú vedľa seba pevne umiestnené laser a kamera. Skenovanie scény funguje tak, že laser nasvieti na scénu postupne viac projekcií v podobe vopred definovaných mriežok. Mriežky sa na objektoch scény deformujú. Takéto scény s deformovanými mriežkami postupne nasníma kamera s rozlíšením 2060x1544 pixelov. Skener dokáže na základe vopred definovaných mriežok a známej relatívnej pozície kamery a lasera zo snímky vypočítať 2D maticu, ktorej každá vzorka nesie informáciu o vzdialenosti daného bodu od kamery skenera. Takáto 2D matica sa nazýva hĺbková mapa scény. Z hĺbkovej mapy vieme na základe toho, že poznáme nakalibrovanú kameru skenera, vypočítať štruktúrované mračno bodov.

Okrem lasera a kamery má skener aj ledku, ktorou môžeme skenovanú scénu osvietiť. Môže teda urobiť kamerou fotku osvietenej scény. Dáta, ktoré nám sú zo skenera sprostredkované sú hĺbková mapa skenovanej scény a fotka skenovanej scény osvietenej ledkou. Fotku skenovanej scény budeme nazývať intenzitnou textúrou scény. Toto pomenovanie vychádza z toho, že fotka z kamery je čiernobiela a teda hodnota každého



Obr. 1.1: Časť intenzitnej textúry (vľavo) a hĺbkovej mapy skenovanej scény zobrazenej ako 2D čiernobiely obraz (vpravo).

pixela je intenzita v šedo-tónovej.

Predpokladáme, že sa pozícia skenera a skenovaná scéna počas skenovania nemenia. Máme teda pozične identické obrazy reprezentujúce skenovanú scénu, ktoré majú rozdielne vlastnosti.

Formát obrazov zo skenera je konfigurovateľný. My uvedieme formát obrazov, ktoré boli vygenerované pre túto prácu. Hĺbkovú mapu scény dostaneme od skenera ako 2D maticu, ktorej prvky sú 32-bitové čísla typu *float* reprezentujúce vzdialenosti bodov v scéne od kamery skenera. Intenzitná textúra je nám poskytnutá ako 2D matica 16-bitových celých čísel, ktoré reprezentujú intenzitu daného pixela v čiernobielej fotke scény.

1.2 Analýza vstupných obrazov

Uviedli sme typy obrazov, ktoré nám skener generuje. Na Obr. 1.1 sme pre ilustráciu zobrazili tú istú časť intenzitnej textúry a hĺbkovej mapy zobrazenej ako 2D čiernobiely obraz, kde je vzdialenosť pixela od kamery skenera vyjadrená intenzitou šedej v škále od čiernej k bielej. Čím je pixel v hĺbkovej mape vzdialenejší, tým je svetlejší.

Vstupným obrazom, na ktorý budeme filtrovanie aplikovať je hĺbková mapa. V tejto práci chceme preskúmať, či nám pri jej filtrovaní môže byť intenzitná textúra nápomocná. Potrebujeme teda zistiť, aký je medzi hĺbkovou mapou a intenzitnou textúrou vzťah. Teraz budme skúmať, v čom sú si intenzitná textúra s hĺbkovou mapou podobné a v čom sa líšia. Chceme zistiť, či nám intenzitná textúra vie o skenovanej scéne podať informáciu, ktorú by sme mohli pri filtrovaní hĺbkovej mapy využiť.

1.2.1 Nedefinované pixely

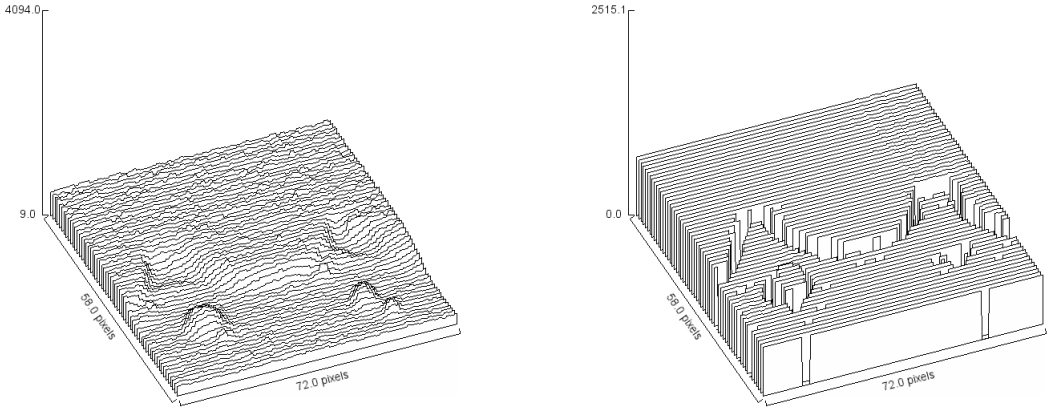
Prvým zásadným rozdielom medzi intenzitnou textúrou a hĺbkovou mapou je, že hĺbková mapa môže obsahovať nedefinované pixely. Nedefinované pixely sú také pixely hĺbkovej mapy, ktorých vzdialenosť od kamery skener nevedel na základe snímok s nasvietenou projekciou vypočítať. Dôvodom toho, že je pixel nedefinovaný môže byť napríklad to, že objekty v scéne, tak ako sú rozložené, zakrývajú kamere výhľad na niektoré časti laserovej projekcie. Ak je niektorá časť scény v zornom poli kamery zatienená nejakým objektom, nezasvieti sa na ňu laserová projekcia. Skener vtedy nemá z čoho vypočítať body povrchu objektov, a preto také pixely popíše ako nedefinované. Ďalším dôvodom toho, že sú pixely nedefinované môže byť lesklý povrch objektov. Ak sa na lesklý povrch nasvieti laserová projekcia tak sa mriežka môže deformovať nesprávnym, nedefinovaným spôsobom. Vtedy tiež skener nevie správne vypočítať vzdialenosť bodu od kamery a takéto pixely uvedie ako nedefinované. Samozrejme, žiadne zariadenie nie je dokonalé a nedefinované pixely môžu vzniknuť aj chybou skenera pri vypočítavaní hĺbkovej mapy. Tiene a lesk na scéne sú však problémové oblasti, ktoré bývajú v hĺbkových mapách spravidla nedefinované. Nedefinované pixely majú v hĺbkovej mape hodnotu 0. Na Obr. 1.1 sú v hĺbkovej mape nedefinované pixely čierne.

Intenzitná textúra je obyčajná fotka scény osvietenej ledkou. Všetky pixely tejto 2D matice sú teda definované. V tomto zjavne intenzitná textúra ponúka oproti hĺbkovej mape o skenovanej scéne informácie navyše.

1.2.2 Hrany skenovanej scény

Zaujímavými časťami na porovnávanie vstupných obrazov sú hrany objektov skenovanej scény. Rozlíšiteľnosť a ostrosť hrán v nejakom obraze vo veľkej miere vypovedá o kvalite daného obrazu. Hrany objektov v intenzitnej textúre pozične s vysokou presnosťou zodpovedajú hranám z hĺbkovej mapy lebo boli obidva obrazy vytvorené pomocou tej istej kamery. Umiestnenie kamery a scéna samotná sa medzi vytváraním hĺbkovej mapy a intenzitnej textúry nemení. Pozície hrán by teda mali byť vo vstupných obrazoch na rovnakých pozíciach so zanedbateľnou nepresnosťou. Pri skúmaní dát sme odpozorovali, že hrany sú v intenzitných textúrach vo všeobecnosti veľmi dobre viditeľné. Vo veľkej miere tomu pomohlo osvietenie ledkou pri fotení scény. Celkovo platí fakt, že čím je scéna, ktorú kamera fotí viac osvetlená, tým lepšie na fotke vidieť hrany.

Keď sme skúmali okolia hrán intenzitných textútúr, narazili sme na jednu ich zlú vlastnosť. Ako sme uviedli vyššie, hrany sú v intenzitných textúrach dobre rozlíšiteľné. Problém však nájdeme na ich veľmi blízkom okolí. Na lokálnom okolí sú hrany v intenzitných textúrach rozmazané. Nie sú tak ostré ako v hĺbkových mapách. Na základe našich pozorovaní sú všetky hrany v intenzitných textúrach oproti tým v hĺbkových mapách na malých okoliach rozmazané. Otázkou je, čím je to spôsobené.

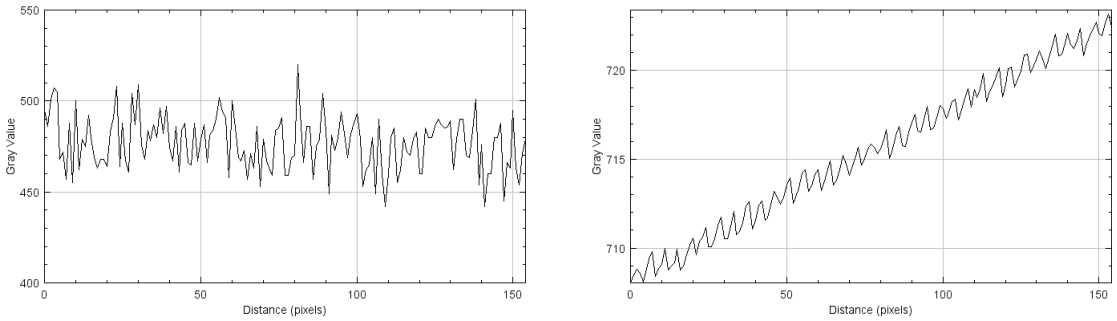


Obr. 1.2: Časť intenzitnej textúry (vľavo) a hĺbkovej mapy (vpravo), ktorá obsahuje hrany objektov scény.

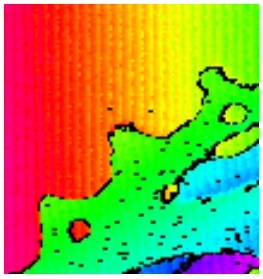
Niektoré rozmazania hrán môžu byť spôsobené svetelnými faktormi pri fotení scény, ako napr. lesk alebo tieň. Domnievame sa však, že príčinou toho, že sú hrany v intenzitných textúrach rozmazané je kvalita fotky z kamery, chyba spôsobená zaostrením alebo iné podobné faktory, ako napr. difúzny povrch objektov v scéne. HĺbkovÁ mapa sa vypočítava z nasvietených projekcií, a preto ak je hrana objektu v scéne ostrá, tak ju vypočíta presne ostrú. Nanajvýš sa môže stať, že niektoré pixely určí ako nedefinované. Aby sme ukázali, ako toto rozmazanie hrán vyzerá, zobrazili sme pomocou softvéru ImageJ [14] na Obr. 1.2 tie isté vyseknuté časti z intenzitnej textúry a hĺbkovej mapy. Zobrazené sú v 3D grafoch, kde X-ová a Y-ová os určujú súradnice pixelov v obraze a Z-ová os predstavuje hodnoty daných pixelov. Na Obr. 1.2 vidíme, že hrany intenzitnej textúry sú oproti hranám v hĺbkovej mape výrazne rozmazané. Jedná sa síce o malé okolie hrany, ale je to zjavná nepresnosť intenzitných textúr oproti hĺbkovým mapám.

1.2.3 Plochy skenovanej scény

Okrem hrán skenovanej scény nás zaujíma aj to, v akom vzťahu sú ploché povrchy neobsahujúce hrany v hĺbkovej mape voči plochým povrchom v intenzitnej textúre. Očakávali sme, že povrchy objektov budú nanajvýš plné náhodného šumu v podobe minimálnych neštruktúrovaných odchýliek od správnych hodnôt pixelov. V intenzitných textúrach sa toto očakávanie splnilo. Plocha v skenovanej scéne sa v intenzitnej textúre vyskytuje ako plocha s malým šumom. To sa však nedá povedať o hĺbkových mapách. HĺbkovÉ mapy obsahujú na plochách v scéne chyby spôsobené jeho nepresnou kalibráciou. Pri nepresne nakalibrovanom skeneri vzniknú pri rekonštrukcii povrchu objektov z nasvietenej laserovej projekcie neexistujúce nevýrazné hrany. Plochy v hĺbkových mapách nie sú hladké ale vrúbkované. Povrch plôch je vrúbkovaný kvôli tvaru projekcií, ktoré laser pri vytváraní hĺbkovej mapy na scénu nasvieti. Pre demonštráciu uvedených šumov porovnáme konkrétne časti vstupných obrazov. Softvér ImageJ [14] ponúka



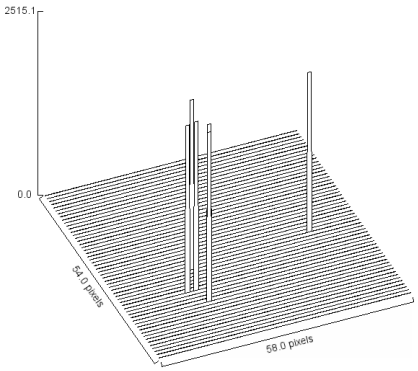
Obr. 1.3: Úsečka pixelov intenzitnej textúry (vľavo) a hĺbkovej mapy (vpravo) prechádzajúca plochou v skenovanej scéne.



Obr. 1.4: Zobrazenie časti hĺbkovej mapy, kde vidieť na skenovanom povrchu vrúbky spôsobené nepresnou kalibráciou 3D skenera.

možnosť vyznačenia úsečky pixelov z daného vstupného obrazu. Z hodnôt pixelov tejto úsečky vie tento softvér vytvoriť graf, kde X-ovú os tvoria pixely úsečky a Y-ovú os tvoria hodnoty daných pixelov. Vyznačili sme teda z hĺbkovej mapy a intenzitnej textúry tú istú úsečku pixelov, ktorá prechádza cez plochu v skenovanej scéne. Grafy tejto úsečky pixelov uvádzame na Obr. 1.3. Okrem grafov vybraných úsečiek pixelov sme zobrazili na Obr. 1.4 malú časť hĺbkovej mapy, na ktorej môžeme vrúbky pozorovať vďaka lineárnemu namapovaniu hodnôt pixelov na farebné spektrum.

Na Obr. 1.3 vidíme, že plocha v intenzitnej textúre obsahuje náhodný šum. Plocha v hĺbkovej mape obsahuje štruktúrovanú, nenáhodnú chybnú informáciu spôsobenú nasvietením projekcie lasera. Keďže je vyznačená úsečka pixelov v obraze na rovnej ploche, v hĺbkovej mape vzdialenosť jednotlivých pixelov od kamery narastá, v grafe teda vidíme rastúcu funkciu hodnôt pixelov. Intenzitná textúra je len fotka, čiže hodnoty pixelov úsečky sa pohybujú okolo jednej stálej hodnoty, ktorá určuje intenzitu šedej farby danej plochy na fotke. To, čo sa dá vyťažiť z tohto pozorovania je, že chybné miniatúrne hrany v podobe vrúbok, ktoré hĺbková mapa obsahuje v intenzitnej textúre nenájdeme. Medzi šumom v intenzitnej textúre a šumom v hĺbkovej mape sme pri ich analýze nenašli žiadnu koreláciu.



Obr. 1.5: Chybné pixely v nedefinovanej oblasti hĺbkovej mapy.

1.2.4 Šum vo vstupných obrazoch

Pri analýze hrán a plôch vstupných obrazov sme ukázali niektoré chyby vstupných obrazov. Okrem týchto nájdeme aj iné typy šumu, ktoré treba odstrániť. Opíšeme niektoré z nich, ktoré sme v dátach odpozorovali. Zariadenie, ktoré na skenovanie používame, nie je dokonalé. Príčiny vzniku šumu sú rôzne. Môžu byť spôsobené numerickou chybou vo výpočte hĺbkovej mapy alebo skenovaním objektov, ktoré majú lesklý povrch. Odrazené svetlo v scéne môže spôsobiť vznik v reálnej scéne neexistujúcich bodov. Príkladom takéhoto šumu sú osamotené pixely umiestnené mimo všetkých skenovaných objektov scény. Príklad takéhoto šumu môžeme vidieť v 3D grafe hodnôt pixelov nedefinovanej oblasti scény na Obr. 1.5. Vidíme, že tieto pixely nereprezentujú žiadny reálny objekt, a teda by mali byť nedefinované.

Ďalšou veľkou chybou dát, ktorú sme odpozorovali, je lesk na povrchu objektov v intenzitných textúrach. Lesk spôsobuje napríklad to, že na tmavom objekte, ktorý má lesklý povrch vzniknú biele škvrny. Ak scéna obsahuje predmety s lesklým povrchom, je pravdepodobnosť vzniku takýchto chybných oblastí veľmi vysoká, lebo pri vytváraní intenzitnej textúry scénu osvietime ledkou. Tým síce získame celkovo lepšiu rozlíšiteľnosť hrán, ale nevyhneme sa chybným, lesklým oblastiam obrazu.

Celkovo sa dá predpokladať, že problémy spôsobené lesklými predmetmi alebo tieňami v scéne sa prejavia v hĺbkovej mape vznikom nedefinovaných pixelov a v intenzitnej textúre chybnými svetlejšími oblasťami obrazu. Ak je nejaký pixel v hĺbkovej mape nedefinovaný, mohol to spôsobiť lesk na povrchu predmetu. Ak to spôsobil lesk, tak na tom istom mieste v intenzitnej textúre nájdeme chybnú svetlú oblasť. Problém nastáva vtedy, keď je oblasť ovplyvnená leskom v intenzitnej textúre väčšia ako nedefinovaná oblasť v hĺbkovej mape. Ak je pixel v hĺbkovej mape definovaný a jemu pozične zodpovedajúci pixel v intenzitnej textúre je ovplyvnený leskom alebo tieňom v scéne, môže to výrazne poškodiť vzťah medzi týmito obrazmi. Lesk a tiene spravidla vznikajú na hranách objektov s lesklým povrchom. To znamená, že okolie hrán v intenzitnej textúre je často výrazne zašumené.

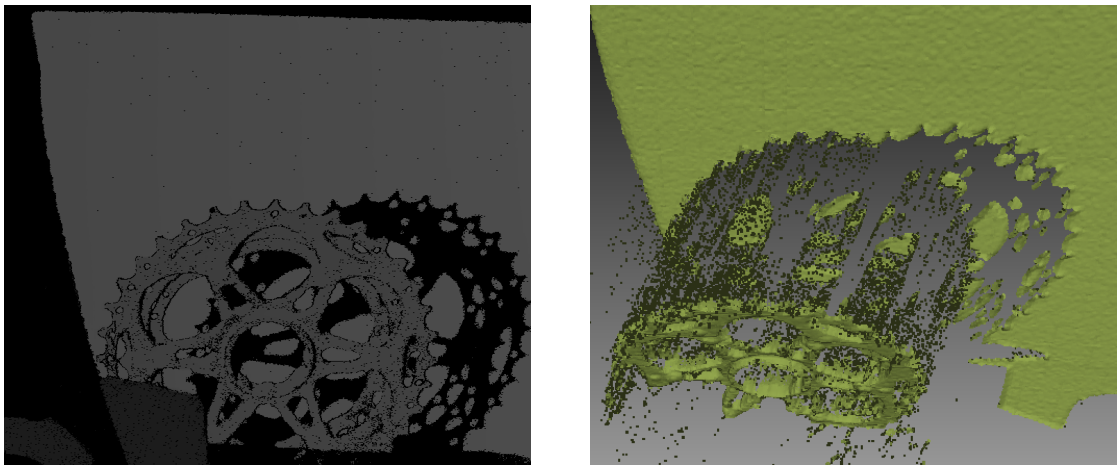
Kapitola 2

Filtrovanie mračien bodov

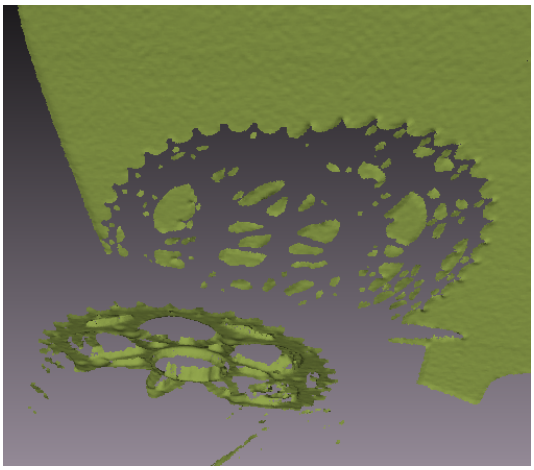
Pri spracovaní obrazu aplikácia filtrov často rieši problém zvýraznenia dôležitých detailov obsahu obrazu alebo naopak potlačenie nechcených vlastností, ktoré obraz má. Pri skenovaní 3D skener naskenuje povrch objektov scény s istou presnosťou. V naskenovanej scéne nie je povrch 3D objektov ideálny, je zašumený. Aplikáciou vybraného filtra sa pokúsime vyhladiť zašumený povrch objektov v 3D scéne a pritom zachovať ich hrany, aby ostali ostré a výsledný obraz bol v konečnom dôsledku kvalitnejší ako vstupný. V tejto kapitole najskôr opíšeme charakter dát, ktoré budeme filtrovať. Potom predstavíme vybrané filtrovacie techniky, ktoré slúžia na vyhladenie plôch so zachovaním hrán v obrazoch. Nakoniec vyberieme jednu, ktorú budeme implementovať.

2.1 Charakter filtrovaných dát

Obraz, ktorý očakávame na výstupe z filtra je hĺbková mapa, kde objekty umiestnené v scéne majú vyhladený povrch a ostré hrany. V konečnom dôsledku teda očakávame kvalitnejšiu hĺbkovú mapu ako na vstupe. Hĺbkové mapy sú iný typ dát ako klasické 2D obrázky, na ktorých je väčšina filtrov definovaných. Pri tomto type dát sa od nás vyžaduje väčšia presnosť. Ak napríklad na klasickej čiernobielej fotke veľmi jemne rozmažeme niektoré hrany, môže sa stať, že to v konečnom dôsledku v obraze nie je ani vidieť. Záleži na tom, čo od výstupného obrazu filtrovania očakávame. Hĺbková mapa je štruktúrované mračno bodov. Môžeme ju teda zobraziť ako body v 3D priestore. Ak by sme pri filtrovaní hoci len veľmi jemne rozmazali nejakú hranu v scéne, v mračne bodov by sme mohli nájsť chybné osamotené body na hranách v priestore, ktoré reálne neexistujú. Pre ilustráciu sme zobrazili na Obr. 2.1 časť hĺbkovej mapy zobrazenú ako čiernobiely 2D obraz (čím je pixel vzdialenejší od kamery, tým má svetlejšiu farbu, čierne pixely sú nedefinované) a ako mračno bodov v 3D priestore. Mračná bodov v tejto práci zobrazujeme pomocou softvéru Meshlab [13]. Keď sa pozrieme na Obr. 2.1 povedali by sme, že hrany prevodového kolesa, ktoré sa v scéne nachádza v hĺbkovej



Obr. 2.1: Hĺbková mapa zobrazená ako čiernobiely 2D obraz (vľavo) a mračno bodov (vpravo) v 3D priestore.



