MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Akcelerácia bilaterálneho filtra na GPU

DIPLOMOVÁ PRÁCA

Jaroslav Žilák

Brno, jar 2013

Prehlásenie

Prehlasujem, že táto diplomová práca je mojím pôvodným autorským dielom, ktoré som vypracoval samostatne. Všetky zdroje, pramene a literatúru, ktoré som pri vypracovaní používal alebo z nich čerpal, v práci riadne citujem s uvedením úplného odkazu na príslušný zdroj.

Vedúci práce: RNDr. Pavel Karas

Poďakovanie

Týmto sa chcem poďakovať pánovi RNDr. Pavlovi Karasovi za odborné vedenie, cenné pripomienky a rady pri vypracovaní tejto práce.

Zhrnutie

Bilaterálny filter je v súčasnosti pomerne často používanou nelineárnou filtráciou, ktorá nachádza široké uplatnenie najmä v oblasti medicíny, spracovania statického aj časovo premenlivého obrazu, zábavného priemyslu a mnoho ďalších. Výpočtovo sa však radí medzi veľmi náročné operácie, čo do značnej miery potláča jeho prednosti. Spôsob ako výpočet urýchliť, sa stali predmetom veľkého množstva publikácii, ktoré prichádzajú s alternatívami, aproximáciami a vylepšeniami. Spomedzi všetkých, táto práca vyberá tri, ktoré do detailov popisuje z hľadiska presnosti, rýchlosti a časovo-priestorových nárokov. Ústrednou témou práce je potom podrobná dokumentácia vlastných GPU implementácii jednotlivých metód pre platformu CUDA, ktoré pôvodné publikované CPU implementácie okrem paralelizácie dopĺňajú o podporu tretej dimenzie. Výsledkom práce je okrem už spomínaných implementácií komplexný prehľad vybraných metód, z ktorých sa vybrala jedna pre doplnenie knižnice i3lib Ústavu spracovania biomedicínskych dát Fakulty informatiky, Masarykovej univerzity.

Kľúčové slová

bilaterálny filter, GPU, CUDA, bilaterálny grid, PBFIC, rekurzívne filtrovanie, konvolúcia, 3D dáta

Obsah

Úvod

Aktuálnym a jedným z najrýchlejšie rastúcim trendom vo svete informatiky, je nepochybne snaha o pralelizáciu. Ide o dôsledok stále pomalšie rastúcej krivky nárastu výkonu zvyšovaním frekvencie. Navýšenie výkonu sa dosahuje multiplikáciou výpočtových jednotiek v podobe procesorových a grafických jadier, ktoré kooperatívne pracujú na riešení daného problému. Zdanlivo elegantné riešenie, však do problematiky zanáša problém nový, ktorým je jednak réžia spojená so vzájomnou komunikáciou týchto jednotiek, no najmä problém ako písať paralelné algoritmy. V súčasnosti ide zatiaľ o neautomatizovaný postup, ktorý tak ostáva na ľudskej zručnosti.

Rovnakou otázkou sa zaoberal aj celosvetovo známy výrobca grafických kariet NVIDIA, ktorý na trh prišiel s užívateľsky prívetivým frameworkom umožňujúcim paralelne programovanie za cenu minimálneho učenia sa. Aj vďaka tomuto frameworku, pomenovanom skratkou CUDA, sa proces paralelizácie značne úrýchlil a umožnil tak potenciál grafických kariet naplno využiť nielen v hernom priemysle. Snaha o paralelizáciu postupne prenikala aj do oblasti digitálneho spracovania obrazu, kde si svoje špecifické uplatnenia našla najmä v oblasti bioinformatiky.

V obore bioinformatiky sa spracovaním obrazu, rozumie časovo náročný výpočet v podobe filtrácií na objemovo veľkých, v zásade mnohodimenzionálnych dátach. Časovo je teda filtrácia veľmi náročnou operáciou, ktorá však obyčajne predstavuje opakované aplikovanie tej istej množiny inštrukcií na každý pixel resp. voxel zvlášť. Tieto inštrukcie sú na sebe väčšinou nezávislé a teda pre paralelizáciu, ktorá uplatňuje ideu rozdeľ a panuj, ideálne.