Obr. 2.2: Mračno bodov s ostrými hranami.

mape nie sú rozmazané. Ak však z tejto hĺbkovej mapy vypočítame mračno bodov, rozmazanie na malom okolí hrán sa prejaví tak, že medzi dvoma disjunktnými objektami scény, prevodovým kolesom a stenou v pozadí, vzniknú body, ktoré v realite neexistujú. Pre porovnanie uvádzame na Obr. 2.2 mračno bodov vypočítané z hĺbkovej mapy, ktorej hrany sú ostré. Vidíme, že toto mračno bodov neobsahuje chybné pixely medzi disjunktnými objektami scény. Našim cieľom je teda pri filtrovaní hĺbkovej mapy čo najviac eliminovať rozmazanie na hranách objektov, teda vznik takýchto chybných pixelov. Celkovo teda pri výslednom obraze potrebujeme vysokú presnosť. Dáta, na ktoré sa pokúsime aplikovať vybranú filtrovaciu techinku sú veľmi citlivé aj na malé nesprávne odchýlky hodnôt pixelov. Špeciálne nesmieme rozmazať hrany, ktoré určujú hranice medzi disjunktnými objektami scény. V realite medzi nimi môže byť dosť veľká vzdialenosť. Pri jemnom rozmazaní nám môžu vzniknúť chybné body, ktoré disjunktné objekty spájajú do jedného.

Ďalším obmedzením, ktoré prichádza pri štruktúrovaných mračnách bodov je to, že ak potrebujeme nejaký pixel z obrazu pri filtrovaní odstrániť, musíme ho nastaviť ako nedefinovaný. To znamená, že mu treba v hĺbkovej mape nastaviť hodnotu 0. Pri 3D zobrazení mračna bodov sa potom daný pixel v nezobrazí. Pri 2D čiernobielom obraze často na odstránenie pixela, ktorý má zlú hodnotu stačí jednoducho hodnotu pixela stiahnuť blízko k hodnote 0, kde už nie je rozdiel oproti nulovým pixelom rozoznateľný. Pri hĺbkových mapách sú všetky pixely, ktoré majú kladnú hodnotu, umiestnené v 3D scéne. To od nás vyžaduje jasné určenie definovaných a nedefinovaných pixelov.

Výsledná hĺbková mapa má byť vstupom pre proces segmentácie, v ktorom sa jednotlivé objekty scény stanú samostatnými segmentami. Ak by sme niektoré objekty pospájali pridaním neexistujúcich hrán, tak by sme tým procesu segmentácie dali zlé vstupné dáta a osegmentovanie objektov by mohlo byť v konečnom dôsledku chybné.

2.2 Filtrovacie techniky

Na vstupné hĺbkové mapy chceme aplikovať taký filter, ktorý vyhladí povrch objektov scény, a pritom zachová ich hrany. Filtre takéhoto typu sa volajú hrany-zachovávajúce filtre. Jedným z najznámejších a najpoužívanejších hrany-zachovávajúcich filtrov je bilaterálny filter [16]. Tento filter používa na vyhladenie povrchu objektov princíp Gaussovho filtra. Pre každý pixel vypočíta vážený priemer z hodnôt pixelov jeho okolia. Čím je nejaký pixel z okolia daného pixela k nemu bližšie, tým väčšiu váhu v priemere jeho okolia má. Aby takýmto spôsobom bilaterálny filter nerozmazal aj hrany objektov, vážený priemer ovplyvňuje okrem vzdialenosti pixelov aj ich hodnotový rozdiel. Hodnotový rozdiel pixelov signalizuje v obraze hranu. Pri bilaterálnom filtri platí, že čím viac sa nejaký pixel okolia daného pixela od neho hodnotovo líši, tým menšiu váhu v jeho váženom priemere okolia bude mať. Hodnotu pixela p výstupného obrazu z bilaterálneho filtra $I^{(BF)}(p)$ môžeme vyjadriť nasledujúcim vzťahom [16]:

$$I^{(BF)}(p) = \frac{1}{W_p} \sum_{q \in \Omega} I^{(vstup)}(q) \exp\left(-\frac{||p-q||^2}{\sigma_s^2}\right) \exp\left(-\frac{|I^{(vstup)}(p) - I^{(vstup)}(q)|^2}{\sigma_r^2}\right), \tag{2.1}$$

kde W_p je normalizačný faktor, ktorý zaistí, že súčet váh hodnôt pixelov bude 1, q je pixel z okolia Ω pixela p a $I^{(vstup)}(p)$ je hodnota pixela p vo vstupnom obraze. Nastaviteľné parametre σ_s a σ_r určujú, ako vzdialené pixely a pixely s akou maximálnou odlišnou hodnotou ovplyvnia vážený priemer hodnôt vypočítaný pre daný pixel.

Jednoducho povedané, hodnotu výsledného pixela pri bilaterálnom filtrovaní ovplyvnia pixely, ktoré sú blízko neho a nemajú výrazne inú hodnotu. Priemerom sa vyhladzujú len časti obrazu, kde je nízka variancia hodnôt pixelov, čiže plochy neobsahujúce hrany. Hrany v podobe hodnotových skokov v obraze sa priemerom nevyhladia. Bilaterálny

filter pre každý pixel vyžaduje vypočítanie váhy každého pixela z jeho okolia s polomerom r. Okolie pixela p definujeme ako štvorcové okno z obrazu so stranou dĺžky $(2r+1)$ pixelov, vycentrované na pixel p. Pri n vstupných pixeloch môžeme teda časovú zložitosť bilaterálneho filtra odhadnúť na $O(n \cdot (2r+1)^2)$. S rastúcim okolím pixela, ktoré berieme pri filtrovaní do úvahy rastie aj čas filtrovania.

Bilaterálny filter má vo všeobecnosti veľmi dobré výsledky, ale sú situácie, kedy zlyháva. Jednou z jeho chýb je, že jeho parametre σ_s a σ_r sú nastaviteľné len globálne pre celý obraz. Ak je v nejakej oblasti obrazu šikmá plocha, ktorej pixely majú vysoký gradient, nastane situácia, kedy majú aj veľmi blízke pixely okolia daného pixela, z ktorého počítame vážený priemer, od neho výrazne odlišné hodnoty. Ak je parameter σ_r nízky, tak bude výsledný vážený priemer každého pixela tejto plochy menej ovplyvnený hodnotami pixelov z jeho blízkeho okolia, lebo sú od neho príliš odlišné. Takáto šikmá plocha v obraze sa dôsledkom toho nevyhladí tak dobre, lebo všetky pixely z okolia daného pixela ho pomocu váženého priemeru upravia len s nízkou váhou. Keď si to prenesieme do terminológie nášho vstupného 3D obrazu, tak hovoríme, že bilaterálny filter na hĺbkových mapách scény zlyháva v oblastiach, kde je uhol normály povrchu nejakého objektu od kamery skenera veľmi veľký. Takéto povrchy nevie dobre vyhladiť.

Tento problém bilaterálneho filtra je viac popísaný v publikácii, kde bol predstavený novší trilaterálny filter [4]. Trilaterálny filter eliminuje nedostatky bilaterálneho filtra takým spôsobom, že bilaterálny filter používa so samostatnými parametrami na oblastiach s rovnakým gradientom hodnôt pixelov. Principiálne robí to, že pre každý pixel funkciu bilaterálneho filtra otáča a škáluje podľa oblastí obrazu s podobným gradientom hodnôt pixelov. Tento filter eliminuje veľa nedostatkov bilaterálneho filtra vrátane problému vyhladenia oblastí s vysokým gradientom, ale je výpočtovo zložitejší. Chceli by sme nájsť rýchlu, robustnú filtrovaciu techniku založenú na inom matematickom modeli, ktorá by bola výpočtovo jednoduchšia ako trilaterálny filter, a pritom by zvládla vyhladiť povrch, ktorého uhol normály od kamery je veľký.

V Kap. 1 sme uviedli, že 3D skener nám okrem hĺbkovej mapy skenovanej scény dáva k dispozícii intenzitnú textúru, čo je čiernobiela fotka skenovanej scény. Máme teda pri filtrovaní možnosť použiť okrem samotného vstupného aj druhý, pomocný obraz. Bilaterálny ani trilaterálny filter z definície nevyužívajú pomocný obraz, máme teda oproti nim informáciu o vstupnom obraze navyše, ktorú môžeme skúsiť využiť. Hlavnou motiváciou témy tejto práce bola práve myšlienka využitia pomocného obrazu v podobe intenzitnej textúry pri filtrovaní hĺbkových máp. Chceme zistiť, či použitím nejakej rýchlej, robustnej filtrovacej techniky, ktorá by využívala intenzitnú textúru vieme dosiahnuť lepšie výsledky tam, kde má bilaterálny filter problémy.

Populárnym filtrovacím pristupom, ktorý využíva okrem vstupného obrazu filtrovania aj pomocný obraz, je guided filter [8]. Tento filter na vyhladzovanie plôch obrazu používa aritmetický priemer. Oproti bilaterálnemu filtru, kde je priemer pixelov okolia

vážený na základe pozícií pixelov a rozdielov ich hodnôt, má pri guided filtri každý pixel okolia daného pixela rovnakú váhu. Na hranách v obraze sa výstupné pixely počítajú pomocou lokálnej lineárnej transformácie pomocného obrazu. Cez pixely okolia daného pixela sa v pomocnom obraze využitím lineárnej regresie prekladá priamka, ktorá určí výslednú hodnotu daného pixela. Tento matematický model by nemal na oblastiach s vysokým gradientom zlyhať, lebo výslednú hodnotu pixela ovplyvňujú všetky pixely z jeho okolia v pomocnom obraze. Podľa publikácie [8], kde bol guided filter predstavený, by mal mať pri n vstupných pixeloch obrazu časovú zložitosť $O(n)$, nezávislú na veľkosti okolia pixela, ktoré sa pri filtrovaní berie do úvahy. Je tu teda možnosť, že guided filter je filtrovacia technika, ktorá by mohla v hĺbkových mapách na povrchoch s veľkým uhlom normály od kamery skenera nezlyhať, a pritom fungovať dokonca v lepšom čase ako bilaterálny filter. Rozhodli sme sa preto v tejto práci implementovať guided filter [8] a overiť, či sa naše odhady s vstupnými dátami z PhoXi 3D skenera [12] naplnia. V nasledujúcej kapitole podrobnejšie opíšeme guided filter a analyzujeme možnosti jeho aplikácie na naše vstupné dáta.

Kapitola 3

Guided filter

Základnou myšlienkou eliminácie šumu z obrazu pri guided filtri je pre každý pixel urobiť aritmetický priemer hodnôt pixelov z jeho okolia. Pixely, ktoré tvoria šum a majú byť odstránené, sa svojou hodnotou výrazne odlišujú od hodnôt pixelov v ich okolí. Keď ich hodnotu zmeníme na aritmetický priemer hodnôt pixelov z ich okolí, ich odlišnosť eliminujeme. Čím väčšie okolie pixela zvolíme, tým výraznejší šum bude možné eliminovať. Zväčšovaním uvažovaného okolia však znižujeme kvalitu obrazu. Ak by sme priemerovali aj pixely, ktoré sú na hranách skenovaných objektov, hrany by sa vyhladili a objekty by boli horšie rozoznateľné. Problém, ktorý guided filter rieši je odlíšenie častí obrazu, ktoré sa majú priemerovať a ktoré nie. Na určenie oblastí, kde sa obraz nemá priemerovať, používa guided filter takzvaný guidance obraz. Guidance obraz je pozične identický so vstupným obrazom. Má na rovnakých miestach hrany, ktoré sú dobre rozoznateľné. Jednoduchým príkladom pre predstavu je fotka s bleskom použitá ako guidance obraz k fotke bez blesku. Na fotke s bleskom je dobre vidieť hrany objektov. To nám môže slúžiť ako nápomocná informácia o tom, kde fotku priemerovať aby sme eliminovali šum a kde zachovať hrany. V tejto kapitole zadefinujeme guided filter [8] a analyzujeme spôsob jeho aplikácie na naše vstupné dáta. Pri jeho opise budeme vychádzať z publikácie [8], v ktorej bol predstavený.

3.1 Model

Pri guided filtri máme vstupný obraz p, na ktorý chceme filter aplikovať. Okrem vstupného obrazu máme aj guidance obraz I, ktorý slúži ako vedenie pri zachovávaní hrán. Hrany sú spravidla v I dobre viditeľné. Aplikovaním filtra dostaneme výsledný obraz q. V obraze p vieme pre pixel k určiť štvorcové okno ω_k, ktoré má stred v pixeli k. Strana okna ω_k má teda dĺžku $2r + 1$, kde r definujeme ako polomer okolia pixela. Okná, ktoré majú stredy blízko hraníc obrazu nebudú celé v obraze. V takom prípade nebudeme pri aplikácii filtra brať ich vyčnievajúce časti do úvahy. Okno ω_k sa v odbornej

literatúre [8] nazýva aj *kernel* pixela k.

Pri opise základnej myšlienky guided filtra sme hovorili o tom, že pri vytváraní výstupného obrazu q sa budeme riadiť guidance obrazom I. Medzi týmito dvoma obrazmi teda chceme zadefinovať nejaký vzťah. Pri guided filtri je tento vzťah medzi I a q definovaný ako lokálna lineárna závislosť, ktorá je pre daný pixel k obrazu q v lokálnom okolí ω_k vyjadrená nasledovne [8]:

$$q_i = a_k I_i + b_k, \forall i \in \omega_k, \tag{3.1}$$

kde (a_k, b_k) sú lineárne koeficienty konštantné pre ω_k. Hľadané koeficienty potrebujeme nejakým spôsobom ohraničiť. Faktom, ktorý určuje isté ohraničenie koeficientov je to, že výsledný obraz q sa má v konečnom dôsledku vo veľkej miere podobať na vstupný obraz p. Preto chceme zadefinovať vzťah medzi p a q. Na vyjadrenie vzťahu medzi p a q stačí jednoduchá úvaha. Výstupný obraz q vieme vyjadriť ako vstupný obraz p, od ktorého odčítame nechcený šum. Táto úvaha je podstatou aplikácie ľubovoľného filtra. Tento vzťah medzi p, q a nechceným šumom n vieme pre pixel $i \in \omega_k$ vyjadriť nasledovne [8]:

$$q_i = p_i - n_i. \tag{3.2}$$

Pre dané okno ω_k budeme hľadať také (a_k, b_k), ktoré minimalizuje rozdiel medzi p a q a pritom sa zachová lineárny model (3.1). Budeme teda minimalizovať súčet kvadratických odchýliek vstupného a výstupného obrazu [8]:

$$E(a_k, b_k) = \sum_{i \in \omega_k} ((a_k I_i + b_k - p_i)^2 + \epsilon a_k^2). \tag{3.3}$$

Parameter ϵ je parametrom regularizácie, ktorý znevýhodňuje veľké hodnoty a_k.

Pre minimalizačný problém (3.3) vieme odvodiť nasledovné výsledky [8]:

$$a_k = \frac{\frac{1}{|\omega|} \sum_{i \in \omega_k} I_i p_i - \mu_k \overline{p}_k}{\sigma_k^2 + \epsilon}, \tag{3.4}$$

$$b_k = \overline{p}_k - a_k \mu_k. \tag{3.5}$$

V týchto vzťahoch μ_k a σ_k^2 predstavujú priemer I v ω_k resp. varianciu I v ω_k. Potom teda vieme vzťahy (3.4) a (3.5) prepísať takto:

$$a_k = \frac{\overline{I_k p_k} - \overline{I}_k \overline{p}_k}{Var(I_k) + \epsilon}, \tag{3.6}$$

$$b_k = \overline{p}_k - a_k \overline{I}_k. \tag{3.7}$$

Z definície variancie si vyjadríme $Var(I_k)$:

$$Var(I_k) = \overline{I_k^2} - \overline{I_k}^2. \tag{3.8}$$

Z definície kovariancie vieme, že platí:

$$Cov(I_k, p_k) = \overline{I_k p_k} - \overline{I}_k \overline{p}_k. \tag{3.9}$$

Použitím rovnosti (3.9) bude vzťah (3.6) možné upraviť nasledovne:

$$a_k = \frac{Cov(I_k p_k)}{Var(I_k) + \epsilon}. \tag{3.10}$$

Vyššie odvodené vzťahy neskôr použijeme pri implementácii filtra. Keď sme odvodili koeficienty a_k a b_k pre okno ω_k vieme pre pixel $i \in \omega_k$ vypočítať výsledné q_i. Môžeme si však všimnúť, že pixel i patrí do viacerých okien, v ktorých sa jeho výsledná hodnota q_i líši, lebo okná môžu mať rôzne koeficienty (a_k, b_k). Tento problém vyriešime tak, že pre daný pixel i spriemerujeme jeho výsledné q_i zo všetkých okien, ktoré obsahujú i. Keď vypočítame koeficienty pre všetky okná, môžeme vypočítať priemerné q_i [8]:

$$q_i = \frac{1}{|\omega|} \sum_{k|i \in \omega_k} (a_k I_i + b_k). \tag{3.11}$$

Podľa publikácie o guided [8] filtri vieme z tejto rovnice odvodiť nasledovné:

$$q_i = \overline{a}_i I_i + \overline{b}_i, \tag{3.12}$$

kde $\overline{a}_i$ a $\overline{b}_i$ sú priemernými koeficientmi všetkých okien, do ktorých patrí pixel i. Aby sme to zhrnuli, definíciu guided filtra tvoria rovnice (3.4), (3.5) a (3.11). Rovnosti (3.8), (3.9) a (3.12), ktoré sme z definície guided filtra odvodili, využijeme pri jeho implementácii.

3.2 Algoritmus

Opíšeme algoritmus guided filtra [8], ktorý budeme neskôr implementovať. Použijeme pri tom definíciu guided filtra a vzťahy, ktoré sme z nej odvodili. Vstupnými parametrami algoritmu sú: vstupný obraz p, guidance obraz I, polomer kernela r a regularizačná konštanta ϵ. Výstupom algoritmu má byť filtrovaný obraz q. V algoritme využijeme odvodené rovnice (3.7) - (3.10) a (3.12). V týchto rovniciach počítame s priemermi obrazov vychádzajúcich z p a I. Pri opise algoritmu budeme takýto priemer obrazov označovať ako funkciu $f_{priemer}$. Pre daný obraz X je teda funkcia $f_{priemer}(X)$ definovaná ako filter, ktorý pre každý pixel obrazu X vypočíta priemernú hodnotu z hodnôt pixelov z jeho kernela so zadaným polomerom r. Keď máme zadefinovanú funkciu $f_{priemer}$ a odvodené potrebné vzťahy popíšeme po jednotlivých krokoch algoritmus guided filtra tak, ako je uvedený v publikácii [8].

Algoritmus 1 Guided filter

Vstup: vstupný obraz p, guidance obraz I, polomer kernela r, regularizácia ϵ

Výstup: výstupný obraz q

1: $\text{priemer}_I = f_{priemer}(I)$
$\text{priemer}_p = f_{priemer}(p)$
$\text{priemer}_{II} = f_{priemer}(I\ .*\ I)$
$\text{priemer}_{Ip} = f_{priemer}(I\ .*\ p)$
2: $\text{var}_I = \text{priemer}_{II} - \text{priemer}_I\ .*\ \text{priemer}_I$
$\text{cov}_{Ip} = \text{priemer}_{Ip} - \text{priemer}_I\ .*\ \text{priemer}_p$
3: $a = \text{cov}_{Ip}\ ./\ (\text{var}_I + \epsilon)$
$b = \text{priemer}_p - a\ .*\ \text{priemer}_I$
4: $\text{priemer}_a = f_{priemer}(a)$
$\text{priemer}_b = f_{priemer}(b)$
5: $q = \text{priemer}_a\ .*\ I + \text{priemer}_b$

Pod operáciami $.*$ a $./$ používanými v algoritme myslíme operácie násobenia a delenia po jednotlivých prvkoch matíc. V prvom kroku algoritmu získame pomocou funkcie $f_{priemer}$ spriemerované obrazy, ktoré budeme v ďalších krokoch využívať. V druhom kroku na základe rovníc (3.8) a (3.9) vypočítame varianciu pre každý kernel obrazu I a kovarianciu medzi k sebe prislúchajúcimi kernelmi obrazov p a I. Získané var_I a cov_{Ip} využijeme pri počítaní koeficientu a v treťom kroku algoritmu, kde sa budeme opierať o vzťahy (3.10) a (3.7), odvodzujúce koeficienty a a b. V štvrtom kroku algoritmu vypočítame pre každý pixel priemerné koeficienty a a b. V piatom kroku vypočítame konečný výstupný obraz q pomocou koeficientov získaných v predošlých krokoch algoritmu a rovnice (3.12) odvodenej z definície guided filtra.

To, čím je guided filter známy a dôvodom jeho veľkého využitia pri spracovaní obrazu je práve jeho časová zložitosť. V algoritme guided filtra (Algoritmus 1) si môžeme všimnúť, že jeho časová zložitosť závisí od implementácie funkcie $f_{priemer}$. Táto funkcia je definovaná ako funkcia, ktorá priemer daného obrazu počíta v lineárnom čase [8]. Existuje veľa rôznych implementácií tejto funkcie, ktoré pracujú v lineárnom čase. V algoritme guided filtra sa funkcia $f_{priemer}$ pre vstupný obraz s n pixelami vykoná len konštantne veľa krát. Z toho vyplýva, že celková časová zložitosť algoritmu guided filtra je $O(n)$ [8].

3.3 Predpoklady guided filtra

Výstupným obrazom q je pri guided filtri lokálna lineárna transformácia guidance obrazu I, pričom sa minimalizuje odchýlka vstupného obrazu p od výstupného obrazu q. Jednoduchým popisom základnej myšlienky modelu guided filtra je, že čím je v nejakej

lokálnej časti obrazu korelácia medzi obrazmi p a I vyššia, tým viac tam bude výstupný obraz q podobný obrazu I. Naopak, čím bude korelácia medzi p a I nižšia, tým viac bude výstupný obraz q v danej lokálnej oblasti spriemerovaný obraz p.

To, že je korelácia na častiach vstupných obrazov vysoká znamená, že sa v týchto častiach obrazu hodnoty pixelov vstupných obrazov menia podobným spôsobom. Fakt, že v nejakej časti obrazu je korelácia medzi vstupnými obrazmi nízka nám zase hovorí o tom, že ide o oblasť, kde medzi zmenami hodnôt pixelov vstupných obrazov nie je žiadny vzťah. Ak je medzi celkovou zmenou hodnôt pixelov v nejakej malej oblasti obrazu p a obrazu I nejaký vzťah, je medzi nimi nenulová korelácia. Hrany v p a I sú oblasti, v ktorých by mala byť vysoká korelácia obrazov, lebo sa tam podobným spôsobom menia hodnoty pixelov. Plochy obrazov p a I obsahujú rôzny náhodný šum, ktorý spôsobí, že je v týchto oblastiach korelácia obrazov blízka nule.

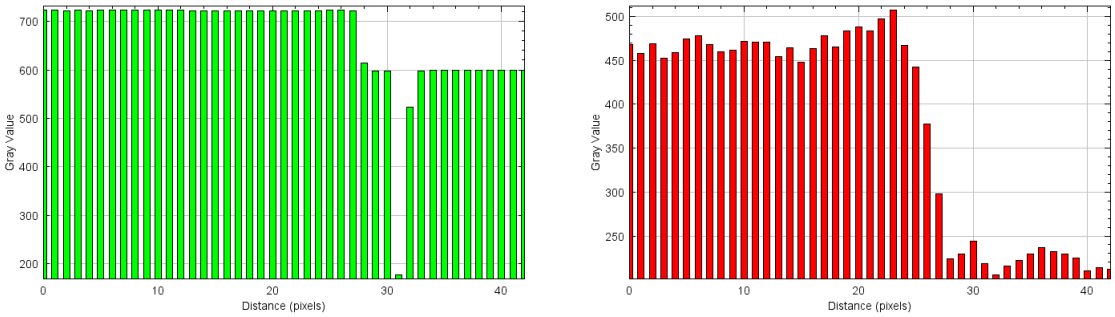
Predpokladom guided filtra pre vstupné obrazy je, že na plochách v obraze, ktoré chceme vyhladiť, je medzi obrazmi p a I nízka korelácia, lebo obsahujú rôzny náhodný šum. Tým pádom sa hodnoty pixelov v týchto oblastiach na základe matematického modelu filtra priemerujú a plochy sa vyhladia. Naopak, na hranách v obraze potrebujeme, aby bola korelácia medzi obrazmi p a I vysoká, aby sa hodnoty pixelov v týchto častiach obrazu nepriemerovali, ale zmenili na vhodnú lineárnu transformáciu obrazu I.

Guided filter očakáva, že všetky hrany, ktoré sú v obraze p majú byť aj v obraze I. Ak by niektorú hranu obraz I neobsahoval, bola by na tom mieste medzi obrazmi nízka korelácia a hrana by sa vyhladila priemerovaním. Okrem toho je veľmi dôležité, aby boli v obraze I hrany ostré aspoň tak, ako vo vstupnom obraze, lebo v konečnom dôsledku to bude práve obraz I, na ktorý sa bude výsledný obraz q na hranách viac podobať.

3.4 Vstupné obrazy

Guided filter je v jeho publikácií [8] definovaný pre použitie na klasické 2D obrázky. Príkladom takýchto obrázkov môžu byť obyčajné 2D farebné alebo čiernobiele fotky. My ideme filter použiť na 3D dáta. Otázkou je, či sa vôbec guided filter dá na 3D dáta prepoužiť. Ak áno, potrebujeme zistiť akým spôsobom je to možné a čo treba vo vstupných dátach zabezpečiť, prípadne modifikovať.

Z definície očakáva guided filter na vstupe dva obrazy. Prvým je obraz p, ktorý má byť filtrovaný. Druhým vstupným obrazom je guidance obraz I, ktorý má slúžiť ako pomocná informácia pre filtrovanie obrazu p. V publikácií majú pri demonštrácií funkcionality guided filtra obidva vstpné obrazy podobný formát a charakter. Boli to napríklad fotka bez použitia blesku a fotka s bleskom. Obidva obrazy ponúkali veľmi podobný typ informácie o scéne. V tejto práci chceme aplikovať guided filter na hĺbkovú



Obr. 3.1: Úsečka pixelov z hĺbkovej mapy (vľavo) a intenzitnej textúry (vpravo), prechádzajúca hranou.

mapu, ktorá teda bude vstupným obrazom p a ako guidance obraz I chceme použiť intenzitnú textúru. Tieto dva obrazy nám ponúkajú o skenovanej scéne veľmi rozdielny typ informácií a nie je úplne zjavné, že filter pri nich bude fungovať správne. Pre overenie toho, či je na naše vstupné dáta možné guided filter aplikovať potrebujeme dokázať, že spĺňajú jeho základné predpoklady.

Pri analýze dát v Kap. 1 sme zistili, že všetky hrany, ktoré nájdeme v hĺbkovej mape sú aj v intenzitnej textúre. Horšie je to s ich kvalitou. Na blízkom okolí sú hrany v intenzitnej textúre oproti hĺbkovej mape rozmazané. Hrany sú tam rozmazané len na malom okolí a koreláciu medzi obrazmi to veľmi nepoškodí. Problém je v tom, že z definície guided filtra sa bude výsledný obraz v okolí hrán viac podobať intenzitnej textúre. Vo výstupnom obraze budú hrany jemne rozmazané, čo je oproti vstupnej hĺbkovej mape horšie. Po tomto zistení sme prišli k záveru, že na naše vstupné dáta nie je možné jednoduchým spôsobom aplikovať guided fiter. Jediná možnosť, ktorú máme je intenzitnú textúru nejakým spôsobom upraviť tak, aby mala dostatočne ostré hrany a aby si zároveň zachovala svoje pôvodné plochy, kde má nízku koreláciu s hĺbkovou mapou. Na Obr. 3.1 môžeme pozorovať, ako sa správajú hodnoty intenzitnej textúry oproti hodnotám hĺbkovej mapy v okolí hrán. Uvedené grafy znázorňujú priebeh hodnôt pixelov vo vstupných obrazoch, umiestnených na úsečke pixelov, ktorá v scéne pretína hranu. X-ová os reprezentuje indexy pixelov od začiatku úsečky a Y-ová os reprezentuje hodnoty daných pixelov. Pre demonštráciu vlastností obrazov a analýzu výsledkov nám bude množina pixelov z tejto úsečky testovacou vzorkou pri skúmaní rôznych obrazov aj neskôr v texte.

Na Obr. 3.1 vidíme, že priebeh hodnôt pixelov v hĺbkovej mape obsahuje zretelný, ostrý skok hodnôt. Samotná hrana tu má šírku 2 pixely. V priebehu hodnôt pixelov v intenzitnej textúre môžeme pozorovať rozmazanú hranu, ktorá má šírku 5 pixelov. Hodnoty pixelov tu namiesto ostrého skoku tvoria plynulý prechod hodnôt.

Guided filter na hranách používa lineárnu transformáciu guidance obrazu. Ak je tam guidance obraz rozmazaný, aj výsledný obraz bude na hranách rozmazaný. Potrebujeme

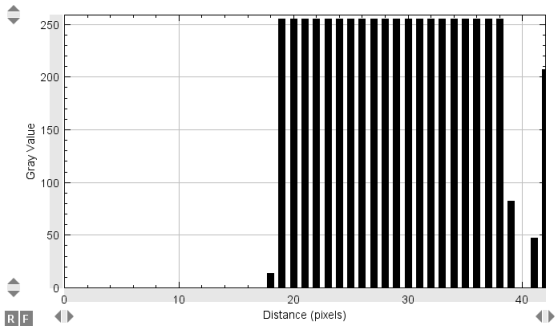
znížiť šírku hrany v intenzitnej textúre najviac, ako sa dá. Ideálne na takú šírku, akú má v hĺbkovej mape. Je nutné, aby sme pritom zachovali presné pozície všetkých hrán. V nasledujúcej podkapitole sa budeme venovať úprave intenzitnej textúry, teda nášho guidance obrazu. Budeme sa snažiť znížiť počet rôznych hodnôt pixelov v okolí hrán na dve dostatočne rozdielne hodnoty definujúce hranu. Zmena medzi týmito dvoma hodnotami bude musieť nastať presne na tom istom mieste, ako sa to deje v hĺbkovej mape.

3.5 Úprava intenzitnej textúry

Proces úpravy vstupnej intenzitnej textúry sa vykoná pred samotnou aplikáciou guided filtra, ktorý potom pracuje už s upravenou verziou vstupného obrazu. Upraviť intenzitnú textúru tak, aby mala ostré hrany a neovplyvnili by sme pritom pixely plôch nie je jednoduchá úloha. Vyskúšali sme viac prístupov k riešeniu tohto problému. V tejto podkapitole najskôr uvedieme základnú myšlienku, na ktorej je založený náš algoritmus úpravy. Následne opíšeme jednotlivé kroky algoritmu a nakoniec ukážeme a vyhodnotíme jeho výsledky.

3.5.1 Základná myšlienka

Keďže intenzitná textúra potrebuje zaostriť hrany, musíme niekde získať informáciu tom, kde sa tieto hrany nachádzajú. Informáciu o ostrých hranách na správnych miestach vieme nájsť v hĺbkovej mape scény, ktorá nám pri úprave môže slúžiť ako pomocný obraz. Hlavnou úvahou, na ktorej je postavený náš algoritmus je, že body v priestore scény, ktorých vzdialenosť od kamery je približne rovnaká by sa v intenzitnej textúre hodnotovo nemali veľmi líšiť. To znamená, že keď vyberieme pixely z hĺbkovej mapy, ktoré majú rovnakú hodnotu, potom by v intenzitnej textúre im pozíciou zodpovedajúce pixely mali mať tiež približne rovnakú hodnotu. Keď z hodnôt takto určených pixelov intenzitnej textúry urobíme aritmetický priemer, nemali by sa od neho hodnoty týchto pixelov veľmi líšiť. Ak by sa niektoré z týchto pixelov nachádzali blízko hrany, mohli by sme im túto priemernú hodnotu nastaviť a tým opraviť ich drobnú chybu, ktorá spôsobuje jemné rozmazanie hrany. Keďže by sa dané pixely hodnotou od priemeru nemali veľmi líšiť, pixely so správnymi hodnotami to až tak neovplyvní. Naopak pixely, ktoré svojou miernou odlišnosťou tvoria plynulý prechod hodnôt na hranách sa ustália na jednej určenej hodnote a tým sa chyba odstráni. Hĺbková mapa nám zaručuje jasný, ostrý schod hodnôt. To znamená, že sa budú na hrane samostatne priemerovať hodnoty pixelov pred jasnou hranicou a po jasnej hranici. Výsledkom budú dve priemerné hodnoty, ktoré sa v intenzitnej textúre daným pixelom blízko hrán nastavia. Získame tak ostrý schod medzi dvoma hodnotami na hranách v intenzitnej textúre.



Obr. 3.2: Vyznačené okolie hrany, ktorou úsečka pixelov prechádza.

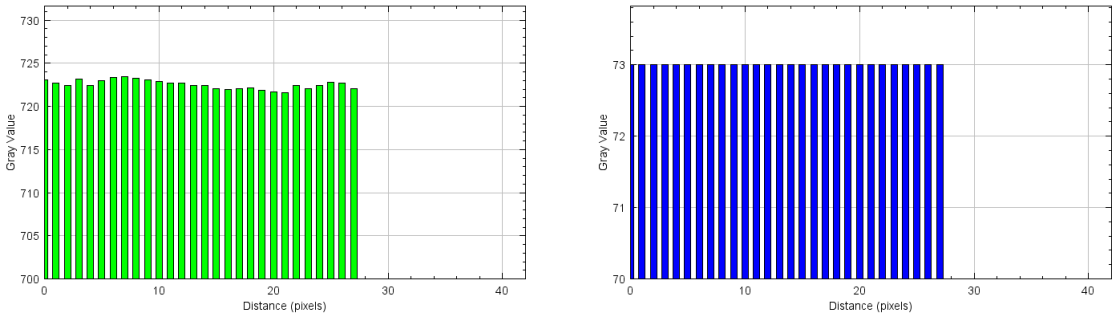
3.5.2 Algoritmus úpravy

Teraz opíšeme algoritmus, ktorý sme navrhli na úpravu intenzitnej textúry zo skenera. Vstupom pre náš algoritmus budú hĺbková mapa a intenzitná textúra, ktorú potrebujeme upraviť. V intenzitnej textúre chceme vylepšiť okolie hrán a plochy nechať nezmenené. Je teda nutné nejakým spôsobom odlíšiť okolie hrán od plôch.

Prvým krokom algoritmu je získanie okolia hrán vstupných obrazov. Tento problém sme sa rozhodli vyriešiť použitím Cannyho algoritmu na detekciu hrán v obraze [2]. Tento algoritmus sme našli implementovaný v knižnici OpenCV, ktorú v práci používame, a teda sa nebudeme bližšie zaoberať jeho vysvetlením. Cannyho algoritmus nám pri správnom nastavení parametrov vygeneruje čiernobiely obraz, ktorého pixely majú buď čiernu (hodnota 0) alebo svetlú farbu. Pixel je svetlý vtedy, ak cezeň v danom vstupnom obraze prechádza hrana. Máme teda obraz, v ktorom vieme zistiť pozície pixelov hrán vstupného obrazu. Ako sme už spomenuli vyššie, nás zaujíma blízke okolie hrán. V Cannyho výstupnom obraze nastavíme všetky pixely okolia svetlých pixelov tiež na svetlé. Jednoducho povedané, tenké svetlé čiary opisujúce hrany zmeníme na svetlé čiary hrubé viac pixelov. Hrúbka čiar bude konfigurovateľná podľa toho, aké veľké okolie hrán budeme chcieť upravovať. Takto sme získali množinu pixelov intenzitnej textúry, na ktoré chceme aplikovať úpravu. Sú to všetky svetlé pixely v našej pomocnej matici. Výsledky nájdeného okolia hrán ukážeme na našej testovacej úsečke pixelov. Oblasť, kde je v hĺbkovej mape na Obr. 3.1 hrana, máme vo výslednej matici s nájdeným okolím hrán vyznačenú vysokými hodnotami. Na Obr. 3.2 sme zobrazili graf našej testovacej úsečky z matice s vyznačenými okoliami hrán. Polomer okolia hrán bol pri generovaní tejto matice nastavený na 10 pixelov.

Keď už vieme, ktoré pixely budeme upravovať, potrebujeme získať samotné hodnoty, ktoré im treba nastaviť. Ideme teda určovať oblasti v intenzitnej textúre, ktorých zodpovedajúce pixely v hĺbkovej mape majú rovnakú hodnotu.

Hĺbkové mapy nie sú bezchybné. Body v priestore scény, ktoré sú v skutočnosti rovnako vzdialené od kamery môžu mať v hĺbkovej mape mierne iné hodnoty. Pixelov s rovnakými hodnotami v nich nájdeme len veľmi málo. Potrebujeme s nejakou presnosťou



Obr. 3.3: Vstup (vľavo) a výstup (vpravo) kvantizácie hĺbkovej mapy.

zistiť, ktoré body sú v realite rovnako vzdialené od kamery. Vstupnú hĺbkovú mapu teda rozdelíme do skupín po pixeloch, ktoré majú s nejakou malou odchýlkou rovnakú hodnotu. Na vytvorenie týchto skupín sme sa rozhodli použiť proces kvantizácie obrazu.

Kvantizácia je proces, pri ktorom rozdelíme hodnoty pixelov obrazu do disjunktných skupín zvaných kvantá. Zvolíme maximum novej škály hodnôt pixelov obrazu a aj počet hodnôt tejto škály. Každá jedna hodnota v škále reprezentuje práve jedno kvantum. Prenásobením všetkých hodnôt pixelov vhodnou konštantou vieme celý obraz preškálovať do našej zvolenej škály kvantizácie. V tejto novej škále je pevne zvolený počet hodnôt, ktoré môžu pixely mať. Ich hodnoty sa teda zaokrúhlia na im najbližšiu možnú hodnotu škály. Hovoríme, že pixely, ktorých hodnoty sa po preškálovaní zaokrúhlili na rovnakú hodnotu novej škály patria do toho istého kvanta, reprezentovaného danou hodnotou. Každý pixel obrazu po procese kvantizácie patrí do práve jedného kvanta. Keď dobre zvolíme škálu kvantizácie, dosiahneme tým, že pixely, ktoré mali v pôvodnom obraze veľmi podobnú hodnotu ju budú mať rovnakú a pixely, ktoré mali výrazne rozdielne hodnoty si svoju odlišnosť zachovali a nepadnú do rovnakého kvanta. Škálu kvantizácie hĺlbkovej mapy sme zvolili experimentálne. Výstupný obraz kvantizácie bude mať hodnoty pixelov 0-255 a tieto hodnoty budú celé čísla. Pri tejto škále sa vzťahy medzi hodnotami pixelov v hĺbkových mapách zachovali a ponúkli nám uspokojivú presnosť. Táto aproximácia hĺbkovej mapy zníži celkovú jej kvalitu, ale my potrebujeme len jednoznačne určiť, ktoré pixely tvoria oblasti s veľmi podobnými hodnotami. Znepresnenie hĺbkovej mapy nám teda v tomto prípade pomôže. Na Obr. 3.3 môžeme vidieť, ako sa jemne rozdielne hodnoty hĺbkovej mapy použitím kvantizácie zaokrúhlia na jednu presnú hodnotu v inej škále hodnôt obrazu.

Na Obr. 3.3 vidíme, že hodnoty pixelov vstupnej hĺbkovej mapy nie sú rovnaké, ale sú veľmi podobné. Kvantizácia spôsobí, že sa zaokrúhlia na jednu presnú hodnotu. Na Obr. 3.3 sa dá pozorovať aj zmena škály hodnôt obrazu.

Ďalším krokom algoritmu je získanie priemrných hodnôt pixelov z intenzitnej textúry, ktoré majú v kvantizáciou upravenej hĺbkovej mape rovnakú hodnotu. Po aplikácii kvantizácie na hĺbkovú mapu máme maticu, ktorej prvky sú celé čísla v škále 0 - 255.

Každá z týchto hodnôt reprezentuje inú množinu pixelov obrazu. Rozdelenie pixelov z upravenej hĺbkovej mapy použijeme na intenzitnú textúru, ktorú tak rozdelíme do 256 po dvoch disjunktných množín pixelov. Pre každú z týchto množín vieme vypočítať priemernú hodnotu jej pixelov. Po tomto kroku algoritmu vieme každej hodnote z upravenej hĺbkovej mapy priradiť priemernú hodnotu vypočítanú z intenzitnej textúry.

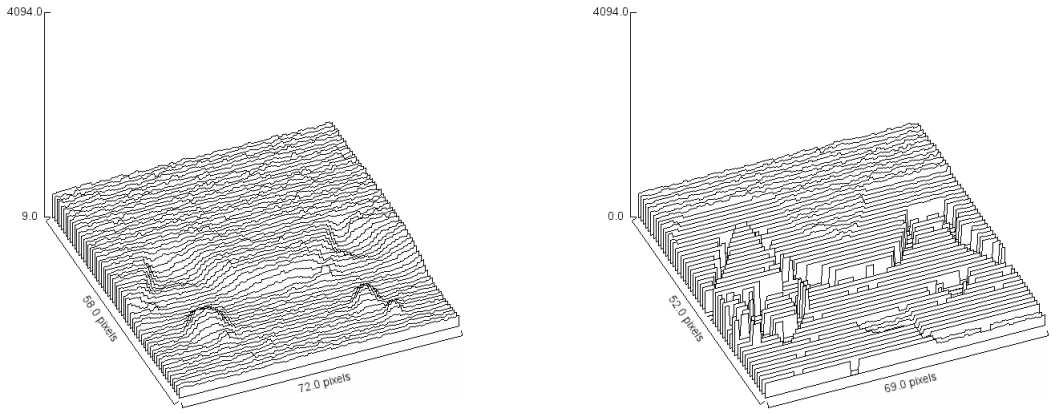
Posledným krokom algoritmu je spojenie výsledkov predchádzajúcich krokov. V tomo kroku budeme už prepisovať intenzitnú textúru. Budeme ju prechádzať po pixeloch. Ak je daný pixel v pomocnej matici s vyznačenými okoliami hrán svetlý, tak ho ideme zmeniť. Zapíšeme doň podiel hodnôt z poľa súčtov a počtov, kde ako index prvkov v oboch poliach zoberieme hodnotu daného pixela z upravenej hĺbkovej mapy. Tým dosiahneme, že v okolí hrán prepíšeme hodnoty pixelov, ktoré svojimi malými odchýlkami spôsobujú rozmazanie, na vypočítané priemerné hodnoty, ktoré pre guided filter zabezpečia potrebnú ostrú hranu v guidance obraze - intenzitnej textúre.

3.5.3 Výsledky úpravy intenzitnej textúry

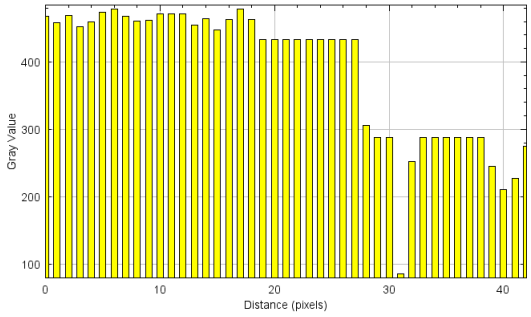
Náš spôsob úpravy intenzitnej textúry má vcelku uspokojiteľné výsledky. Na Obr. 3.4 vidíme, že na rovnej ploche zobrazenej časti obrazu je zachovaný šum z intenzitnej textúry. Naopak, v blízkom okolí hrany sú pixely spriemerované a na hrane samotnej je ostrý skok, ktorý sme chceli oproti pôvodnej intenzitnej textúre s rozmazanými hranami úpravou dosiahnuť. Algoritmus úpravy intenzitnej textúry, ktorý sme uviedli má parametre, ktoré sa dajú meniť, aby sme dosiahli pri danej scéne lepší výsledok. Hodnoty týchto parametrov závisia od špecifických vlastností skenovanej scény. Jedným parametrom je veľkosť okolia hrán, ktoré sa budú v intenzitnej textúre upravovať. Tento parameter môžeme meniť napríklad vtedy, keď chceme guided filter na vstupný obraz aplikovať s použitím väčšieho kernela. Druhý parameter, ktorý sa dá meniť je škála kvantizácie hĺbkovej mapy. Vieme určiť rozlíšenie jej hodnôt a tým pádom ovplyvniť, aké malé odchýlky hodnôt pixelov chceme zaokrúhľovať na tú istú hodnotu a aký hodnotový rozdiel pixelov je nutný na zachovanie hrany. Tento parameter potrebujeme nastaviť tak, aby boli všetky hrany rozlíšiteľné a plochy vyrovnané.