Jedným z týchto filtrov a zároveň predmetom tejto práce je aj bilaterálny filter. Ten svojími vlastnosťami, rozmazanie súvislých plôch pri zachovaní hrán, predstavuje dôležitý prostriedok pre elimináciu chýb a šumu v obraze, zvyšovanie detailov obrazu, separáciu luminozity od textúry a množstvo ďalších aplikácií. Svojím významom je porovnateľný s gaussovým filtrom, z ktorého vychádza. Na rozdiel od neho, však ide o filter nelineárny, čo sa prejavuje jeho asymptotickou časovou zložitosťou výpočtu. Efektívny výpočet bilaterálneho filtra je už takmer 20 rokov

predmetom výskumu a odborných článkov. Tie navrhujú aproximáciu s rôznými vylepšeniami, urýchlením a spresnením. Spomedzi všetkých, táto práca vyberá tri, metódu bilaterálneho gridu, metódu PBFIC rezov a rekurzívnu metódu, ktorých paralelná varianta buď neexistuje, alebo existuje paralelizovaná len jej časť. Na týchto troch metódach sa snaží čitateľovi priblížiť problematiku bilaterálnej filtrácie a paralelizácie v prostredí CUDA. Zároveň prezentuje ich konkrétnu implementáciu v tomto prostredí s jednotlivými modifikáciami a optimalizáciami.

Celá práca sa delí na teoretickú a praktickú časť. V časti teoretickej sa v prvej kapitole venuje praktickým základom platformy CUDA. Popisuje trojúrovňovú hierarchiu programového modelu, hierarchiu pamätí a značenie compute capability, pre ktoré na jednoduchom príklade vysvetľuje jeho praktický význam. Zároveň poskytuje stručný prehľad najpoužívanejších inštrukcií spolu s ich použitím a významom. Druhá a tretia kapitola sa potom už venujú samotnej filtrácii. Najskôr intuitívne popisujú najdôležitejšie vlastnosti gaussovej konvolúcie a rekurzívneho filtrovania, z ktorých v ďalšej kapitole priamo vyplynie bilaterálna filtrácia. Každý typ filtra je naviac doplnený o matematický predpis, prípadne aj s odvodením. Tretia kapitola sa ďalej člení na podkapitoly popisujúce a hodnotiace vybrané metódy výpočtu bilaterálneho filtra. Opäť ich dopĺňa o istý matematický základ spolu so slovným odvodením. Praktická časť detailne popisuje všetky GPU implementácie spolu s odôvodnením niektorých jej častí. Posledná kapitola praktickej časti sa potom venuje metodike merania, výsledkom a ich zhodnoteniu.

Kapitola 1

Platforma CUDA

CUDA ako univerzálna platforma určená pre všeobecnú paralelizáciu výpočtov a programovací model bola vytvorená firmou NVIDIA v roku 2006. Ako programové prostredie využíva iné jazyky vyššej úrovne, primárne C/C++ a FORTRAN. K dispozícii sú však aj wrappery tretích strán pre podporu ďalších jazykov ako napr. JAVA, Python, Matlab a iné, čím umožňuje vývojárom programovať vysoko paralelizované aplikácie bez nutnosti učiť sa nový jazyk. Ďalšou výhodou platformy CUDA je odtienenie architektúry od samotnej aplikácie, čo zaručuje jednoduchú škálovateľnosť vzhľadom k počtu jadier GPU.

Ako základná výpočtová jednotka sa rozumie vlákno (thread). Tie sú ďalej organizované do uniformne veľkých blokov (threadblock), ktoré sú usporiadané vo forme 1 až 3 dimenzionálnej mriežky (grid). Uvedená trojúrovňová abstrakcia tak umožňuje dostatočne jemnú výpočtovú a dátovú paralelizáciu riešeného problému.

1.1 Programovací model

Napriek tomu, že CUDA rozširuje veľa jazykov, my sa zameriame výlučne na jazyk C resp. C++, v ktorom je písaná celá implementačná časť tejto diplomovej práce.

1.1.1 Typové kvalifikátory

CUDA zavádza systém typových kvalifikátorov, ktorými určuje, resp. delí aplikačný priestor na dve časti, časť hostiteľskú a časť zariadenia grafickej karty. Príslušnosť ku konkrétnemu podpriestoru určujú typové kvalifikátory. Pre funkcie sú to kvalifikátory:

- `__device__` – kód funkcie je volateľný a spustiteľný len na zariadení GPU,

- `__host__` – kód funkcie je volateľný a spustiteľný len v hostiteľskom a z hostiteľského prostredia a
- `__global__` – ktorý umožňuje volať funkciu z hostiteľského prostredia, pričom sa jej kód vykoná na zariadení GPU.