Výsledok algoritmu môžeme demonštrovať aj na našej testovacej úsečke pixelov. Obr. 3.5 ukazuje priebeh hodnôt na našej testovacej úsečke vo výslednom obraze algoritmu. Vidíme, že hodnoty pixelov na tomto obrázku pred vyznačenou oblasťou z Obr. 3.2 sú totožné s hodnotami pixelov z intenzitnej textúry na Obr. 3.1, takže plochy v obraze si zachovali svoj šum. Oblasť pixelov, ktoré sme chceli zmeniť, sme zmenili pomocou upravenej hĺbkovej mapy. Dostali sme rovnako širokú hranu, ako je v hĺbkovej mape na Obr. 3.1, len v škále hodnôt intenzitnej textúry.

Naša úprava intenzitnej textúry má jeden veľmi dobrý vedľajší účinok. Do určitej miery dokáže eliminovať lesk v scéne. Zmení síce len okolie hrán ale aj to vie často



Obr. 3.4: Vybraná časť pôvodnej (vľavo) a upravenej (vpravo) intenzitnej textúry.

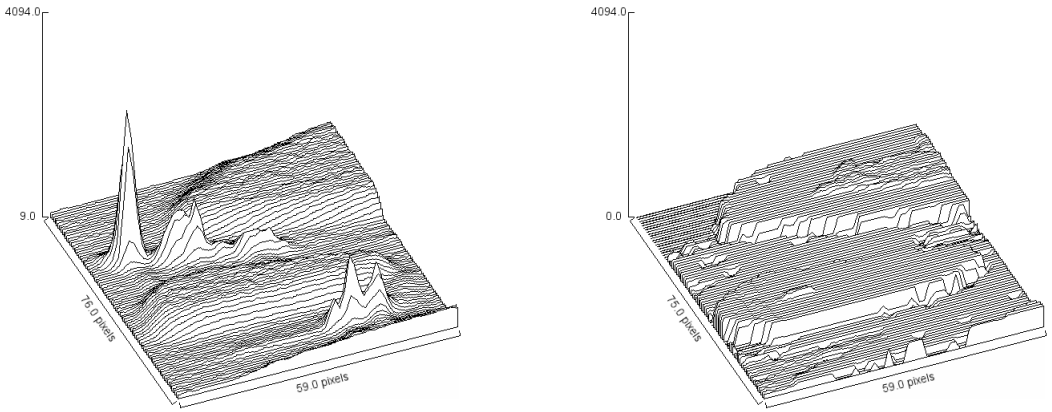


Obr. 3.5: Výsledok úpravy intenzitnej textúry zobrazený na testovacej úsečke pixelov.

výslednému obrazu pomôcť. Lesk sú nesprávne vysoké hodnoty pixelov. Ak priemerujeme väčšiu oblasť pixelov, ktoré majú nižšie hodnoty a s nimi nejaké množstvo pixelov ovplyvnených leskom, priemer všetky pixely ustáli na rovnakej hodnote a lesk sa rozptýli do väčšej oblasti. Na našich dátach sme odpozorovali, že toto rozptýlenie lesku je prospešné a pixely so správnymi hodnotami veľmi neovplyvní. Je to z toho dôvodu, že lesklých oblastí je spravidla málo a v priemerovaných oblastiach tvoria výraznú menšinu pixelov oproti celku. Touto úpravou sme teda docielili aj to, že vylepšujeme lesklé oblasti intenzitnej textúry. Príklad takéhoto vylepšenia môžeme vidieť na Obr. 3.6. Keď porovnáme pôvodnú intenzitnú textúru s opravenou, všimneme si, že na zuboch prevodového kolesa sme eliminovali lesk. Pixely povrchu zubov prevodového kolesa v upravenej intenzitnej textúre majú stálu, priemernú hodnotu. Detail eliminácie šumu spôsobenej našou úpravou sme zobrazili pomocou ImageJ [14] v 3D grafe na Obr.3.7.

Náš algoritmus opraví rozmazané hrany v intenzitnej textúre. Urobí jej hrany s uspokojiteľnou presnosťou podobne ostré ako hrany v hĺbkovej mape. Tým pádom upravený guidance obraz s väčšou presnosťou spĺňa predpoklady guided filtra. Podarilo sa nám aj na niektorých miestach v guidance obraze odstrániť chybný lesk. Ak je však nejaká oblasť veľmi ovplyvnená leskom, algoritmu sa jednoduchým priemerovaním, ktoré používa, nemusí podariť hrany v danej oblasti opraviť.

Obr. 3.6: Časť pôvodnej intenzitnej textúry obsahujúca lesk (vľavo) a upravenej intenzitnej textúry (vpravo), kde sa podarilo lesk do istej miery eliminovať.



Obr. 3.7: 3D graf hodnôt pixelov časti pôvodnej intenzitnej textúry obsahujúca lesk (vľavo) a upravenej intenzitnej textúry (vpravo).

Typ šumu	Príčiny	Guided filter
nedefinované pixely	tieň, lesk, chyba skenera	dodefinuje na plochách
chybné pixely	odrazené svetlo	neodstráni
artefakty laserovej projekcie	nasvietenie projekcie	vyhladí
nerovnosti plôch	nepresnosť skenera	vyhladí

Tabuľka 3.1: Prehľad eliminácie šumov

3.6 Aplikácia guided filtra

V predošlom texte sme definovali guided filter a analyzovali vstupné dáta vygenerované pre túto prácu. Vstupné obrazy sme upravili tak, aby s čo najväčšou presnosťou spĺňali potrebné predpoklady. Máme teda dva obrazy, ktorými sú hĺbková mapa a upravená intenzitná textúra. Hĺbková mapa je pre guided filter vstupným obrazom p. Guidance obrazom I je upravená intenzitná textúra. V tejto podkapitole budeme skúmať, ktoré chyby vstupného borazu p vieme pomocou guided filtra odstrániť. Pre niektoré typy chýb môže byť táto filtrovacia technika nevhodná. Bude nás zaujímať len samotné použitie guided filtra, prípadne jeho malej modifikácie. Nebudeme sa snažiť k filtru niečo pridať alebo aplikovať inú techniku navyše. Chceme preveriť schopnosti guided filtra pri použití na našich vstupných obrazoch.

V kapitole o vstupných dátach sme uviedli niekoľko typov šumu, ktorý hĺbková mapa, teda náš vstupný obraz p, obsahuje. Typy šumu, ktoré sa pokúsime pomocou guided filtra eliminovať sú: nedefinované pixely, neexistujúce pixely, artefakty laserovej projekcie a nerovnosti plôch v scéne. Teraz budeme tieto typy šumu podrobnejšie analyzovať a určíme, ktoré z nich vieme eliminovať a ktoré nie. Pre prehľadnosť sme zhrnuli výsledky analýzy šumov v Tab. 3.1.

3.6.1 Dodefinovanie pixelov

Keď sa nám podarilo opraviť intenzitné textúry, vyzerá to tak, že by naše vstupné dáta mali spĺňať predpoklady použitia guided filtra. Narážame však na jednu špecifickú vlastnosť dát, s ktorou jeho definícia nepočíta. Hĺbkové mapy obsahujú veľké množstvo nedefinovaných pixelov. Pri filtrovaní k nim musíme zaujať jasné stanovisko.

Prvá vec, ktorá by nám mohla napadnúť je nebrať tieto pixely ako nedefinované a pokúsiť sa guided filter aplikovať na celú hĺbkovú mapu. Keďže má intenzitná textúra všetky pixely definované a výsledný obraz je definovaný ako jej lokálna lineárna transformácia, mohli by sa v hĺbkovej mape niektoré nedefinované pixely dodefinovať. Na 2D obrázkoch to funguje veľmi dobre. Predstavme si napríklad situáciu, kedy je

Obr. 3.8: Časť skenovanej scény obsahujúca nedefinované pixely vybraná zo vstupnej hĺbkovej mapy (vľavo), výstupnej hĺbkovej mapy filtrovania (v strede) a z nej vypočítaného mračna bodov (vpravo).

jeden nedefinovaný pixel uprostred rovnej plochy definovaných pixelov. Nedefinovaný pixel má hodnotu 0. Všetky pixely v jeho okolí majú vyššiu hodnotu. Keďže sa jedná o rovnú plochu, guided filter by mal túto časť obrazu vyhladiť priemerovaním pixelov z jeho okolia. To nám spôsobí, že hodnota nedefinovaného pixela by sa zvýšila na priemer z jeho blízkeho okolia. Samozrejme by ovplyvnil aj hodnoty pixelov z jeho blízkeho okolia, ktoré čím by boli k nemu bližšie, tým by mali o niečo menšiu hodnotu. V 2D obrázkoch sa toto prejaví ako vylepšenie, rušivý pixel zmizne a hodnoty okolo neho sa zmenia len málo, tým pádom dostaneme peknú plochu, v ktorej sme eliminovali rušivý pixel. Nevieme však, ako to bude fungovať s naším 3D obrazom. Aby sme to zistili, vyskúšali sme guided filter aplikovať na vstupné dáta s tým, že sme do filtrovania zahrnuli aj nedefinované pixely. Výsledky tejto aplikácie filtra uvádzame na Obr. 3.8.

Na Obr. 3.8 vidíme ako čiernobiely 2D obraz zobrazenú časť definovanej rovnej plochy hĺbkovej mapy, ktorá obsahuje niekoľko nedefinovaných pixelov. Očakávali by sme, že guided filter tieto pixely, ktoré sa správajú ako šum, vyhladí. V 2D čiernobielom ponímaní, na ktorom bol filter definovaný výsledok vyzerá uspokojivo. Keď však nezobrazíme hĺbkovú mapu ako čiernobiely obraz, ale z nej vygenerujeme mračno bodov, zistíme, že sa okolo nedefinovaných pixelov vytvoril akýsi kužeľ okolitých pixelov. Toto sa deje z toho dôvodu, že hodnota 0, ktorou nedefinovaný pixel disponuje, je od okolitých pixelov veľmi vzdialená a priemer okolia ju nedokáže dostatočne zvýšiť. Svojou hodnotou pokazí aj priemer okolí blízkych pixelov a vytvoria sa kužely bodov.

Hodnota 0, ktorú majú nedefinované pixely nastavenú, sa nedá v hĺbkovej mape použiť ako relevantný údaj. Slúži len ako určenie toho, či sú dané pixely nedefinované. Pixely v hĺbkovej mape nesú informácie o vzdialenosti bodov od kamery v priestore. Vzdialenosť od kamery dĺžky 0 je teda údaj, ktorý žiaden bod scény mať nemôže a teda sa používa na informáciu o definovanosti daného pixela. Na obrázku Obr. 3.8 v mračne bodov kužely smerujú špicom ku kamere. Je to práve z toho dôvodu, že okolie nedefinovaného pixela, ktorý bol umiestnený vo vzdialenosti 0 od kamery sa snaží pritiahnuť jeho hodnotu bližšie k definovanej ploche, kde by mal byť.

Otázkou je, či je možné pomocou nástrojov, ktoré má guided filter k dispozícii, nejakým spôsobom pixely dodefinovať. Uviedli sme, že ich hodnoty nemôžeme používať normálnym spôsobom. Nástrojom priemerovania ich teda opraviť nevieme. Musíme sa chytiť niečoho iného. Prvým nápadom bolo, nedefinované pixely úplne ignorovať. Nedefinované pixely by sme prekopírovali aj do intenzitnej textúry a počas algoritmu by sme pixely s nulovou hodnotou jednoducho preskakovali. Guided filter by sme takto aplikovali iba na definované pixely. Toto samozrejme funguje a bola to prvá vec, ktorú sme pri implementácii filta vyskúšali. Prišli sme však na to, že vďaka modelu guided filtra nedefinované pixely vieme s dosť vysokou presnosťou dodefinovať. Potrebujeme si len uvedomiť čo hovorí jeho definícia a správnym spôsobom ju implementovať. Vráťme sa k definícii guided filtra. Výsledná hodnota pixela je pri použití guided filtra určená priamkou, ktorej parametre (a, b) získavame lineárnou regresiou z pixelov jeho okolia. Na to, aby sme získali pre daný pixel priamku preloženú cez pixely v jeho malom okolí, nepotrebujeme, aby boli všetky pixely jeho okolia definované. Na definovanie priamky by teoreticky stačili 2 definované pixely. Čím viac definovaných pixelov však v okolí počítaného pixela máme, tým kvalitnejšiu výslednú hodnotu vieme použitím lineárnej regresie dostať. Samotný pixel môže byť nedefinovaný. Dôležité je, že v intenzitnej textúre je každý pixel definovaný. Keďže parametre (a, b) vieme vypočítať z definovaných pixelov v danom okolí pixela a v intenzitnej textúre je každý pixel definovaný, vieme dosadiť do rovnice 3.12 nejaké hodnoty pre každý pixel vstupného obrazu p, či už je definovaný alebo nie. V tomto prístupe nedefinované pixely síce pri výpočte výsledného obrazu ignorujeme, ale s nejakou presnosťou im výslednú hodnotu vypočítať vieme. Záleží na tom, koľko pixelov z okolia daného pixela je definovaných a ako dobrú hodnotu má daný nedefinovaný pixel v intenzitnej textúre. Keďže sú v hĺbkovej mape tieto pixely nedefinované, v intenzitnej textúre ich našim algoritmom úpravy nevieme vylepšiť. Ak je teda nejaký nedefinovaný pixel v scéne na hrane, je dosť pravdepodobné, že jeho hodnota v intenzitnej textúre je nesprávna a neopravená. Ak je nedefinovaný pixel na ploche, jeho hodnota v intenzitnej textúre by mala byť správna, a teda by sme jeho výslednú hodnotu použitím modela guided filtra mali vedieť celkom presne vypočítať.

Primárnou úlohou guided filtra nie je dodefinovanie pixelov. Existujú iné techniky, ktoré sa tejto téme venujú a sú presnejšie. Túto funkcionalitu sme však v tejto práci implementovali, aby sme ukázali, že guided filter je vo svojej podstate schopný dodefinovania pixelov do hĺbkových máp. Dodefinovanie je však možné nepoužiť. Keď ho používateľ bude chcieť použiť, existuje na to prepínač, ktorý spôsobí, že program dodefinované pixely vo výslednom obraze nevymaže. Dodefinovanie pixelov má ešte jeden parameter, ktorým sa riadi. Je to nutné percento definovaných pixelov v kerneli. Ak by chcel program nejaký pixel dodefinovať, overí, či je v jeho kerneli aspoň nutné percento definovaných pixelov. Ak nie je, pixel ostane nedefinovaný. Toto opatrenie

umožňuje nastavenie minimálnej kvality pixelov výstupného obrazu.

3.6.2 Odstránenie chybných pixelov

Ukážku toho, ako takýto šum vyzerá, môžeme vidieť na Obr. 1.5. V 3D grafe vidíme zobrazené pixely, ktoré nepatria žiadnemu reálnemu objektu. Sú to chybné v realite neexistujúce pixely, ktoré sú obklopené nedefinovanou oblasťou. Vznikli pravdepodobne chybou odrazeného svetla pri skenovaní. Otázkou je, či takýto typ šumu vie guided filter z hĺbkovej mapy odstrániť. Pixely, ktoré potrebujeme odstrániť, sú obklopené nedefinovanými pixelami. V časti o nedefinovaných pixeloch v tejto kapitole sme povedali, že nedefinované pixely neberieme pri filtrovaní do úvahy. Guided filter tak, ako je definovaný, eliminuje šum tak, že robí priemer hodnôt pixelov z daného okolia. Pri odstránení šumu tohto typu čelíme teda dvom problémom. Prvým je, že tieto pixely nemajú definované okolie, ktorého hodnoty by ich priemerom opravili. Druhým problémom je, že aj keby sme povedali, že nejakým spôsobom budeme používať okolité 0-ové hodnoty, aby sme tento šum eliminovali, nepodarí sa nám to, lebo aritmetický priemer, ktorý sa robí z kladných a nulových hodnôt nikdy nedosiahne presnú hodnotu 0. Priemerovaním s okolitými pixelmi by sme dosiahli len to, že by sme tento šum rozmazali na maličké hodnoty blízko nuly, ktoré by možno na 2D čiernobielom obrázku šum odfiltrovali a pixely by neboli viditeľné. Pri vygenerovaní 3D mračna bodov by sme v konečnom dôsledku šum zväčšili. Ak nejaký pixel nemá hodnotu 0, je definovaný, a teda sa zobrazí v 3D priestore. Tento typ šumu teda aplikáciou guided filtra nevieme odstrániť.

3.6.3 Vyhladenie nerovností povrchu

V Kap. 1 sme uviedli, že povrch objektov scény je zašumený. V hĺbkových mapách nájdeme na povrchu objektov popri bežnom šume spôsobenom nepresnosťou merania skenera aj štruktúrovaný šum spôsobený nasvietenou laserovou projekciou. Na Obr. 1.3 vidíme, že plocha, ktorá je v realite rovná, má v naskenovanej hĺbkovej mape na sebe jemné vrúbky. Okrem vrúbok obsahuje aj neštruktúrovaný šum. Cieľom filtrovania je tieto vrúbky aj šum vyhladiť. Guided filter je vhodný na použitie pri takýchto typoch šumov. Guidance obraz tieto vrúbky neobsahuje a tým pádom je na plochách medzi vstupným obrazom p a guidance obrazom I nízka korelácia. Vtedy model guided filtra hodnoty pixelov priemeruje, a teda vyhladí. Tento typ šumu budeme v tejto práci primárne odstraňovať. Budeme sa snažiť vyhladiť definované plochy v scéne a pritom zachovať hrany. Graf hodnôt pixelov úsečky z Obr. 1.3 by vo výslednej hĺbkovej mape mal vyzerať ako rovná čiara, lebo sa táto úsečka v scéne nachádza na rovnej ploche povrchu objektu. Vrúbky aj náhodný šum by sa mali vyhladiť priemerovaním okolí pixelov.

Kapitola 4

Implementácia

V predošlých kapitolách sme opisovali vstupné dáta, s ktorými budeme v tejto práci pracovať. Na tieto dáta sme sa rozhodli aplikovať guided filter. Preskúmali sme možnosti jeho použitia a uviedli sme, aké typy chýb touto filtrovacou technikou vieme odstrániť a aké nie. V tejto kapitole budeme opisovať našu implementáciu tohto filtra. Opíšeme aj implementáciu úpravy intenzitnej textúry, ktorá je nutnou podmienkou pre používanie guided filtra na naše dáta. Najskôr uvedieme detaily našej prvej implementácie guided filtra a úpravy textúry, ktorá pracuje v jednom vlákne na CPU, a teda dáta nespracúva paralelne. Potom uvedieme teóriu potrebnú na paralelizáciu CPU implementácie. Nakoniec opíšeme našu implementáciu guided filtra, ktorá dáta spracováva paralelne vo viacerých vláknach na GPU.

4.1 CPU Implementácia guided filtra

Naša prvá implementácia guided filtra, ktorú budeme opisovať, bude dáta spracovávať sériovo. Keď overíme, že dosahuje na vstupných dátach uspokojivé výsledky, budeme ju považovať za náš referenčný bod pre zefektívnenie paralelizáciou. Táto implementácia bude na svoj chod používať CPU a pracovať bude v jednom vlákne. Filter budeme implementovať v programovacom jazyku C++. Pre prácu s obrazom budeme používať knižnicu OpenCV, verziu 3.4.10 [11]. Pri implementácii sme sa inšpirovali publikáciou [7], kde je algoritmus guided filtra implementovaný v jazyku Matlab, aplikovaný na 2D čiernobiele obrázky. Algoritmus guided filtra potrebujeme aplikovať na naše vstupné dáta. V tejto podkapitole budeme opisovať našu implementáciu tohto algoritmu, ktorá pracuje s našimi vstupnými dátami.

Algoritmus guided filtra (Algoritmus 1) má päť krokov. Ich význam sme podrobnejšie skúmali v kapitole 3. Kroky algoritmu teda už analyzovať nebudeme. Náš program bude so vstupnými obrazmi robiť to, čo je v krokoch algoritmu definované. Teraz sa budeme zaoberať implementovaním funkcií na obrazoch, ktoré sa v krokoch používajú. Konkrétne

môžeme funkcie, ktoré sa v algoritme na obrazy aplikujú rozdeliť na dva typy funkcií: funkcia $f_{priemer}$ a aritmetické operácie. Najskôr uvedieme, ako sme implementovali aritmetické operácie a potom opíšeme našu implementáciu funkcie $f_{priemer}$.

4.1.1 Implementácia aritmetických operácií

Všetky aritmetické operácie, ktoré sa v algoritme guided filtra na maticiach vykonávajú sú definované ako operácie po jednotlivých prvkoch matíc. Pre operácie sčítania a odčítania matíc teda používame ich matematickú definíciu. Operácie násobenia a delenia matíc sa robia tiež po prvkoch matice. Pre odlíšenie týchto operácii od matematicky definovaného násobenia a delenia matíc v algoritme 1 používame označenie $.*$ pre násobenie po prvkoch a ./ pre delenie po prvkoch.

Jedinú vec, ktorú musíme ošetriť je delenie nulou. Situácia, kedy delíme nulou nám môže v algoritme guided filtra pri našich vstupných maticiach nastať v jednom známom prípade. V maticiach existujú nedefinované pixely, ktoré majú nastavenú hodnotu 0. V algoritme sa rátajú priemery obrazov, ktoré sa pre každý pixel vypočítajú ako podiel súčtu hodnôt pixelov z okolia daného pixela a počtu definovaných pixelov v jeho okolí. Počet definovaných pixelov v okolí môže byť nulový. Tým pádom je pri delení menovateľ nulový. Ak je niektorý pixel nedefinovaný a nedefinované sú aj všetky pixely v jeho okolí, v priemernom obraze má byť tento pixel tiež nedefinovaný, a teda výsledná hodnota pixela bude 0. V implementácii operácie delenia matíc teda pribudne podmienka, ktorá hovorí, že ak je menovateľ podielu nulový, operáciu nespravíme a výsledkom bude nula.