Pri špecifikácii funkcie ako `__global__` nesmie táto funkcia vracať inú návratovú hodnotu ako void a zároveň musí byť pri volaní parametrizovaná, viď nasledujúci odstavec o kerneloch. Pre premenné sú to kvalifikátory:

- `__shared__` – premenná má vyhradené miesto v zdieľanej pamäti, a teda k nemu majú prístup len vlákna príslušného bloku, s ktorým tiež zaniká,
- `__constant__` – premenná je uložená v konštantnej pamäti GPU, vďaka čomu je viditeľná všetkými vláknami a má životnosť aplikácie,
- `__restrict__` – reflektuje rozšírenie jazyka C v iso norme C99 v ktorom boli zavedené tzv. restricted ukazatele. Tie slúžia ako nápoveda kompilátora, že daný ukazovateľ nie je aliasom iného objektu, a teda je možné nad ním vykonávať optimálizácie.
- `__device__` – je doplňujúcim kvalifikátorom pre predošlé tri kvalifikátory. Samostatne určuje, že premenná má byť umiestnená do globálnej pamäti a teda rovnako ako u `__constant__` je viditeľná všetkými vláknami a má životnosť aplikácie.

1.1.2 Kernely

Jedným z mála rozšírení štandardného jazyka C/C++ je možnosť definovať špeciálne funkcie nazývane kernely. Tie sú na rozdiel od štandardných funkcii volané súčasne každým vláknom, a teda kód takejto funkcie sa vykonáva paralelne. Syntax volania kernelu rozširuje štandardné volanie C/C++ funkcie o konfiguráciu označovanú trojitou ostrou zátvorkou <<<, >>>. Definuje v poradí: veľkosť bloku vlákien, veľkosť mriežky, veľkosť dynamicky alokovanej zdieľanej pamäte a ukazovateľ na aktuálny stream.

1.1.3 Compute capability

Pre podrobnejší popis jednotlivých parametrov konfigurácie a najmä ich HW obmedzení sa najskôr sústreďme na označenie, ktorým NVIDIA popisuje funkcionalitu každej grafickej karty podporujúcej platformu CUDA.

Ide o konvenciu, ktorá nesie skratku c.c. (compute cpability) a začína sa používať od grafických kariet rady G80.

V súčasnosti sa pohybuje v rozsahu 1.0 až 1.3 pre grafické karty rady G80 až GT200, 2.0, 2.1 pre rady GF104 až GF116, 3.0 GK104 až GK107 a 3.5 kde sú predstaviteľmi TESLA K20(X) a najnovšie GTX Titan. Intuitívne, čím vyššia hodnota c.c., tým viac špecializovaných funkcii a prostriedkov v podobe počtu výpočtových jadier a pamäte karta poskytuje.

1.1.4 Parametre konfigurácie

Opäť sa ale vráťme ku, pre nás, podstatným parametrom konfigurácie, ktorými sú veľkosti bloku a mriežky. Dynamická alokácia ani streamy sa v tejto práci nevyužívajú, a preto sa o nich zmienime len okrajovo. Zásadnými obmedzeniami dynamickej alokácie sú neschopnosť definovať viac ako jeden súvislý blok pamäte a maximálna alokovateľná veľkosť. Tá sa pohybuje v rozsahu od 16 kiB (c.c. < 2.0) do 48 kiB (c.c. ≥ 2.0) na blok a defaultne je nastavená na nulu, t.j. nealokuje sa žiadna zdieľaná pamäť. Stream je rovnako ako parameter dynamickej alokácie nastavený vo všetkých volaniach kernelov tejto práce na defaultnú hodnotu, nula, čím sa jednotlivé po sebe nasledujúce kernely vykonávajú sekvenčne bez prelínania.

Veľkosti bloku a veľkosti gridu sa definujú preddeklarovanou štruktúrou `dim3`, ktorá ma zložky `x`, `y`, `z` zodpovedajúce rozmerom jednotlivých dimenzii. Rovnako ako u dynamického alokavania zdieľanej pamäte, aj tu sú limity závislé od c.c. grafickej karty. Spolu s maximálnym počtom vlákien a maximálnym počtom blokov na multiprocesoroch potom určujú mieru vyťaženia GPU.

Príklad

Nasledujúci príklad ilustruje túto závislosť. Uvažujme c.c. 2.0, pre ktorú sú limity nasledujúce: Počet blokov na multiprocesor označme $M = 8$, Počet vlákien na multiprocesor $T = 1536$. Ak zvolíme veľkosť bloku 8x8, potom pre $100\,\%$ utilizáciu GPU potrebujeme $\frac{1536}{8.8} = 12$ blokov. GPU s c.c. 2.0 však na multiprocesor podporuje maximálne 8 blokov, a teda utilizácia GPU bude len $\frac{8}{12} = 66.667\,\%$. Zvoľme teda väčší blok, napríklad 32x32. Rovnakým postupom dospejeme k výsledku $\frac{1536}{32.32} = 1.5$ bloku. Keďže multiprocesor nedokáže spracovávať fragmenty blokov, vo výsledku dostaneme jediný blok na multiprocesor, čo zodpovedá 1024 vláknam a $66.667\,\%$ utilizácii.

Ďalším faktorom určujúcim mieru využitia potenciálu GPU je celkový

počet blokov na multiprocesor. Pri ich nedostatku, má plánovač multiprocesora len obmedzené možnosti prepínania medzi blokmi, čo opäť vedie k zníženiu utilizácie.

Odvodením jednoduchej rovnice:

$$U = \frac{M * (B_x * B_y)}{T}, \tag{1.1}$$

kde U je utilizácia, B_x, B_y je šírka, resp. výška bloku, sme schopní určiť optimálnu veľkosť bloku pre danú c.c. Pri voľbe veľkosti bloku je vhodné vyberať mocniny čísla dva a teda pre náš príklad je optimálnym blokom napríklad blok veľkosti: 16x16.

Pre jednoduchú orientáciu v pomerne zložitej trojúrovňovej hierarchii CUDA zaviedla indexačné premenné. Pre každé vlákno a blok je vyhradená jedna s názvom `threadIdx`, resp. `blockIdx`. Všetky indexačné premenné sú typu `dim3`, pričom `threadIdx` určuje pozíciu aktuálneho vlákna v rámci tredbloku a `blockIdx` určuje pozíciu bloku v rámci gridu.

1.1.5 Synchronizácia

Ďalším dôležitým rozšírením jazyka C/C++ zjednodušujúcim koopertivitu vlákien pri výpočte a prístupu k pamätiam sú synchronizačné rutiny. Delia sa na dve kategórie. Prvá kategória sú synchronizácie na úrovni vlákien v rámci bloku. Patrí sem funkcia `__syncthreads()`, ktorá je od c.c. 2.0 doplnená o ďalšie 3 varianty. Každá z týchto variant vyžaduje najviac jeden parameter, predikát, ktorý sa vyhodnotí každým vláknom v rámci bloku a následne funkcia vráti výsledok podľa toho, o akú variantu sa jedná. Pre

- `__syncthreads_and()` bude výsledkom logická hodnota true len a len vtedy, ak všetky vlákna bloku spĺňajú predikát. Analogicky pre
- `__syncthreads_or()`, kde funkcia vráti true, ak aspoň jedno vlákno spĺňa zadaný predikát. Poslednou variantou je
- `__syncthreads_count()`, ktorá vráti počet vlákien, ktoré vyhodnotili predikát ako pravdivý.

  Druhá kategória synchronizačných rutín synchronizuje na základe aktuálneho stavu pamätí. Patria sem celkovo tri funkcie:

- `__threadfence_block()`, ktorá uspí vlákno dovtedy, kým nie sú všetky zmeny globálnej a zdieľanej pamäte vykonané daným vláknom viditeľné všetkými ostatnými v danom bloku.

- `__threadfence()` rozširuje funkciu `__threadfence_block` o čakanie na viditeľnosť zmien globálnej pamäte všetkými ostatnými vláknami v rámci zariadenia. Poslednou funkciou tejto kategórie je
- `__threadfence_system()`, ktorá funkciu `__threadfence()` ďalej rozširuje o čakanie na viditeľnosť zmien zdieľanej RAM vláknami CPU.

Synchronizácia medzi blokmi nie je natívne podporovaná a jej potreba zvyčajne nasvedčuje nedostatočnú dekompozíciu riešeného problému. Je však možná, a to buď s využitím atomických operácii, alebo preferovanejšie, ukončením a opätovným vyvolaním iného/toho istého kernelu. Z hľadiska časovej náročnosti je invokácia nového kernelu pomerne drahá operácia, rádovo jednotky až desiatky mikro sekúnd, a preto je žiadúce vyhýbať sa prípadnému presegmentovávaniu aplikácie na kernely.