Máme definovanú aj operáciu sčítania, ktorej jeden operand je matica a druhý operand konštanta. Výsledkom takejto operácie je matica, ktorej prvky sú prvky vstupnej matice, ku ktorým sme pripočítali vstupnú konštantu.

Operácie implementujeme ako funkcie, ktorých vstupnými parametrami sú dve matice. Vo funkcii najprv vytvoríme prázdnu maticu rozmerov totožných s prvou vstupnou maticou. Matice majú v našej implementácii vždy rovnaké rozmery. Túto maticu prechádzame po prvkoch. Do prázdneho prvku vždy zapíšeme výsledok danej aritmetickej operácie aplikovanej na prvky vstupných matíc, ktoré tomuto prvku súradnicami zodpovedajú. Pri operácii delenia sa operácia na prvky aplikuje, len ak je splnená podmienka o delení nulou. Inak bude mať daný prvok vo výslednej matici hodnotu 0. Na konci máme celú maticu naplnenú hodnotami, ktorú vrátime na výstupe funkcie.

Operácia sčítania matice a konštanty je implementovaná analogicky k operáciám na dvoch maticiach. Rozdiel je len v tom, že sa do prázdneho prvku výslednej matice vždy vloží súčet vstupnej konštanty a daného prvku vstupnej matice.

Časová zložitosť všetkých uvedených funkcií je lineárna od počtu prvkov vstupných

matíc, lebo pre každý prvok matice urobíme konštantný počet operácií.

4.1.2 Implementácia funkcie $f_{priemer}$

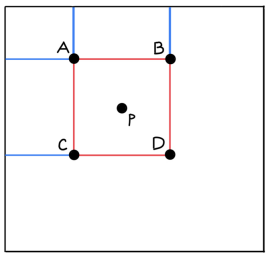
Funkcia $f_{priemer}$ je pri guided filtri definovaná ako funkcia s dvoma vstupnými parametrami. Tieto parametre sú vstupný obraz a polomer kernela. Vo výslednom obraze je hodnota každého pixela daná aritmetickým priemerom hodnôt pixelov jeho kernela zo vstupného obrazu [8]. Z definície aritmetického priemeru potrebujeme pre každý pixel získať dve hodnoty: súčet hodnôt pixelov kernela daného pixela vstupného obrazu a počet týchto pixelov kernela. Podiel týchto dvoch hodnôt nám dá výslednú priemernú hodnotu daného pixela. Z vyššie uvedeného vyplýva, že potrebujeme implementovať dve funkcie: f_{sucet} a f_{pocet}. Výsledné matice týchto dvoch funkcií budú reprezentovať súčty hodnôt pixelov a počty pixelov v jednotlivých kerneloch pixelov vstupného obrazu. Výsledná matica funkcie $f_{priemer}$ bude daná nasledujúcim vzťahom:

$$f_{priemer} = \frac{f_{sucet}}{f_{pocet}}. \tag{4.1}$$

Tento podiel získame aplikovaním nami implementovanej aritmetickej operácie delenia matíc. V nasledujúcich sekciách textu budeme opisovať našu implementáciu funkcií f_{sucet} a f_{pocet}. V publikácii o guided filtri [8] má $f_{priemer}$ časovú zložitosť lineárnu od počtu pixelov jej vstupného obrazu. Časová zložitosť funkcií f_{sucet} a f_{pocet} musí teda byť rádovo maximálne lineárna od počtu pixelov vstupného obrazu.

Funkcia f_{sucet} má rovnaké vstupné parametre ako $f_{priemer}$. Táto funkcia má pre každý pixel vstupného obrazu vypočítať súčet hodnôt pixelov jeho kernela, ktorého polomer r dostane ako vstupný parameter. Prvé riešenie tohto problému, ktoré by nám mohlo napadnúť, je pre každý pixel postupne sčítať všetky hodnoty pixelov so vzdialenosťou od daného pixela menšou alebo rovnakou ako polomer kernela. Tým by sme zaručene dostali správnu výslednú maticu. Toto riešenie je však dosť neefektívne, lebo s rastúcim polomerom kernela rastie aj počet prístupov do matice, ktoré pri prechádzaní matice robíme. Pre každý jeden pixel vstupného obrazu by sme potrebovali $(2r+1)^2$ prístupov do matice. Výsledná časová zložitosť tejto funkcie by pri n vstupných pixeloch bola $O(n(2r+1)^2)$, čo je stále $O(n)$, keďže r je vstupná konštanta. S rastúcim r sa však konštanta, ktorou počet krokov násobíme, kvadraticky zväčšuje, čo je veľmi neefektívne. V našej implementácii chceme dosiahnuť to, aby časová zložitosť funkcie f_{sucet} bola lineárna od počtu pixelov vstupného obrazu a pritom nebola závislá na veľkosti kernela.

Existuje viac rôzne efektívnych riešení tohto problému. My sme sa rozhodli použiť na získavanie súčtov hodnôt v kerneloch takzvaný integrálny obraz [5]. Integrálny obraz vstupnej matice je matica, v ktorej hodnota každého prvku je daná súčtom všetkých hodnôt prvkov z oblasti vľavo hore od daného prvku, vrátane samotného



Obr. 4.1: Integrálny obraz

Obr. 4.2: Prvá (vľavo) a druhá (vpravo) fáza procesu získavania súčtov hodnôt kernelov z integrálneho obrazu.

prvku. Súčty hodnôt prvkov ľubovoľných obdĺžnikových oblastí vieme získať veľmi efektívne, pomocou len štyroch prístupov do matice. Takéto získanie súčtu ukážeme na obrázku:

Na Obr. 4.1 je načrtnutý integrálny obraz. Červený štvorec so stranou dĺžky $2r + 1$ pixelov, kde r je polomer kernela, znázorňuje kernel pixela p. Ak má pixel p vo vstupnom obraze súradnice (x, y), potom majú prvky A, B, C a D v integrálnom obraze nasledujúce súradnice:

$$\begin{aligned} A: &\quad (x - r - 1, y - r - 1) \\ B: &\quad (x + r, y - r - 1) \\ C: &\quad (x - r - 1, y + r) \\ D: &\quad (x + r, y + r) \end{aligned}$$

Súčet všetkých hodnôt kernelu pixela p sa z definície integrálneho obrazu dá vypočítať ako hodnota $D - C - B + A$. Takúto hodnotu vypočítame pre každý pixel vstupného obrazu. Implementáciu procesu vypočítania hodnôt výsledných pixelov z integrálneho obrazu sme rozdelili do dvoch fáz, ktoré opíšeme na nasledujúcich obrázkoch:

Prvok výslednej matice prvej fázy so súradnicami (x, y) definujeme ako rozdiel prvkov integrálneho obrazu načrtnutého na Obr. 4.1 so súradnicami $(x + r, y)$ a $(x - r - 1, y)$. Vo výslednej matici prvej fázy, načrtnutej na Obr. 4.2 vľavo, máme vypočítané rozdiely hodnôt $D - C$ a $B - A$. Prvok výslednej matice druhej fázy so súradnicami (x, y) definujeme ako rozdiel prvkov výstupnej matice z prvej fázy so súradnicami $(x, y + r)$ a $(x, y - r - 1)$. Vo výslednej matici druhej fázy, načrtnutej na Obr. 4.2 vpravo, má pixel p vypočítanú výslednú hodnotu $D - C - B + A$. V obidvoch fázach pre každý pixel

vstupného obrazu pristupujeme k dvom prvkom matice. Dokopy sú to štyri prístupy do matice pre každý pixel. Počet prístupov bude konštantný pre sčítanie ľubovoľne veľkého kernela. Časová zložitosť počítania výsledného obrazu z integrálneho obrazu uvedeným spôsobom je teda pri n pixeloch vstupného obrazu $O(n)$.

Otázkou je, ako vytvoríme integrálny obraz vstupného obrazu a koľko času to bude stáť. Integrálny obraz v našej implementácii vytvárame pomocou kumulatívnych súm. Kumulatívnu sumu riadku matice vytvoríme tak, že postupne ku každému prvku v riadku pripočítavame súčet všetkých jeho predošlých prvkov. Analogickým spôsobom vieme vytvoriť kumulatívnu sumu stĺpca matice. Implementácia kumulatívnej sumy je triviálna. Prechádzame po prvkoch daný riadok alebo stĺpec matice. V pomocnej premennej si ukladáme súčet všetkých predošlých prvkov. K prvku vždy pripočítame obsah tejto pomocnej premennej. Potom obsah premennej zmeníme na aktuálnu hodnotu a pokračujeme na ďalší prvok. Keď takto prejdeme celý riadok alebo stĺpec, máme v každom jeho prvku súčet jeho pôvodnej hodnoty a hodnôt všetkých jeho predchodcov. Integrálny obraz matice vytvárame v našej implementácii tak, že vytvoríme kumulatívnu sumu (smerom doprava) na všetkých riadkoch vstupnej matice. Na výslednej matici, ktorej riadky sú kumulatívne sumy pôvodných riadkov matice, vytvoríme kumulatívne sumy na všetkých jej stĺpcoch (smerom dole). Prvky výslednej matice obsahujú súčty prvkov matice vľavo hore od nich, vrátane ich hodnoty, čiže tvoria integrálny obraz vstupnej matice. Ak n je počet pixelov vstupného obrazu, tak tento proces počas behu urobí $2n$ prístupov do matice. Všetky prvky matice navštívime 2-krát. Najskôr pri vytváraní kumulatívnych súm po riadkoch a potom po stĺpcoch.

Pri vytváraní integrálneho obrazu pomocou kumulatívnych súm riadkov a stĺpcov matice sme odpozorovali, že výsledné hodnoty prvkov integrálneho obrazu boli plné chybných hodnôt. Po analýze tohto problému sme zistili, že ide o numerickú chybu pri sčítavaní prvkov matíc. Keďže sú prvky matíc čísla typu float, treba brať do úvahy fakt, že pri sčítavaní dvoch veľmi rôznych čísel nastáva strata presnosti výsledku. Táto strata presnosti je zapríčinená zjednocovaním exponentov pri aritmetických operáciach na číslach typu float. Pri vytváraní kumulatívnej sumy spravidla sčítavame veľké číslo (súčet všetkých predošlých prvkov) s malým číslom (hodnota daného prvku). Každou jednou operáciou sčítania sa v riadku alebo stĺpci kumuluje stále väčšia chyba. V integrálnom obraze a tým pádom aj celkovo vo výstupnom obraze guided filtra to spôsobuje závažnú chybu. Čím je nejaký pixel umiestnený v obraze viac vpravo a dole, tým je jeho hodnota viac chybná. Pri riešení tohto problému sme sa opreli o nasledujúcu úvahu.

Čím je obraz, ktorého integrálny obraz chceme vypočítať menší, tým menšia chyba sa v hodnotách jeho pixelov nakumuluje. Ďalej vieme, že guided filter, ktorý v konečnom dôsledku na obraze počítame, je lokálny lineárny filter. To znamená, že každý pixel jeho výsledného obrazu je ovplyvnený len pixelmi vstupných obrazov patriacich do

jeho kernela. Ak by sme zo vstupného obrazu vysekli nejaký blok pixelov a spustili na ňom samostatne guided filter, dosiahli by sme pre pixely, ktorých celé kernely patria do tohto bloku rovnakú výslednú hodnotu ako pri spúšťaní filtra na celom obraze. Pixely, ktorých kernely blok obsahuje len čiastočne, budú mať tiež vypočítanú hodnotu, ale bude o niečo menej presná. Lineárna regresia, ktorú guided filter používa na počítanie hodnoty daného pixela prekladá priamku cez pixely jeho kernela. Čím je týchto pixelov menej, tým menej presnú hodnotu guided filter danému pixelu vypočíta.

Na základe tejto úvahy sme sa rozhodli rozdeliť vstupný obraz na štvorcové bloky pixelov a spustiť guided filter na každom bloku samostatne. Tým sme dosiahli menšiu chybu pri počítaní integrálnych obrazov a výsledný obraz v každom bloku bol výrazne presnejší ako pri spúšťaní guided filtra na celom obraze. Jedinou malou nepresnosťou tohto riešenia sú hranice blokov v obraze. Kernely pixelov, ktoré sú blízko okraja bloku majú menej pixelov, z ktorých sa počíta ich výsledná hodnota. Lineárna regresia však tieto hodnoty počíta s uspokojivou prenosťou a tieto chyby v hodnotách sú výrazne menšie ako tie, ktoré sa kumulovali pri filtrovaní celého obrazu. Počítaním integrálnych obrazov na menších blokoch vstupného obrazu sme dosiahli to, že hodnoty pixelov výsledného obrazu sú výrazne presnejšie vo všetkých častiach obrazu. Eliminovali sme tak kumulovanie chyby spôsobenej aritmetickými operáciami na float číslach.

Vytvorenie integrálneho obrazu a vypočítanie súčtov hodnôt v kerneloch sú dva po sebe nasledujúce procesy. Výstup prvého procesu je vstupom pre druhý proces. Pre neskoršie opisovanie paralelnej implementácie sa nám hodí zadefinovať tieto dva procesy ako samostatné funkcie: $f_{integral}$ a f_{okolie}. Výstup funkcie $f_{integral}$ je integrálny obraz vstupného obrazu. Tento integrálny obraz bude vstupným obrazom pre funkciu f_{okolie}, ktorá z neho vypočíta súčty hodnôt z okolia pixelov vstupného obrazu. Pomocou týchto dvoch funkcií vieme teda funkciu f_{sucet} vyjadriť nasledovne:

$$f_{sucet} = f_{okolie} \circ f_{integral}. \tag{4.2}$$

Takto definovaná funkcia f_{sucet} má pri n pixeloch vstupného obrazu celkovú časovú zložitosť $O(2n+4n) = O(6n)$, čo je rádovo stále $O(n)$. Konštanta, ktorou n násobíme, sa s rastúcim polomerom kernela nemení. Tým sme splnili cieľ, ktorý sme chceli pomocou integrálneho obrazu dosiahnuť. Máme implementáciu funkcie f_{sucet}, ktorá má časovú zložitosť lineárnu od počtov pixelov vstupného obrazu, nezávislú od veľkosti kernela.

Keď sme opísali implementáciu funkcie f_{sucet}, ostalo nám implementovať už len funkciu f_{pocet}. Ak sa zamyslíme nad definíciou funkcie f_{pocet} zistíme, že je špeciálnym prípadom funkcie f_{sucet}. Keď na maticu samých jednotiek aplikujeme funkciu f_{sucet}, dostaneme výslednú maticu reprezentujúcu počty pixelov v kerneloch. Je to z toho dôvodu, že každý prvok kernela do súčtu jeho hodnôt prispel hodnotou 1.

Jedinú vec, ktorú pri implementácii tejto funkcie musíme ošetriť je to, že náš

vstupný obraz obsahuje nedefinované pixely s hodnotou 0, ktoré nemôžeme brať pri priemerovaní hodnôt pixelov obrazu do úvahy. Tieto nedefinované pixely musíme teda nejakým spôsobom z priemerovania vyradiť. Urobíme to tak, že nedefinované pixely vstupného obrazu v matici jednotiek pred aplikáciou funkcie f_{sucet} prepíšeme na hodnoty 0. Tým pádom ich nezahrnieme pri priemerovaní do menovateľa podielu súčtu hodnôt pixelov a počtu pixelov. Pri počítaní priemerných hodnôt pixelov ich teda budeme ignorovať. Funkcia f_{pocet} nám teda v konečnom dôsledku vráti počty definovaných pixelov v kerneloch vstupného obrazu. Kontrola, či je pixel definovaný a jeho prípadné prepísanie v matici jednotiek má lineárnu časovú zložitosť od počtu pixelov vstupného obrazu a funkcia f_{sucet} tiež. Tým pádom má lineárnu časovú zložiosť od počtu pixelov vstupného obrazu aj funkcia f_{pocet}.

4.1.3 Zhrnutie CPU implementácie guided filtra

V tejto podkapitole sme opísali našu implementáciu funkcií, ktoré sa používajú v krokoch algoritmu guided filtra (Algoritmus 1). Implementovali sme aritmetické operácie na maticiach a funkciu $f_{priemer}$. Z týchto funkcií vieme vyskladať každý krok uvedeného algoritmu. Všetky nami implementované funkcie majú časovú zložitosť lineárnu od počtu pixelov vstupného obrazu, nezávislú od veľkosti kernela, ktorý pri guided filtri použijeme. Keďže v algoritme sa naše funkcie volajú len konštantne veľa krát, je jeho časová zložitosť tiež lineárna od počtu pixelov vstupného obrazu. Tým sme zachovali časovú zložitosť guided filtra, uvedenú v publikácii [8]. Aby sme eliminovali kumulujúcu sa chybu pixelov obrazu, rozhodli sme sa vstupný obraz rozdeliť do blokov, na ktorých spúšťame guided filter samostatne. Tým sme dosiahli uspokojivú presnosť hodnôt pixelov výsledného obrazu vo všetkých jeho častiach.

4.2 CPU Implementácia úpravy intenzitnej textúry

Implementáciu úpravy intenzitnej textúry sme do značnej miery opísali pri predstavení jej algoritmu. Uvedieme implementačné detaily tohto procesu, ktoré ešte neboli spomenuté. Algoritmus úpravy intenzitnej textúry pozostáva z týchto krokov: detekcia hrán, vyznačenie okolia hrán, kvantizácia hĺbkovej mapy, priemerovanie oblastí intenzitnej textúry, úprava hodnôt pixelov. Teraz si rozoberieme implementácie jednotlivých krokov algoritmu a odhadneme ich časovú zložitosť.

Na detekciu hrán vo vstupnom obraze aplikujeme Cannyho algoritmus [2]. Používame jeho implementáciu z knižnice OpenCV. Podľa publikácie má tento algoritmus pri počte n pixelov vstupného obrazu a voliteľnom polomere kernela r časovú zložitosť $O(n \cdot (2r+1)^2)$.

Vyznačenie okolia hrán je vlastne len rozmazanie čiar pixelov s nenulovými hodno-

tami, ktoré vytvorí Cannyho algoritmus. Stačí si uvedomiť, že nami implementovaná priemerovacia funkcia $f_{priemer}$ rozmazáva obraz na konfigurovateľne veľkom okolí pixelov. Okolie nájdených hrán teda vyznačíme tak, že na výstupnú Cannyho maticu použijeme funkciu $f_{priemer}$. Tá ich rozmaže na ich potrebne veľké okolie. Vo výslednom obraze tohto kroku sú teda všetky pixely, ktoré chceme pri úprave meniť tie s nenulovými hodnotami.

Proces kvantizácie je veľmi jednoduchý. Škála kvantizácie, ktorú sme experimentálne zvolili, je škála celých čísel od 0 po 255. Kvantizáciu vstupnej matice sme teda implementovali tak, že sme každý pixel prenásobili hodnotou 255 a vydelili maximálnou hodnotou vstupnej matice. Použili sme pritom celočíselné delenie, ktoré nám zabezpečilo potrebné zaokrúhlenie hodnôt. V tomto kroku sme použili nami implementované aritmetické operácie násobenia a delenia. Okrem toho sme implementovali jednoduchú funkciu pre nájdenie maximálnej hodnoty pixela matice, aby sme vedeli správne hodnoty preškálovať. Aritmetické operácie na maticiach pracujú v lineárnom čase od počtu vstupných pixelov. To isté platí aj o nájdení maximálnej hodnoty v matici. Tým pádom je aj časová zložitosť kvantizácie lineárna od počtu pixelov vstupného obrazu.

Ďalším krokom algoritmu úpravy je priemerovanie oblastí intenzitnej textúry. Kvantizácia nám určila 256 disjunktných množín pixelov obrazu. Toto rozdelenie aplikujeme na intenzitnú textúru. Teraz chceme pre každú z týchto množín získať priemernú hodnotu z jej pixelov. Do ktorej množiny pixel intenzitnej textúry patrí, určuje hodnota tohto pixela v kvantizáciou upravenej hĺbkovej mape. V našej implementácii tohto kroku algoritmu používame dve polia veľkosti 256 prvkov. V prvom poli budeme priebežne ukladať pre každú množinu súčet hodnôt jej pixelov. V druhom poli budeme pre každú množinu priebežne ukladať počet jej pixelov. Opísané polia budeme plniť nasledujúcim spôsobom: budeme po pixeloch prechádzať intenzitnú textúru. Vždy pripočítame hodnotu daného pixela do poľa súčtov na index, ktorým je hodnota daného pixela v kvantizáciou upravenej hĺbkovej mape. Do poľa počtov na ten istý index pripočítavame jednotku. Keď takto prejdeme celú intenzitnú textúru, vieme pre každú množinu určenú hodnotou k vypočítať priemernú hodnotu jej pixelov ako podiel hodnoty z poľa súčtov a hodnoty z poľa počtov na indexe k. Časová zložitosť našej implementácie tohto kroku je lineárna od počtu pixelov vstupného obrazu, lebo jediné, čo robíme je, že raz prejdeme po prvkoch intenzitnú textúru a na každom jej prvku urobíme len konštantne veľa artimetických operácii a jeden prístup do upravenej hĺbkovej mapy.

V poslednom kroku algoritmu upravujeme hodnoty správnych pixelov na hodnoty získané algoritmom. Len spojíme to, čo sme získali v predošlých krokoch. Tento krok algoritmu sme implementovali tak, že po pixeloch prechádzame intenzitnú textúru a vždy na základe vyznačených okolí hrán v pomocnej matici kontrolujeme, či máme daný pixel upravovať. Ak máme daný pixel opraviť, tak doň zapíšeme priemernú hodnotu z pixelov množiny, v ktorej sa nachádza. V ktorej množine sa pixel nachádza zisťujeme

z jeho hodnoty v kvantizáciou upravenej hĺbkovej mape. Priemernú hodnotu získame podielom správnych prvkov z pomocných polí z predchádzajúceho kroku algoritmu. Časová zložitosť tohto kroku je zjavne lineárna od počtu pixelov vstupného obrazu, lebo pre každý pixel vykonáme konštantne veľa operácií.

Opisovali sme našu implementáciu algoritmu úpravy intenzitnej textúry, ktorá pracuje v jednom vlákne na CPU. Všetky kroky algoritmu okrem kroku detekcie hrán sme implementovali s časovou zložitosťou lineárnou od počtu pixelov vstupného obrazu. Krok detekcie hrán má pri n vstupných pixeloch a zvolenom polemere kernela r časovú zložitosť $O(n \cdot (2r+1)^2)$. Tým pádom je aj časová zložitosť celého algoritmu úpravy intenzitnej textúry $O(n \cdot (2r+1)^2)$.

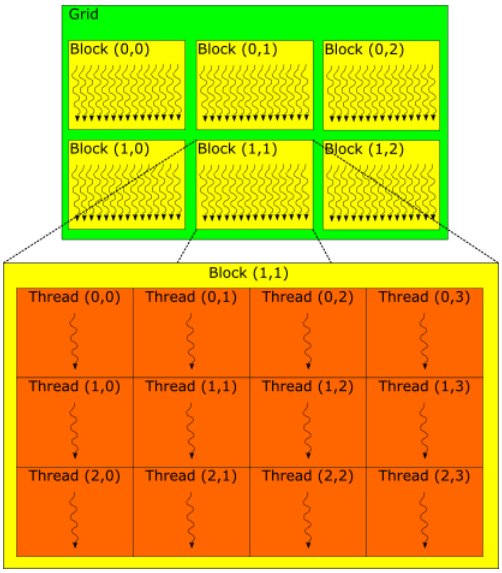
4.3 Paralelizácia na GPU

Táto podkapitola bude úvodom potrebným pre opisovanie našej paralelnej GPU implementácie guided filtra. Základné pojmy aj princípy fungovania nami využívanej GPU architektúry, ktoré v tejto podkapitole uvedieme, sú spracované z verejne dokumentácie paralelizačnej CUDA paltformy od spoločnosti NVIDA [10].

Implementácia, ktorú sme doteraz opisovali, pracovala v jednom vlákne. Teraz chceme dáta spracovávať paralelne vo viacerých vláknach. Na jednovláknovú implementáciu bolo efektívne používať výpočtové jednotky CPU, lebo sú navrhnuté na čo najrýchlejšie vykonanie jednej úlohy. Pri paralelnej implementácii filtra budeme používať výpočtové jednotky GPU, ktoré sú navrhnuté na vykonanie čo najviac úloh v minimálnom čase. V zadaní práce je uvedené, že paralelný GPU filter má byť implementovaný v CUDA. CUDA (Compute Unified Device Architecture) je platforma pre paralelné výpočty a model programovania vyvíjaný spoločnosťou NVIDIA. Táto platforma nám dáva možnosť využitia GPU jadier, podporujúcich CUDA platformu, na účel všeobecného spracovania dát. Výhodou pri používaní CUDA je to, že ponúka rozšírenie jazyka C++, kde sa dá potrebný paralelizmus vyjadriť pomocou pár kľúčových slov. Budeme teda v jazyku C++ programovať procesy, ktoré sa budú spúšťať paralelne na GPU. V tejto podkapitole opíšeme teoretický model paralelného programovania, ktorý budeme používať a uvedieme, ako je reprezentovaný v CUDA platforme. Nakoniec opíšeme zariadenie, na ktorom sa bude naša GPU implementácia filtra spúšťať.

4.3.1 Model paralelného programovania

Majme vstupnú množinu dát, ktorej prvky potrebujeme nejakým spôsobom spracovať. Ak chceme na všetky prvky tejto množiny aplikovať ten istý proces spracovania, tak by mohli inštancie daného procesu na jednotlivých prvkoch bežať súčasne. Náš model definuje vstupnú množinu dát ako množinu, ktorej prvky sa majú spracovať



Obr. 4.3: Hierarchia vlákien v CUDA [10].

rovnakým spôsobom. Ďalej definuje funkciu, ktorá sa má aplikovať na jednotlivé prvky vstupnej množiny. Túto funkciu budeme označovať f_{prvok}. Proces vykonania funkcie f_{prvok}, aplikovanej na prvok vstupnej množiny, konkrétnou programovateľnou jednotkou definujeme ako vlákno. Pre danú vstupnú množinu, ktorej prvky chceme spracovávať paralelne, teda potrebujeme implementovať funkciu f_{prvok}, ktorej inštancie budú pre každý prvok vykonávané paralelne v samostatných vláknach.

4.3.2 Reprezentácia modelu v CUDA

V CUDA platforme sa funkcia f_{prvok} nazýva *kernel*. Vlákno je tu definované ako inštancia *kernela*. Kernel sa volá z CPU a vykonáva sa na GPU vo viacerých vláknach. Pre kernel potrebujeme zadefinovať koľko jeho inštancií sa má spustiť. Musíme tiež vedieť vlákna identifikovať, aby sme im vedeli priradiť vstupné prvky, ktoré majú spracovať. Aby sme vedeli s vláknami rozumne pracovať, existuje v CUDA organizačná štruktúra, ktorú môžeme vidieť znázornenú na Obr. 4.3. V CUDA sú vlákna rozdelené do blokov. Každé vlákno má v bloku svoje unikátne súradnice, ktoré môžu byť 1, 2 alebo 3-rozmerné. Veľkosť blokov je obmedzená, záleží od používanej GPU architektúry. Bloky sú usporiadané v mriežke. Každý blok má v mriežke svoje unikátne súradnice. Mriežka môže byť 1, 2 alebo 3-rozmerná. Vždy, keď spúšťame kernel, musíme mu nastaviť rozmery bloku a mriežky, na ktorej sa má spustiť. Vlákna môžu využiť ich umiestnenie v mriežke, aby sme určili, ktoré prvky vstupnej dátovej štruktúry majú spracovávať. Funkcie, ktoré sme predtým implementovali na CPU budeme teda implementovať na GPU ako kernely, ktoré budeme spúšťať na 2D mriežkach s rozmermi prispôsobenými našim vstupným obrazom.

Každá paralelizovaná funkcia je CUDA kernelom spúšťaným na mriežke vlákien, v

ktorých bežia jej inštancie. V našej implementácii bude každý pixel vstupného obrazu spracovávaný jedným vláknom. Mriežka pre naše funkcie bude teda 2-rozmerná a bude mať počet vlákien v riadku a stĺpci rovný počtu pixelov v riadku a stĺpci vstupného obrazu. Vlákna sú v mriežke rozdelené do blokov, ktoré majú GPU architektúrou obmedzenú veľkosť. V bloku môžu mať 1, 2 alebo 3-rozmerné súradnice. Vstupný obraz potrebujeme rozdeliť do blokov tak, aby sa nám s pixelmi vo vláknach dalo rozumne pracovať. My sme si určili, že bloky v našich mriežkach budú jednorozmerné. Vlákna v jednom bloku teda budú spracovávať vždy časť jedného riadku alebo stĺpca vstupného obrazu. Toto rozdelenie sme zvolili lebo vo funkciách budeme potrebovať robiť operácie na riadkoch a stĺpcoch, nie len na jednotlivých pixeloch.

Keďže každý pixel vstupného obrazu je spracovávaný jedným vláknom mriežky danej CUDA kernel funkcie, potrebujeme každému vláknu daný pixel jednoznačne priradiť. Vlákno má dostupné informácie o mriežke v ktorej sa nachádza. CUDA vláknam ponúka prístup k nasledujúcim premenným: $gridDim$ - rozmery mriežky v blokoch, $blockDim$ - rozmery bloku vo vláknach, $blockIdx$ - súradnice bloku v mriežke, v ktorom sa dané vlákno nachádza a $threadIdx$ - súradnice vlákna v bloku. Každá z týchto premenných je typu $dim3$, čo je 3-rozmerný vektor celých čísel (má parametre x, y, z). Ak sa niektorý rozmer v danej premennej nepoužíva, tak je jeho hodnota v tomto vektore 1. Keď chceme priradiť danému vláknu súradnice pixela vstupného obrazu, ktorý má spracovávať, stačí pre každú súradnicu urobiť jednoduchý výpočet. Tento výpočet si ukážeme na X-ovej súradnici pixela vstupného obrazu:

$$x = blockIdx.x * blockDim.x + threadIdx.x. \tag{4.3}$$

Y-ovú súradnicu pixela vieme dostať analogickým spôsobom. Takto získané súradnice jednoznačne určia prvok vstupnej matice, ktorý má vlákno spracovávať.

Keď spúšťame kernel na nejakej mriežke, chceme, aby každé vlákno spracovalo nejakú časť vstupných dát. Otázkou je, ako tieto dáta kernelu posunúť. Pamäte CPU a GPU sú oddelené. GPU má viac typov pamätí s inými vlastnosťami, do ktorých sa dajú ukladať rôzne množstvá dát. Prvá vec, ktorú treba pri spracovávaní dát na GPU urobiť, je nahrať tieto dáta do globálnej pamäte GPU. Keď máme dáta v globálnej pamäti, vedia k nim pristupovať všetky bežiace vlákna. Vstupné dáta vieme vláknu posunúť napríklad tak, že pridáme do parametrov kernelu smerník ukazujúci na ne. Okrem globálnej pamäte, do ktorej majú prístup všetky vlákna, existuje zdieľaná pamäť, ktorá je lokálnou pamäťou pre každý blok. Túto pamäť majú spoločnú vlákna, ktoré patria do toho istého bloku mriežky. Ďalším typom pamäte sú registre, ktoré má každé vlákno svoje a nezdieľa ich so žiadnym iným vláknom. Tento typ pamäte je najrýchlejší. Ešte existuje konštantná a textúrová pamäť, ktoré používajú cache, a preto sú rýchlejšie ako globálna. Nebudeme ich však viac rozoberať lebo v našej implementácii budeme používať len globálnu, zdieľanú a registrovú pamäť. Vstupná matica, na ktorú aplikujeme filter je

nahratá v globálnej pamäti GPU. Prístup do globálnej pamäte je pomalý. Aby sme pri niektorých funkciách na maticiach nemuseli pristupovať do globálnej pamäte, načítame si pred behom niektorých funkcií časti matice do zdieľanej pamäte, ktorá je podľa výrobcu približne 100-krát rýchlejšia [10]. Potom počítame na dátach v zdieľanej pamäti. Keď máme výsledok, nakopírujeme ho späť do matice v globálnej pamäti. Zdieľaná pamäť je rýchla len vtedy, keď sa s ňou pracuje správnym spôsobom. Je reprezentovaná ako pole, ktorého prvky sú v našej implementácii float hodnoty. Správnym indexovaním prvkov vieme maximalizovať počet prístupov k poľu, ktoré sa vykonávajú paralelne. Spôsob indexovania prvkov v zdieľanej pamäti sme prebrali z publikácie [6], ktorá sa danej téme venuje podrobnejšie.

4.3.3 NVIDIA Jetson TX2

Pre spúšťanie paralelnej implementácie sme potrebovali vybrať zariadenie, ktoré disponuje CUDA GPU jadrami. Rozhodli sme sa pre NVIDA Jetson TX2 [9]. Toto zariadenie má dve jednotky SM (Streaming Multiprocesor) pozostávajúce zo štyroch blokov po 32 CUDA GPU jadrách, čo je dokopy 256 CUDA GPU jadier [9]. Na jednotkách SM bežia paralelne vlákna viacerých blokov mriežky aktuálne spusteného kernela. Vlákien môže bežať súčasne aj niekoľko tisíc. Koľko presne sa ich vykonáva naraz, je určené efektivitou implementácie. GPU jadrá sú architektúry NVIDIA Pascal. Dôležitou informáciou pre našu implementáciu je to, že spomínaná GPU architektúra ponúka veľkosť bloku mriežky kernela maximálne 1024 vlákien, čo je dvojnásobne viac ako podporovali staršie architektúry [10]. Jetson nám ponúka všetky potrebné nástroje pre paralelizáciu. Toto zariadenie sme vybrali aj kvôli tomu, že náš filter má byť zaradený do postupnosti iných filtrov, ktoré spracovávajú dáta z PhoXi 3D skenera [12]. Táto postupnosť filtrov funguje práve na zariadení NVIDIA Jetson TX2 [9]. Výstupom z práce má byť filter zaraditeľný do tejto postupnosti, takže je vhodné tomu našu implementáciu prispôsobiť.

4.4 GPU Implementácia guided filtra

Funkcie, ktoré budeme paralelizovať sú aritmetické operácie na maticiach a funkcia $f_{priemer}$, ktorá sa dá odvodiť pomocou operácie delenia matíc a funkcie f_{sucet} zloženej z funkcií $f_{integral}$ a f_{okolie}. Funkcia f_{pocet} je len špeciálnym prípadom funkcie f_{sucet}. V tejto podkapitole opíšeme implementáciu týchto funkcií ako CUDA kernelov, ktoré budeme spúšťať paralelne na našich vstupných obrazoch. Podobne ako pri CPU implementácii budeme vstupný obraz rozdelovať do blokov kvôli eliminácii kumulovaných chýb hodnôt pixelov spôsobených vytváraním integrálneho obrazu. Čím je obraz, ktorého funkciu f_{sucet} počítame menší, tým menšia je nakumulovaná chyba hodnôt jeho pixelov. Paralelizácia nám sama o sebe rozdeľuje obraz do mriežky blokov. Tento fakt využijeme

a rozdelenie vlákien do blokov v mriežke kernela bude pri našej GPU implementácii guided filtra ekvivalentné rozdeleniu pixelov do blokov obrazu. Guided filter, a teda aj funkciu f_{sucet} budeme spúšťať samostatne na blokoch, nie na celom vstupnom obraze.

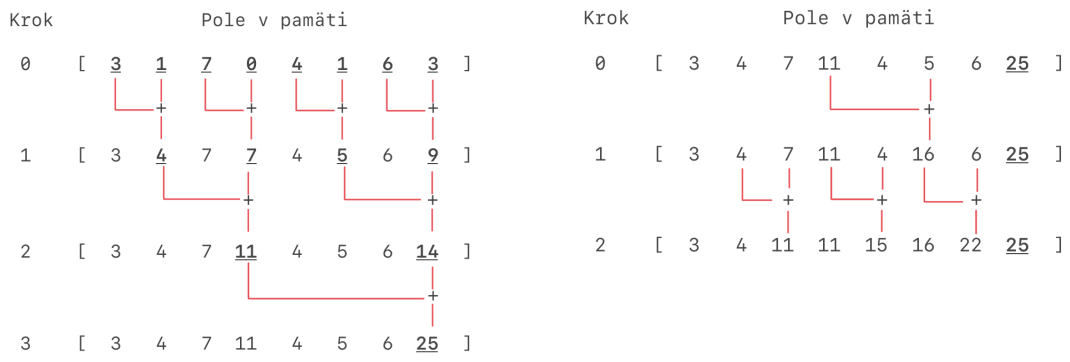
4.4.1 Implementácia aritmetických operácií

Implementácia GPU CUDA kernela, ktorý na maticiach vykonáva aritmetické operácie je triviálna. Na vstupe dostaneme smerníky na dve vstupné a jednu výstupnú maticu v globálnej pamäti GPU. Každé vlákno mriežky totožnej s rozmermi matíc, na ktorej sa bude kernel spúšťať, spracováva vždy jeden ich prvok. Najskôr si v danom vlákne odvodíme súradnice prvku matice, ktorému je toto vlákno priradené. Zo vstupných matíc na týchto súradniciach načítame hodnoty daných prvkov, vykonáme na nich aritmetickú operáciu a výsledok zapíšeme do výslednej matice na tie isté súradnice. Pri operácii delenia okrem toho skontrolujeme, či nie je menovateľ nulový. V tom prípade by sme len do výstupnej matice zapísali na dané súradnice hodnotu 0. Pri takejto implementácii nevytvoríme žiadne konflikty pri zapisovaní alebo čítaní prvkov z matíc, lebo každému prvku matice je pridelené práve jedno vlákno mriežky, na ktorej je kernel spustený. Pri dostatočnom množstve GPU jadier by takto implementované aritmetické operácie na maticiach mali mať konštantnú časovú zložitosť, lebo pre každý prvok matice robíme konštantný počet operácii a tieto operácie sa pre všetky prvky robia paralelne. Oproti lineárnej časovej zložitosti CPU implementácie dosahujeme takouto paralelizáciou rádovo lepší časový výsledok.

4.4.2 Implementácia funkcie $f_{priemer}$

Funkciu $f_{priemer}$ vieme odvodiť pomocou operácie delenia matíc a funkcie f_{sucet}, ktorú získame zložením funkcií $f_{integral}$ a f_{okolie}. Budeme teda implementovať tieto dve funkcie. Implementácia funkcie f_{sucet} bude potom vyzerať tak, že najskôr zavoláme funkciu $f_{integral}$ na vstupný obraz a výstupný obraz tejto funkcie dáme na vstup funkcii f_{okolie}. Výstupný obraz funkcie f_{okolie} bude aj výstupným obrazom funkcie f_{sucet}.

Pri paralelnej GPU implementácii funkcie $f_{integral}$ zachováme základný princíp jej fungovania CPU implementácie. Integrálny obraz vstupného obrazu vytvoríme tak, že urobíme kumulatívne sumy na všetkých jeho riadkoch a na výstupnej matici kumulatívnych súm po riadkoch urobíme kumulatívne sumy na všetkých jej stĺpcoch. Výsledkom takéhoto procesu je integrálny obraz. Keď chceme tento proces paralelizovať, musíme paralelizovať proces vytvárania kumulatívnej sumy. V CPU implementácii sme pole prechádzali po prvku a v pomocnej premennej sme si kumulovali súčet predošlých prvkov poľa. Pri takomto prístupe potrebujeme pre výpočet každého prvku poľa poznať vypočítanú hodnotu predošlého prvku. Takto implementovaný výpočet kumulatívnej sumy nevieme dobre paralelizovať. Na vytvorenie kumulatívnej sumy poľa dĺžky n



Obr. 4.4: Prvá (vľavo) a druhá (vpravo) fáza algoritmu prefixových súm.

prvkov potrebujeme $n - 1$ operácií sčítania. Chceli by sme nájsť spôsob, ako čo najviac operácií robiť naraz. Pre riešenie tohto paralelizačného problému sme sa rozhodli použiť algoritmus prefixových súm [1]. Základnou myšlienkou algoritmu je vytvoriť nad poľom vyvážený binárny strom. Algoritmus nevytvára nad poľom žiadne dátové štruktúry navyše. Binárny vyvážený strom je len abstrakcia, z ktorej vieme odvodiť, kedy a na ktorých prvkoch majú vlákna robiť operácie. Algoritmus má dve fázy, v ktorých tento strom prechádza. Najskôr smerom od listov ku koreňu stromu a potom opačne. Pritom využíva abstrakciu stromu nad poľom a prepisuje jeho prvky. Na Obr. 4.4 ukážeme príklad obidvoch fáz algoritmu.

V prvej fáze algoritmu sa získavajú čiastočné súčty prvkov poľa a ukladajú sa na indexy odvodené z binárneho vyváženého stromu. V druhej fáze sa zo získaných čiastočných súm dopočítajú hodnoty ešte nevypočítaných prvkov poľa. Pri obidvoch fázach prepisujeme prvky zvolené na základe abstrakcie binárneho vyváženého stromu. Každý znak + na Obr. 4.4 predstavuje jedno vlákno. Operácie sčítania, ktoré sú vo fáze algoritmu na rovnakej úrovni sa vykonávajú paralelne, je však nutné, aby vlákna pri každom kroku počkali, kým nie sú všetky hodnoty zapísané do poľa. Pri samotnej implementácii algoritmu sme použili jeho verziu uvedenú vo verejne dostupnej prezentácii z prednášky [3] o algoritme prefixových súm.

Výpočet kumulatívnej sumy, ktorý sme implementovali pomocou algoritmu prefixových súm pozostáva z dvoch prechodov po binárnom vyváženom strome. Prvý krát od listov ku koreňu a druhý krát od koreňa k listom. Jednotlivé operácie sčítania, ktoré sa vykonávajú v jednom kroku algoritmu by sa pri dostatočnom počte GPU jadier mali vykonávať paralelne. Krokov algoritmu je pri každom prechode stromu pri m prvkoch poľa $log\ m$. Prechody po strome sú dva, čiže krokov algoritmu je $2 \cdot log\ m$.

Vo funkcii $f_{integral}$ robíme kumulatívne sumy najskôr na riadkoch a potom na stĺpcoch bloku obrazu. Pri dostatočnom počte GPU jadier by sa mali všetky kumulatívne sumy počítať paralelne, čiže by počet krokov vytvárania integrálneho obrazu mal byť

dvojnásobne väčší ako počet krokov pri vytváraní kumulatívnej sumy, teda maximálne $4 \cdot log\ m$, kde m je počet prvkov riadku alebo stĺpca, ktorý blok spracováva. Výsledná časová zložitosť nami paralelne implementovanej funkcie $f_{integral}$ je $O(log\ m)$, čo je oproti lineárnej časovej zložitosti na CPU výrazné lepší výsledok.

Teraz opíšeme GPU implementáciu funkcie f_{okolie}. Funkcia f_{okolie} z integrálneho obrazu vypočíta pre každý pixel vstupného obrazu súčty hodnôt z jeho štvorcového okolia, ktorého polomer dostaneme na vstupe. Pri GPU paralelnej implementácii budeme využívať ten istý algoritmus ako v CPU implementácii. V CPU implementácii má výpočet tejto matice dve fázy, v ktorých prechádzame najskôr integrálny obraz po riadkoch a potom výsledý obraz prvého prechodu po stĺpcoch. Pri prvom prechode získa každý pixel so súradnicami (x, y) hodnotu rovnú rozdielu prvkov integrálneho obrazu $(x + r, y)$ a $(x - r - 1, y)$. Pri druhom prechode získame výslednú hodnotu pixela so súradnicami (x, y) rozdielom prvkov výslednej matice prvého prechodu so súradnicami $(x, y + r)$ a $(x, y - r - 1)$. Na CPU pri každom prechode cez maticu pristupujeme k dvom jej prvkom, ktoré od seba odčítame. Tomu sa pri GPU paralelnej implementácii musíme vyhnúť. Ak by dve vlákna skúšali pristúpiť k rovnakému prvku naraz, vznikol by konflikt. Potrebujeme, aby jedno vlákno pristupovalo vždy k práve jednému unikátnemu prvku matice. Rozdiel prvkov sa dá rozdeliť na dva kroky. V prvom kroku načítame prvý prvok do našej výslednej matice. V druhom kroku načítame druhý prvok a odčítame ho od prvku, ktorý máme v matici uložený z prvého kroku.