1.1.6 Správa pamätí

Posledným rozšírením, ktorému sa v tejto kapitole budeme venovať sú rutiny pre prácu s pamäťou. Ide o sadu nasledujúcich funkcii:

- `cudaMalloc()` – ktorá slúži pre alokáciu lineárnej časti globálnej pamäte na zariadení GPU,
- `cudaFree()` – je náprotivkom funkcie `cudaMalloc` a alokovanú GPU pamäť uvoľňuje,
- `cudaMemcpy()` – slúži pre kopírovanie dát nie len medzi alokovanou pamäťou GPU, ale aj medzi hostiteľskou a GPU pamäťou. Umožňuje nám tak grafickej karte dáta predávať, vykonať nad nimi príslušný výpočet, po ktorom si ich z GPU môžme opäť vybrať.

Smer kopírovania určuje posledný parameter, ktorý nadobúda jednu z hodnôt: `cudaMemcpyHostToDevice`, `cudaMemcpyHostToDevice`. Okrem funkcii pre správu GPU pamätí CUDA tiež definuje obdobné funkcie pre hostiteľskú pamäť. Sú to:

- `cudaHostAlloc()` – ktorá na rozdiel od štandardnej funkcie jazyka C – `malloc()`, pamäť nielen alokuje, ale aj uzamkne proti stránkovaniu. To by pri behu kernelu mohlo spôsobovať výpadky stránok a beh celej aplikácie spomaliť.
- `cudaHostFree()` – náprotivok funkcie `cudaHostAlloc()`,

- `cudaHostRegister()` – umožňuje uzamknúť pamäť alokovanú štandardným spôsobom cez `malloc()`.

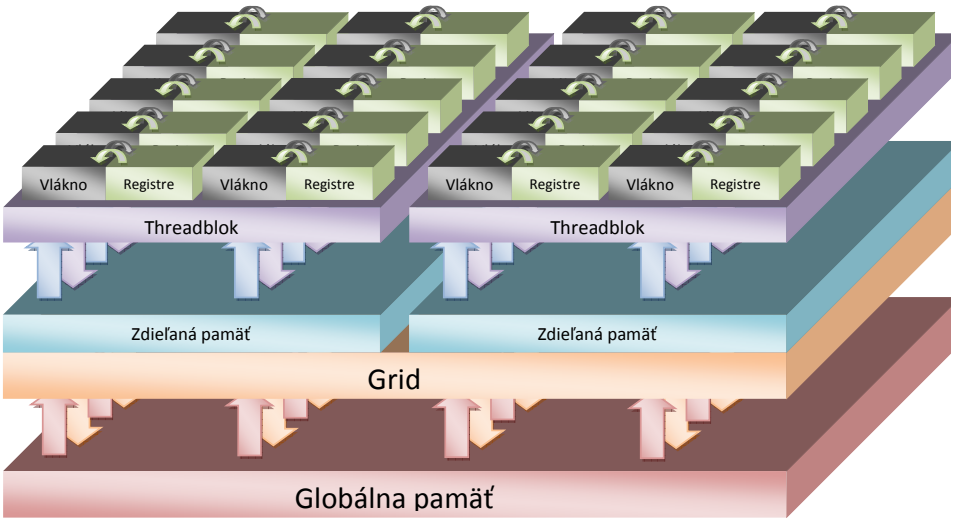
V tejto kapitole sme uviedli len časť funkcionality, ktorú táto práca v implementácii využíva, a pre ďalšie informácie čitateľa odkážeme na oficiálne stránky NVIDIA, prípadne užívateľský manuál, z ktorého čerpá aj táto kapitola [1].

1.2 Hierarchia pamätí

Jednotlivé vlákna majú možnosť počas svojho behu pristupovať k celkom trom druhom pamätí. Každé vlákno má svoju vlastnú lokálnu pamäť v podobe registrov a v prípade ich nedostatku taktiež časť globálnej pamäte. Táto pamäť má životnosť vlákna a prístup k nej má výlučne len zodpovedajúce vlákno. Zároveň ide o najrýchlejšiu pamäť.

Na nižšej úrovni je už spomínaná zdieľaná pamäť, ktorá je dostupná pre všetky vlákna bloku a spolu s blokom aj zaniká. Jej primárne využitie je medzi-vláknová komunikácia a cacheovanie globálnej pamäte.