Každý rozdiel prvkov pri prechodoch matíc teda rozdelíme na dva kroky. Vysvetlíme si to na prvom prechode algoritmu, druhý bude fungovať analogicky. V prvom kroku načítame do prvku matice so súradnicami (x, y) prvok integrálneho obrazu so súradnicami $(x + r, y)$. V druhom kroku odčítame od tohto prvku hodnotu prvku z integrálneho obrazu so súradnicami $(x - r - 1, y)$. Pri každom z opísaných krokov na prvku matice so súradnicami (x, y) bol načítaný práve jeden unikátny prvok z integrálneho obrazu. Tieto dva kroky budeme robiť pri každom prechode sériovo jeden po druhom. Vrámci jedného kroku však môžeme operácie robiť na všetkých prvkoch matice paralelne. Pri dostatočnom množstve GPU jadier majú tieto kroky konštantnú časovú zložitosť.

Funkcia f_{okolie} pozostáva z dvoch prechodov matíc rozmerov integrálneho obrazu na jej vstupe. Každý z týchto prechodov má dva kroky, ktoré musia byť vykonané sériovo. Kroky majú pri dostatočnom množstve GPU jadier konštantnú časovú zložitosť. Tým pádom má pri dostatočnom množstve GPU jadier aj funkcia f_{okolie} konštantnú časovú zložitosť, čo je oproti lineárnej časovej zložitosti implementácie tejto funkcie na CPU výrazné zlepšenie.

4.4.3 Zhrnutie GPU implementácie guided filtra

Opisovali sme paralelnú GPU implementáciu funkcií, ktoré sa používajú v krokoch algoritmu guided filtra (Algoritmus 1). Implementovali sme aritmetické operácie na maticiach a funkciu f_{sucet} zloženú z funkcií $f_{integral}$ a f_{okolie}. Pomocou funkcie f_{sucet} a operácie delenia matíc vieme vyskladať funkciu $f_{priemer}$. Kvôli eliminácii numerických chýb pri vytváraní integrálneho obrazu rozdeľujeme vstupný obraz do blokov, na ktorých guided filter spúšťame samostatne.

Aritmetické operácie na maticiach majú v našej implementácii pri dostatočnom množstve GPU jadier konštantnú časovú zložitosť. Nami implementovaná funkcia $f_{integral}$ má časovú zložitosť $O(log\ m)$, kde m je počet pixelov v riadku štvorcového bloku vstupného obrazu, na ktorý sa funkcia aplikuje. Pri dostatočnom množstve GPU jadier je to aj celková časová zložitosť tejto funkcie na celom vstupnom obraze. Funkcia f_{okolie} má pri dostatočnom množstve GPU jadier konštantnú časovú zložitosť. Tým pádom má celkovo funkcia f_{sucet} a aj $f_{priemer}$ pri dostatočnom množstve GPU jadier časovú zložitosť $O(log\ m)$, kde m je počet pixelov v riadku štvorcového bloku vstupného obrazu, na ktorý sa funkcia aplikuje. Keďže guided filter vo svojom algoritme funkciu $f_{priemer}$ a aritmetické operácie volá len konštantne veľa krát, má naša paralelná implementácia algoritmu guided filtra celkovú časovú zložitosť $O(log\ m)$, kde m je počet pixelov v riadku štvorcového bloku vstupného obrazu, na ktorý sa guided filter aplikuje.

Uvedená časová zložitosť je však len teoretickým odhadom. Reálne na našom zariadení nebeží naraz toľko vlákien, koľko máme pixelov vstupného obrazu. To, koľko vlákien kedy beží naraz nevieme jedoducho určiť. Záleží to od implementačných detailov jednotlivých funkcií a vnútornej organizácie GPU jednotiek nášho zariadenia. Ďalším faktom je, že pred spúšťaním celého paralelizovaného algoritmu guided filtra na GPU je nutné skopírovať vstupné obrazy do globálnej pamäte GPU. Tým sme sa dostali na lineárnu časovú zložitosť od počtu pixelov vstupného obrazu. Ak však odhliadneme od kopírovania vstupných dát do globálnej pamäte GPU, tak sme čas behu algoritmu guided filtra oproti CPU implementácii rádovo zlepšili.

4.5 GPU Implementácia úpravy intenzitnej textúry

Úpravu intenzitnej textúry paralelizovať nebudeme. Od začiatku nebola v cieľoch tejto práce. Po analýze vstupných dát však vyvstala potreba textúru pred behom algoritmu upraviť, aby spĺňala predpoklady filtra. Pred spustením guided filtra vždy textúru upravíme na CPU. Do budúcnosti by bolo pre nasadenie na zariadení vhodné tento proces paralelizovať, nie je to však predmetom tejto práce.

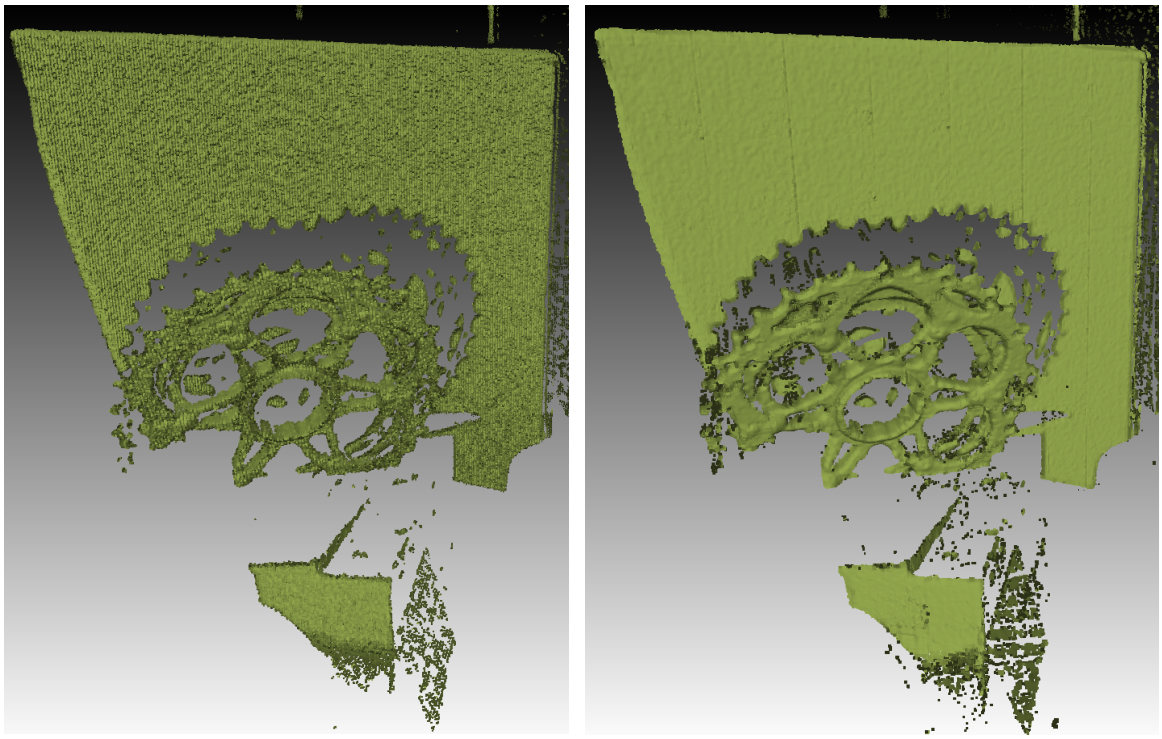
Kapitola 5

Výsledky a evaluácia filtrovania

V tejto kapitole si rozoberieme výsledky, ku ktorým sme sa v práci dopracovali. Je viac typov výsledkov, ktoré budeme analyzovať. Najsôr sa pozrieme, ako sme boli úspešní pri eliminácii šumu vo vstupných obrazoch, potom preskúmame efektivitu našich CPU a GPU implementácií a nakoniec porovnáme výsledky, ktoré sme dosiahli s inými filtrovacími technikami. Na generovanie obrázkov a analýzu dát budeme používať voľne dostupný softvér ImageJ [14].

5.1 Eliminácia šumu

V tejto podkapitole sa budeme zaoberať typmi šumu z Tab. 3.1, ktoré na základe našej analýzy vieme pomocou guided filtra odstrániť. Keďže naše vstupné intenzitné textúry nie sú ideálne a obsahujú chyby, ktoré pri ich úprave nevieme úplne odstrániť, napr. výrazný lesk, guided filter zachováva hrany objektov lepšie, keď sa spúšťa s menším kernelom. Pri väčších kerneloch sa časti obrazu, ktoré sú v intenzitnej textúre výrazne pokazené leskom môžu na hranách jemne rozmazať. Najlepšie výsledky sme dosiahli s polomerom kernelu $r = 2$. Regularizáciu ϵ určujeme pre hĺbkové mapy experimentálne. Pre demonštráciu výsledkov sme vybrali konkrétnu hĺbkovú mapu naskenovanej scény vypočítanú skenerom, upravili sme jej intenzitnú textúru a spustili sme nami implementovaný guided filter s polomerom kernela $r = 2$ a regularizáciou $\epsilon = 500$. Aby sme ukázali celkové výsledky filtrovania, zobrazili sme pomocou softvéru Meshlab [13] hĺbkové mapy ako mračná bodov v 3D priestore. Na Obr. 5.1 vidieť, že povrch objektov vo výstupnom mračne bodov z nášho filtrovania je oproti vstupnému mračnu bodov vyhladený a hrany sa zachovali s určitou presnosťou, čo sme chceli v konečnom dôsledku dosiahnuť. Niektoré sú minimálne rozmazané. Isté chybné oblasti intenzitnej textúry naša úprava nedokázala eliminovať. Väčšinu hrán sme však upravili s uspokojivou presnosťou. Výsledný obraz z Obr. 5.1 môžeme porovnať s výsledným obrazom z Obr. 5.2. Na tomto obrázku nájdeme výsledok filtrovania, pred ktorým sme

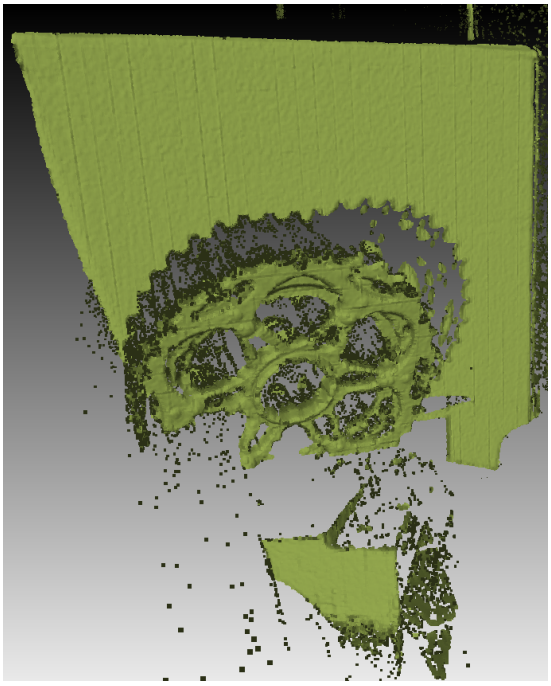


Obr. 5.1: Časť mračna bodov vytvoreného zo vstupnej (vľavo) a výstupnej (vpravo) hĺbkovej mapy filtrovania.

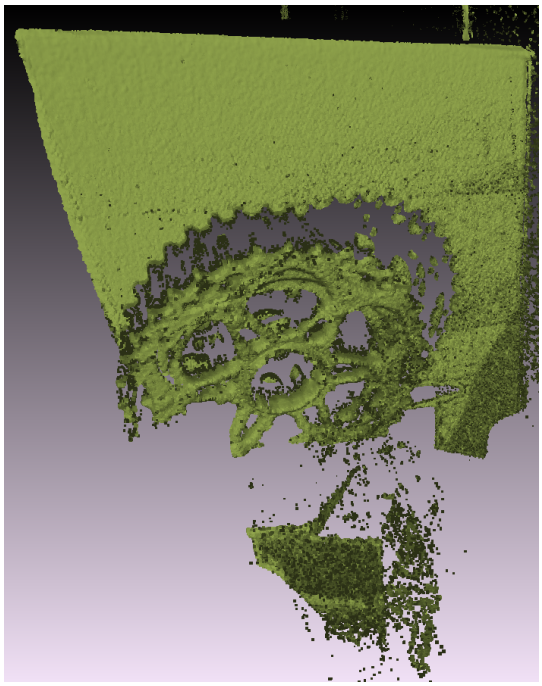
intenzitnú textúru neupravili. Oproti výstupnému mračnu bodov z Obr. 5.1 sú v mračne bodov na Obr. 5.2 výrazne rozmazanejšie hrany objektov, lebo naša úprava intenzitnej textúry poskytla guided filtru pomocný guidance obraz s ostrejšími hranami.

Malou pozorovateľnou chybou vo výstupnom mračne bodov na Obr. 5.1 sú drobné nepresnosti na hraniciach blokov, na ktorých filter spúšťame samostatne. Táto chyba je však menšia ako nepresnosť získaná numerickými chybami pri spúšťaní filtra na celom obraze, ktorého výsledné mračno bodov sme pre porovnanie zobrazili na Obr. 5.3. Vidíme, že čím sú body v mračne bodov na Obr. 5.3 viac vpravo a dole, tým je ich pozícia menej presná a povrch objektov je výrazne zašumený. Hranice blokov sú zjavne oveľa menšou chybou výstupného obrazu.

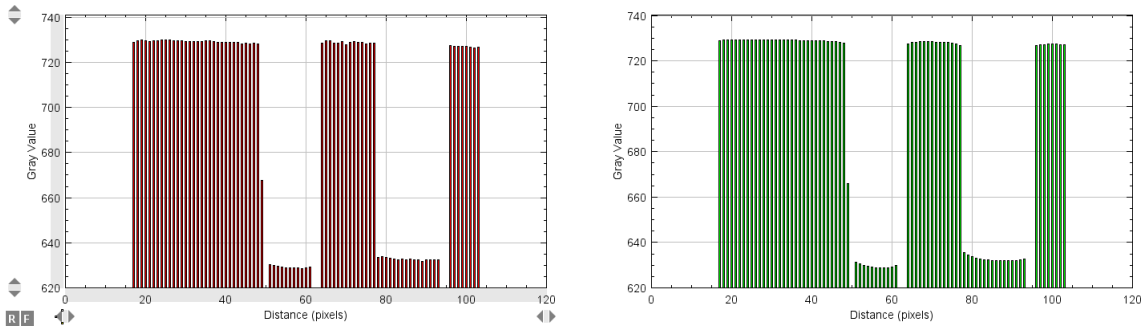
Keď sme ukázali celkové výsledky filtrovania, pozrieme sa na konkrétne časti obrazu a budeme analyzovať, ako sme eliminovali šumy z Tab. 3.1. Podstatou guided filtra je vyhladiť povrch objektov scény, a pritom zachovať ich hrany. Ako na prvý šum sa teda pozrieme na nerovnosti plôch v hĺbkových mapách. Pomocou ImageJ [14] sme vyselektovali tú istú úsečku pixelov zo vstupnej hĺbkovej mapy a výstupnej hĺbkovej mapy z guided filtra. Zvolili sme úsečku pixelov, ktorá prechádza cez viac hrán aby sme ukázali, že ich filter nerozmaže. Grafy vygenerované z nami zvolenej úsečky môžeme vidieť na Obr. 5.4. Tieto grafy ukazujú, že výstupný obraz nie je oproti vstupnému na hranách rozmazaný. Aby sme ukázali, že pritom filter vyhladil zašumené plochy, potrebujeme sa pozrieť na hodnoty pixelov detailnejšie. Grafy na Obr. 5.5 sú priblížením



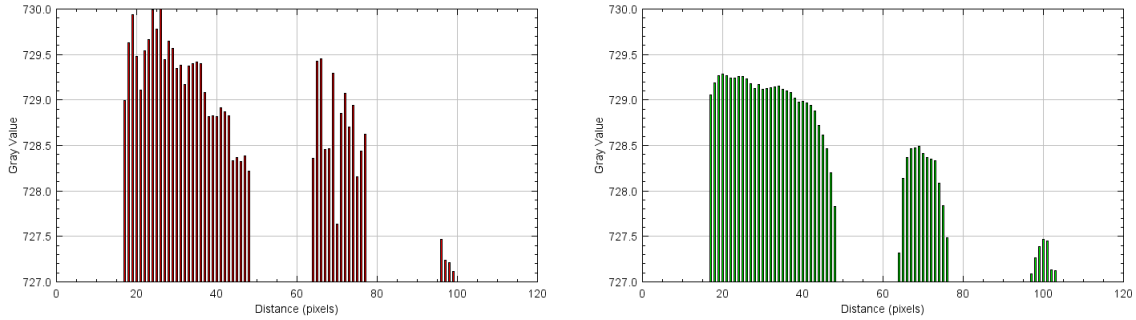
Obr. 5.2: Časť mračna bodov vypočítaného z výstupnej hĺbkovej mapy, kedy sme pred spustením guided filtra neupravili intenzitnú textúru.



Obr. 5.3: Časť mračna bodov vytvoreného z výstupnej hĺbkovej mapy, kedy sme guided filter spustili na celom obraze a nie jednotlivo na samostatných blokoch.



Obr. 5.4: Úsečka pixelov vstupnej (vľavo) a výstupnej (vpravo) hĺbkovej mapy filtrovania, pretínajúca hrany. V grafoch možno pozorovať, že sa pri filtrovaní hrany, cez ktoré úsečka prechádza hĺbkovej mapy zachovali.

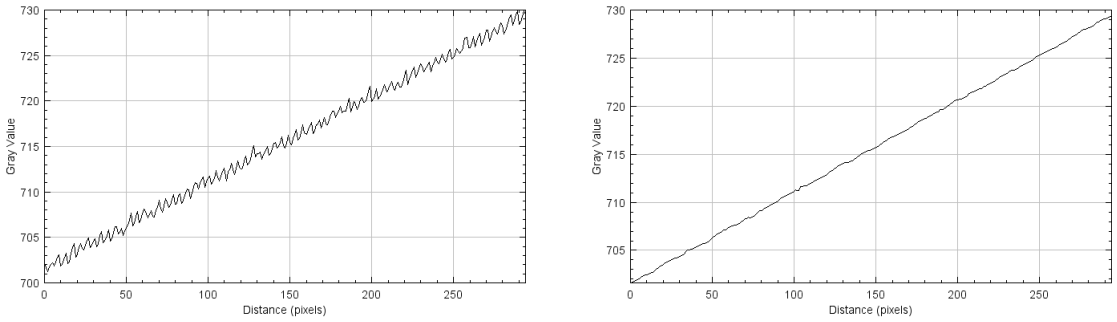


Obr. 5.5: Priblíženie hodnôt pixelov úsečky vstupnej (vľavo) a výstupnej (vpravo) hĺbkovej mapy filtrovania, pretínajúca hrany. V grafoch možno pozorovať, že sa pri filtrovaní hrany hĺbkovej mapy povrch objektu, cez ktorý úsečka prechádza, vyhladil.

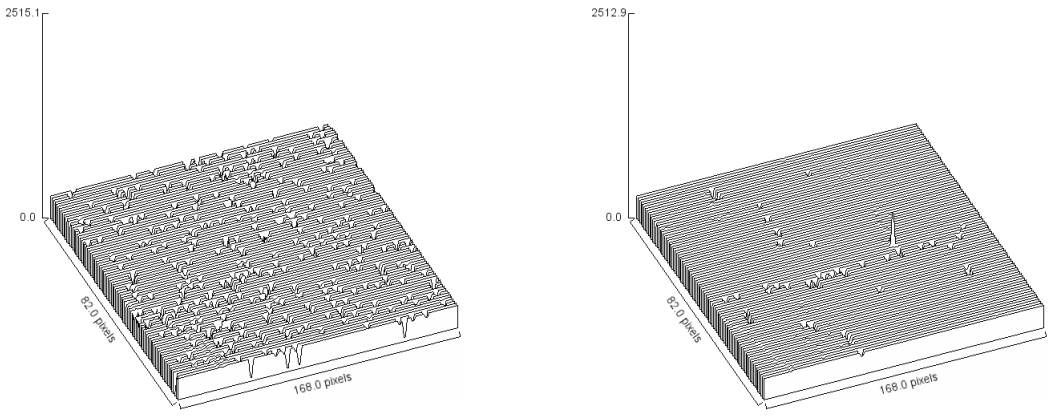
hornej časti grafov z Obr. 5.4. Vo vstupnej hĺbkovej mape vidíme, že plocha, ktorou úsečka v obraze prechádza je zašumená. Vo výstupnej mape sa tento šum použitím guided filtra výrazne eliminoval a hodnoty pixelov sú stálejšie. Na našej testovacej vzorke obrazu teda nami implementovaný guided filter vyhladil malé nerovnosti na povrchu objektov v scéne, ako ukazuje Obr. 5.5 a nerozmazal pritom hrany, čo vidíme na Obr. 5.4.

Ďalším typom šumu, ktorý je nerovnostiam plôch veľmi podobný, sú artefakty laserovej projekcie skenera. Sú to v realite neexistujúce vrúbky, ktorými je pokrytý povrch objektov v scéne. Ako testovaciu vzorku pre tento typ šumu sme vybrali úsečku pixelov prechádzajúcu rovnou plochou v scéne. Na Obr. 5.6 vidíme porovnanie grafov hodnôt pixelov tejto úsečky zo vstupnej a výstupnej hĺbkovej mapy. Vo vstupnej hĺbkovej mape je jasne vidieť vrúbky, ktoré plocha obsahuje. Vo výstupnom obraze sú tieto vrúbky vyhladené. Nami implementovaný guided filter teda na tejto testovacej vzorke obrazu eliminoval artefakty laserovej projekcie skenera.

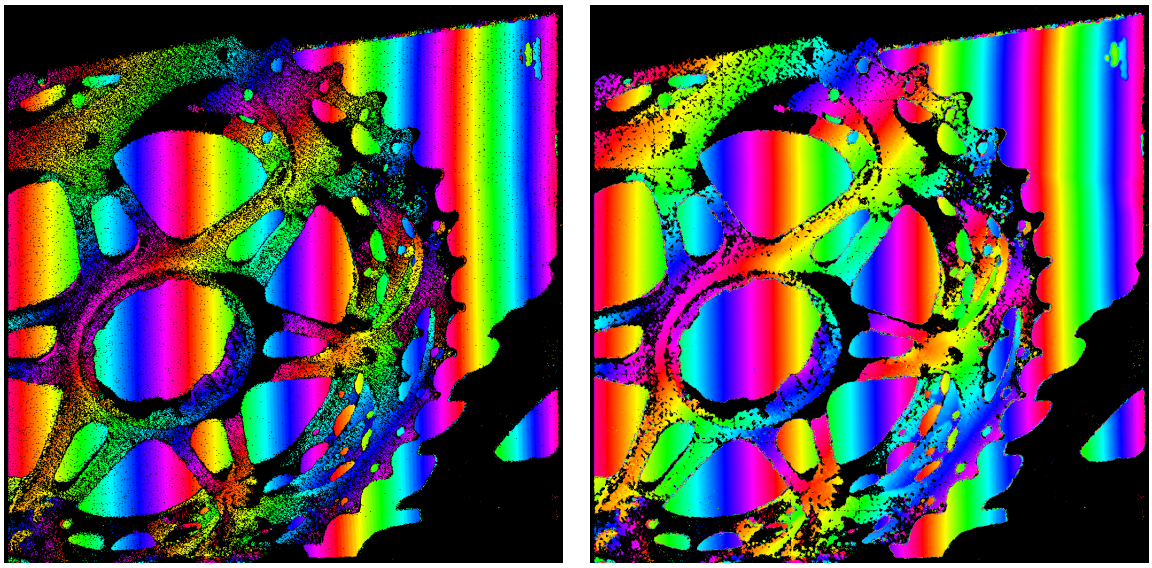
Pri analýze šumu vstupných dát sme prišli na to, že guided filter má jednu veľmi dobrú vlastnosť. Dokážeme pomocou neho s istou presnosťou dodefinovať nedefinované



Obr. 5.6: Úsečka pixelov vstupnej (vľavo) a výstupnej (vpravo) hĺbkovej mapy filtrovania prechádzajúca cez rovnú plochu.



Obr. 5.7: Časť rovnej plochy zo vstupnej (vľavo) a výstupnej (vpravo) hĺbkovej mapy filtrovania. Môžeme tu pozorovať, že sa pomocou guided filtra na tejto časti plochy podarilo dodefinovať pomerne veľa nedefinovaných pixelov.



Obr. 5.8: Časť vstupnej (vľavo) a výstupnej (vpravo) hĺbkovej mapy filtrovania zobrazenej pomocou lineárneho mapovania hodnôt pixelov na farebné spektrum. Čierne pixely sú v týchto hĺbkových mapách nedefinované. Môžeme tu pozorovať, že sa pomocou guided filtra podarilo dodefinovať pomerne veľa nedefinovaných pixelov.

pixely hĺbkovej mapy. Čím viac pixelov je v okolí nejakého nedefinovaného pixela definovaných, tým presnejšie ho vie guided filter dodefinovať. Samozrejme, tento pixel musí mať v intenzitnej textúre správnu hodnotu. Na demonštráciu tejto funkcionality filtra sme pomocou ImageJ [14] vygenerovali 3D zobrazenie časti hĺbkovej mapy, ktorá obsahuje plochu. Tieto zobrazenia môžeme vidieť na Obr. 5.7. Plochu sme vybrali, lebo na ploche majú pixely v intenzitnej textúre spravidla dobré hodnoty a tým pádom sa dajú pomocou guided filtra dodefinovať. Pre každý nedefinovaný pixel sme pri filtrovaní nastavili, že sa dodefinuje len ak je v jeho kerneli aspoň 75% pixelov definovaných, aby sme dosiahli uspokojivú kvalitu výsledku. Obr. 5.7 ukazuje, že vo výslednom obraze sa nám podarilo s rozumnou presnosťou dodefinovať viac pixelov. Niektoré pixely zostali, samozrejme, nedefinované. Dodefinovanie pixelov na väčšej časti hĺbkovej mapy prezentujeme na Obr. 5.8. Na tomto obrázku sú hĺbkové mapy zobrazené pomocou farebného spekra, na ktoré sú hodnoty ich pixelov lineárne namapované. Čierne pixely na Obr. 5.8 sú nedefinované. Na Obr. 5.8 môžeme pozorovať, že veľa čiernych pixelov zo vstupnej hĺbkovej mapy má vo výstupnej hĺbkovej mape filtrovania s pomerne dobrou presnosťou dodefinovanú farbu. Prevodové koleso v scéne je vo výstupnej hĺbkovej mape celkovo lepšie rozoznateľné. Dodefinovalo sa aj veľa pixelov na rovnej stene v pozadí scény.

Percento nutných definovaných pixelov v kerneli sa dá nastavovať. Používateľ môže na úkor kvality výsledku skúšať dodefinovať viac pixelov. Spravidla sa pixely na plochách dodefinujú dobre. Pixely, ktoré sú blízko hrán nevie guided filter dobre

Číslo behu	CPU čas v ms	GPU čas v ms
1	9194.350	7.057
2	9238.384	9.093
3	9154.863	8.819
4	9205.231	6.869
Priemer	9198.207	7.960

Tabuľka 5.1: Časy behu CPU a GPU implementácii guided filtra na zariadení NVIDIA Jetson TX2 pri rozlíšení vstupnej hĺbkovej mapy 2064×1544 pixelov.

dodefinovať, lebo sú v intenzitnej textúre nepresné. Úprava intenzitnej textúry upravuje len pixely, ktoré sú v hĺbkovej mape definované. Na testovacej vzorke obrazu sme ukázali, že nami implementovaný guided filter dokáže s určitou presnosťou dodefinovať nedefinované pixely vstupnej hĺbkovej mapy.

5.2 CPU a GPU Implementácia

Hlavnou motiváciou paralelizácie CPU implementácie guided filtra bolo zvýšiť rýchlosť jeho behu. V Kap. 4 sme uviedli, že vstupné obrazy musíme pred behom programu nahrať do GPU globálnej pamäte. Nejaký čas nám zaberú aj alokácie pomocných matíc. Všetky potrebné alokácie teda v GPU implementácii vykonáme pred behom samotného algoritmu, ktorý k nim potom pristupuje. Alokáciu matíc ani nahrávanie obrazov do globálnej pamäte do merania rýchlosti našej implementácie algoritmu guided filtra nebudeme rátať. Chceme zistiť ako sa nám paralelizáciou jednotlivých funkcií podarilo zefektívniť filtrovanie našich vstupných obrazov. Pri CPU implementácii takisto nebudeme zarátavať načítavanie matíc zo súborov. Bude nás zaujímať len čas behu algoritmu. Pre porovnanie rýchlosti GPU a CPU implementácie sme ich spustili na tom istom vstupe s polomerom kernela $r = 2$ a regularizáciou $\epsilon = 500$. Veľkosť blokov obrazu, na ktorých samostatne spúšťame guided filter sme nastavili na 128×128 pixelov. Obidvom implementáciám sme merali čas behu. Výsledky tohto merania uvádzame v Tab. 5.1. Z údajov tejto tabuľky vidíme, že GPU implementácia výrazne zrýchlila beh algoritmu guided filtra. Oproti CPU verzii je GPU implementácia viac ako 1150-*krát* rýchlejšia. Tým sme splnili cieľ zrýchlenia filtrovania obrazu paralelizáciou na GPU.

5.3 Porovnanie výsledkov filtrovacích techník

Guided filter, ktorý sme implemetovali, budeme porovnávať s bilaterálnym filtrom, ktorý slúži na ten istý účel. Vyhladzuje povrchy a zachováva pritom hrany. Tento filter funguje na inom princípe a nepoužíva intenzitnú textúru. To je v prípade našich

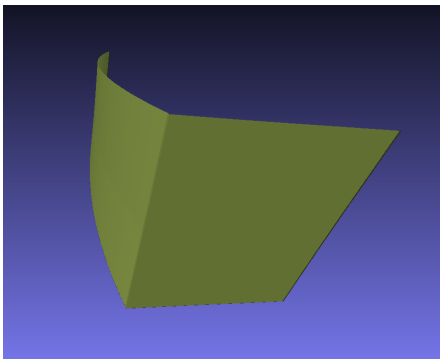
Filter	Priemerný čas v ms	Počet behov
Guided	11.859	4
Bilateral	12.937	4

Tabuľka 5.2: Časy behu našej GPU implementácie guided filtra a Photoneo [12] GPU implementácie bilateral filtra na zariadení NVIDIA Jetson TX2 pri rozlíšení vstupnej hĺbkovej mapy 2064×1544 pixelov.

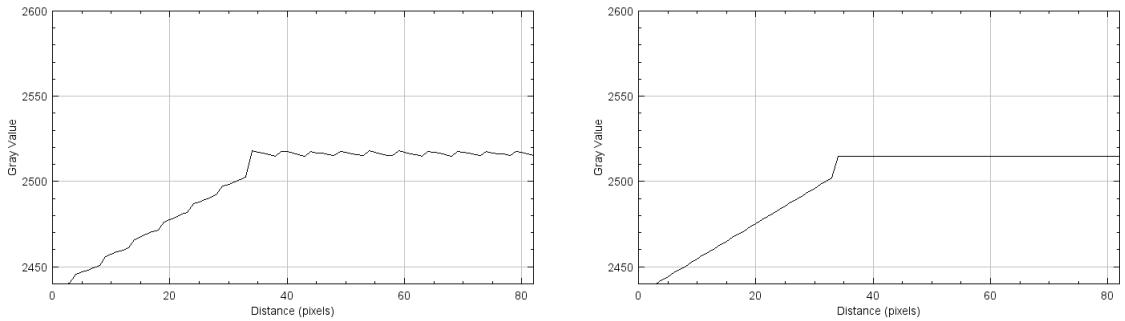
vstupných dát výhodou, lebo sú nepresné a nie všetky chyby intenzitných textúr vieme dobre odstrániť. Vo všeobecnosti má bilaterálny filter presnejšie výstupné obrazy ako náš guided filter, lebo netrpí chybami intenzitných textúr. Náš výstupný obraz má o niečo horšie výsledky na hranách, ktoré sú výrazne ovplyvnené leskom v scéne. Tie môžu byť jemne rozmazané. Okrem toho vie bilaterálny filter odstrániť šum v podobe nesprávnych pixelov obklopených nedefinovanými pixelmi. Toto guided filter na našich vstupných dátach nedokáže prirodzeným spôsobom urobiť.

Vo všeobecnosti by mal byť náš guided filter oproti bilateral filtru rýchlejší, lebo má z definície lepšiu časovú zložitosť. Bilaterálny filter má pri polomere kernela r a n vstupných pixeloch časovú zložitosť $O(n(2r+1)^2)$ [16]. Čím väčší kernel používame, tým dlhšie bude bilaterálny filter bežať. Týmto guided filter netrpí, má len lineárnu časovú zložitosť od počtu vstupných pixelov [8]. Tieto odhady časovej zložitosti však nepočítajú s paralelizáciou filtrov. Filtre môžeme paralelizovať rôzne efektívne. My budeme výsledky našej paralelnej implementácie guided filtra porovnávať s výsledkami paralelnej implementácie bilaterálneho filtra, ktorá je súčasťou postupnosti filtrov od firmy Photoneo na dáta z PhoXi 3D skenera [12]. Očakávali sme, že sa rýchlosťou aspoň priblížime k rýchlosti tohto paralelne implementovaného bilaterálneho filtra. Spustili sme teda obidva filtre s rovnakým polomerom kernela $r = 4$ a výsledné časy behov uvádzame v Tab. 5.2. Podľa nameraných dát mal náš filter v priemere podobný čas behu, čo pokladáme za uspokojivý výsledok.

Hlavnou motiváciou implementácie guided filtra bola eliminácia šumu, na ktorom bilaterálny filter zlyháva. Bilaterálny filter má problémy, keď je uhol kamery a normály skenovaného povrchu objektu veľmi veľký [4]. V tejto práci sme chceli nájsť príklad scény, ktorá takýto fenomén obsahuje a ukázať, že guided filter na daných miestach nezlyhá. Guided filter funguje na inom princípe, ktorý je závislý od intenzitnej textúry, nie od normál povrchu objektov scény, preto by mal v tomto prípade fungovať. Vytvorili sme syntetickú 3D scénu pozostávajúcu z dvoch plôch. Jedna je kolmá na kameru skenera a uhol normály druhej plochy od kamery skenera je dostatočne veľký na to, aby spôsobil spomínanú chybu bilaterálneho filtra. Náhľad 3D scény, ktorú sme opísali, vidíme na Obr. 5.9. Túto scénu sme naskenovali virtuálnym skenerom, zodpovedajúcim PhoXi 3D skeneru [12]. Na povrchu plôch teda chybou skenera v scéne vznikol šum.



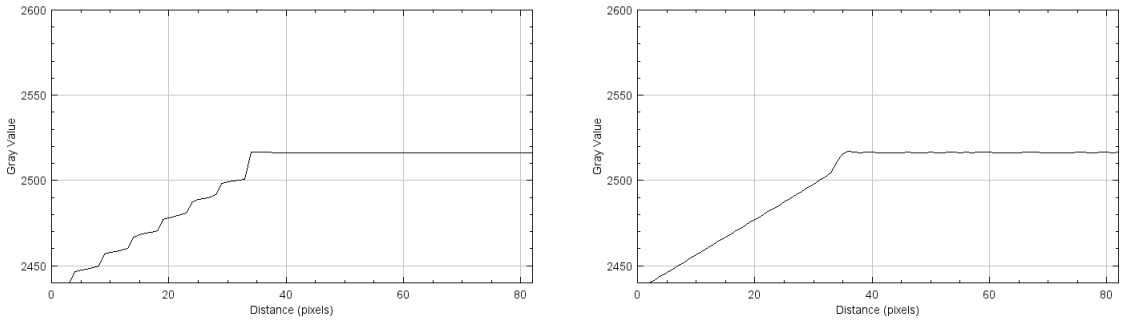
Obr. 5.9: 3D syntetická scéna, na ktorej by mal bilateral filter zlyhať.



Obr. 5.10: Úsečka pixelov pretínajúca hranu plôch zašumenej (vľavo) a ideálnej (vpravo) hĺbkovej mapy syntetickej 3D scény.

3D syntetickú scénu sme vytvorili kvôli tomu, že z nej vieme vypočítať jej ideálnu hĺbkovú mapu, ktorá zodpovedá presne realite scény. Máme teda s čím porovnať výsledky filtrovania. Na Obr. 5.10 uvádzame porovnanie ideálnej hĺbkovej mapy s hĺbkovou mapou, v ktorej pri skenovaní scény vznikol na plochách šum. Pri filtrovaní zašumenej hĺbkovej mapy by sme sa chceli teda vo výsledku čo najviac priblížiť k ideálnej hĺbkovej mape. Na naskenovanej, zašumenej hĺbkovej mape sme spustili bilaterálny a guided filter s rovnakým polomerom kernela. Z výsledných hĺbkových máp obidvoch filtrov sme pre porovnanie vyselektovali tú istú úsečku pixelov ako na Obr. 5.10. Grafy úsečky pixelov z výstupných obrazov uvádzame na Obr. 5.11.

Na Obr. 5.11 je jasne vidieť, že bilaterálny filter povrch jednej plochy nevyhladil, zatiaľ čo guided filter áno. Guided filter je teda vhodný pre použitie, keď potrebujeme vyhladiť objekty v skenovanej scéne, ktorých uhol normály povrchu od kamery skenera je veľký. Takéto časti obrazu dokáže filtrovať lepšie ako bilaterálny filter.



Obr. 5.11: Úsečka pixelov pretínajúca hranu medzi plochami výstupnej hĺbkovej mapy po použití bilateral (vľavo) a guided (vpravo) filtra.

Záver

V tejto práci sme sa venovali problematike filtrovania 3D dát. Vstupným obrazom filtrovania bola hĺbková mapa skenovanej scény vygenerovaná PhoXi 3D skenerom [12]. Výstupom filtrovania mala byť hĺbková mapa, v ktorej je povrch objektov vyhladený a objekty majú ostré hrany. Naším cieľom bolo preskúmanie využitia intenzitnej textúry ako pomocnej informácie pri filtrovaní hĺbkovej mapy. Ako našu filtrovaciu techniku sme sa rozhodli použiť guided filter [8]. Tento filter z definície používa pomocný obraz. Analyzovali sme, či naše vstupné obrazy, hĺbková mapa a intenzitná textúra spĺňajú predpoklady použitia tohto filtrovacieho prístupu. Zistili sme, že intenzitná textúra je oproti hĺbkovej mape nepresná a guided filter na pôvodné vstupné obrazy nemôžeme jednoducho aplikovať. Rozhodli sme sa preto intenzitnú textúru upraviť tak, aby spĺňala predpoklady pomocného obrazu guided filtra. Navrhli sme algoritmus úpravy intenzitnej textúry, ktorý zaostrí jej hrany pomocou presnejšej informácie, ktorú obsahuje hĺbková mapa. Na hĺbkovú mapu sme potom aplikovali guided filter a ako pomocný obraz sme použili upravenú intenzitnú textúru. Hrany výslednej hĺbkovej mapy boli pri použití upravenej intenzitnej textúry ako pomocného obrazu výrazne menej rozmazané ako pri použití neupravenej intenzitnej textúry. Guided filter sme najskôr implementovali na CPU. Pri implementácii sme zistili, že numerickou chybou pri aritmetických operáciach na *float* hodnotách, ktorá vzniká pri priemerovaní obrazu, získavame nepresný, zašumený výsledný obraz. Problém sme vyriešili tak, že sme vstupnú hĺbkovú mapu rozdelili na menšie bloky, na ktorých guided filter aplikujeme samostatne. Získali sme tým síce malú nepresnosť výsledného obrazu na hraniciach týchto blokov, ale tá je v porovnaní s chybou pri aplikácii filtra na celý obraz zanedbateľná. Keď sme mali uspokojivo presné výsledky filtrovania na CPU, paralelizovali sme našu implementáciu na GPU. Tým sme beh filtrovania viac ako 1000-*krát* zrýchlili. Výsledky našej GPU implementácie guided filtra sme porovnali s výsledkami GPU implementácie bilaterálneho filtra [16] vyvinutej firmou Photoneo na filtrovanie hĺbkových máp z PhoXi 3D skenera [12]. Oproti tejto implementácii bilaterálneho filtra sme dosiahli v priemere podobný čas behu filtra, čo je uspokojivý výsledok. Motiváciou témy tejto práce bolo zistiť, či využitím intenzitnej textúry ako pomocného obrazu vieme na hĺbkovú mapu aplikovať rýchlu, robustnú filtrovaciu techniku, ktorá vie dobre vyhladiť povrch v scéne s veľkým uhlom normály od kamery skenera. Na takýchto miestach obrazu bilaterálny filter zlyháva. Vytvorili

sme syntetickú 3D scénu, ktorá takýto povrch obsahuje a spustili sme na nej bilaterálny filter aj nami implementovaný guided filter. Bilaterálny filter povrch objektu v scéne podľa očakávania nevyhladil, pričom guided filter áno. Implementovali sme teda guided filter, ktorý je porovnateľne rýchly ako bilaterálny filter a navyše vyhladí aj povrchy, ktoré majú veľký uhol normály od kamery skenera. Bilaterálny filter má však oproti guided filtru na našich vstupných dátach výhodu v tom, že sa naň neprenáša nepresnosť intenzitnej textúry, preto vo všeobecnosti v hĺbkových mapách lepšie zachováva hrany. Ak by sme zo skenera vedeli dostať presnejšiu intenzitnú textúru, dosahoval by nami implementovaný guided filter lepšie výsledky.

V tejto práci sme implementovali paralelný GPU filter aplikovaný na štruktúrované mračná bodov, ktorý vyhladzuje povrch objektov skenovanej scény a zachováva pritom ich hrany. Ako pomocný obraz používa intenzitnú textúru scény. Na niektorých častiach obrazu vyhladzuje povrch lepšie ako bilaterálny filter [16] a pritom má podobný čas behu.

Literatúra

[1] Guy E Blelloch. Prefix sums and their applications. Technical Report CMU-CS-90-190, School of Computer Science, Carnegie Mellon University, 1990.

[2] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679--698, 1986.

[3] Yong Cao. UIUC ECE408/498AL Course Notes. Virginia Tech, 2013. Retrieved March 8, 2021, from `https://people.cs.vt.edu/yongcao/teaching/cs5234/spring2013/slides/Lecture10.pdf`.

[4] Prasun Choudhury and Jack Tumblin. The trilateral filter for high contrast images and meshes. In *Eurographics Symposium on Rendering*, pages 186--196, 2003.

[5] Franklin C Crow. Summed-area tables for texture mapping. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 207--212, 1984.

[6] Mark Harris, Shubhabrata Sengupta, and John D Owens. Parallel prefix sum (scan) with cuda. *GPU gems*, 3(39):851--876, 2007.

[7] K. He. Guided Image Filtering (Matlab Code), 2010. Retrieved May 5, 2021, from `http://kaiminghe.com/eccv10/index.html`.

[8] K. He, J. Sun, and X. Tang. Guided image filtering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(6):1397--1409, 2013.

[9] NVIDIA. Jetson TX2. Retrieved May 1, 2021, from `https://developer.nvidia.com/embedded/jetson-tx2`.

[10] NVIDIA. Cuda C Programming Guide. *Design Guide*, 2017. Retrieved May 10, 2021, from `https://eva.fing.edu.uy/pluginfile.php/174141/mod_resource/content/1/CUDA_C_Programming_Guide.pdf`.

[11] OpenCV. Version 3.4.10 Documentation. Retrieved April 12, 2021, from `https://docs.opencv.org/3.4.10/`.

[12] Photoneo. Photoneo 3D scanner. Retrieved March 17, 2021 from `https://www.photoneo.com/products/phoxi-scan-m/`.

[13] Guido Ranzuglia, Marco Callieri, Matteo Dellepiane, Paolo Cignoni, and Roberto Scopigno. Meshlab as a complete tool for the integration of photos and color with high resolution 3d geometry data. In *CAA 2012 Conference Proceedings*, pages 406--416. Pallas Publications - Amsterdam University Press (AUP), 2013.

[14] W. S. Rasband. ImageJ. National Institutes of Health, Bethesda, Maryland, USA, `https://imagej.nih.gov/ij/`, 1997-2018.

[15] Jie Shao, Wuming Zhang, Nicolas Mellado, Pierre Grussenmeyer, Renju Li, Yiming Chen, Peng Wan, Xintong Zhang, and Shangshu Cai. Automated markerless registration of point clouds from tls and structured light scanner for heritage documentation. *Journal of Cultural Heritage*, 35:16--24, 2019. Modern and Contemporary Art.

[16] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Proceedings of the Sixth International Conference on Computer Vision*, ICCV '98, pages 839--846, USA, 1998. IEEE Computer Society.