Na najnižšej úrovni je pamäť globálna, ku ktorej majú prístup všetky vlákna, všetkých blokov. Táto pamäť má životnosť aplikácie a je najpomalšou pamäťou (400–600 taktov na I/O operáciu). Špeciálnym druhom globálnej pamäti sú textúrovacia pamäť a pamäť konštánt. Textúrovacia pamäť na rozdiel od globálnej, poskytuje hardwarovo implementovanú funkcionalitu, ako napríklad bilineárnu/trilineárnu interpolácu, rôzne adresovacie módy, atď. Pamäť konštánt, ako už názov napovedá, je primárne určená pre ukladanie statických dát, ktoré sa počas behu aplikácie nemenia. Jej výhodou je, pri dodržaní istých pravidiel, rýchlosť prístupu k dátam. Nevýhodou konštantnej pamäte je predovšetkým jej veľkosť, ktorá sa zatiaľ s ohľadom na c.c. nezmenila a predstavuje 64 kiB pre celú aplikáciu. Obe tieto špecializované pamäte sú určené len pre čítanie. Hierarchiu pamätí spolu s hierarchiou architektúry CUDA, znázorňuje nasledujúca ilustrácia:



Obr. 1.1: Architektúra platformy CUDA spolu s hierarchiou pamätí.

Kapitola 2

Bilaterálny filter

2.1 Gaussov filter

Pre lepšie pochopenie nasledujúcej kapitoly pojednávajúcej o bilaterálnej filtrácii, si najskôr pripomenieme filtráciu gaussovskú, ktorá je jej základom.

Základnou operáciou linárneho filtrovania je konvolúcia s kladným jadrom tiež chápaným ako matica váh. Gaussovský filter je konvolúcia, ktorého váhy gaussovského jadra kvadraticky klesajú s rastúcou vzdialenosťou od stredu. Pre 1D jadro matematicky popisujeme ako:

$$f_\sigma(x) = e^{-\frac{(x-c)^2}{2\sigma}}, \tag{2.1}$$

kde x je súradnica v jadre, c vyjadruje centrálny bod jadra a σ určuje rýchlosť poklesu váh v závislosti na vzdialenosti od stredu. Obdobne pre 2D by bolo jadro definované ako:

$$f_\sigma(x,y) = e^{-\frac{(x-c_x)^2}{2\sigma_x}-\frac{(y-c_y)^2}{2\sigma_y}}, \tag{2.2}$$

prípadne ak je $\sigma_x = \sigma_y$:

$$f_\sigma(x,y) = e^{-\frac{(x-c_x)^2+(y-c_y)^2}{2\sigma}}. \tag{2.3}$$

Analogicky môžeme rozširovať 1D jadro do ľubovoľnej dimenzie, prostým pridávaním členov $e^{-\frac{(i-c_i)^2}{2\sigma_i}}$. Tento postup je reverzibilný, čo znamená, že ľubovoľné Gaussovo jadro možno rozštiepiť na jednotlivé jeho členy, 1D jadrá. Táto vlastnosť sa nazýva separabilita a umožňuje nám výsledok konvolúcie počítať v čase $O(NM)$, miesto $O(N^2)$, kde M je počet všetkých rozmerov.

Ďalšia príjemná vlastnosť tohto jadra plynie z lokality energie. Tá je sústredená v okolí centrálneho bodu c, od ktorého rýchlo klesá. Dá sa matematicky dokázať, že približne $99.7\,\%$ celkovej energie Gaussovej funkcie

je vo vzdialenosti 3σ od centrálneho bodu. Pri výpočte tak nemusíme pre každý počítaný bod prechádzať celý rozmer, ale len jeho časť. Toto tvrdenie však platí len pre primerane veľké hodnoty σ.

Konvolúcia dvoch signálov, f a g, je definovaná ako:

$$(f * g)(x) = \int_{n=-\infty}^{\infty} f(x-n)g(n)dn, \tag{2.4}$$

no vzhľadom na povahu počítačovej reprezentácie dát, sa bez ujmy na obecnosti obmedzíme na diskrétnu variantu tejto definície:

$$(f * g)[x] = \sum_{n=-\infty}^{\infty} f[x-n]g[n], \tag{2.5}$$

kde f chápeme ako vstupný 1D diskrétny signál a g ako nekonečne veľké 1D jadro. Substitúciou g gaussovským jadrom f_σ obmedzeným veľkosťou 3σ, konvolvovanej funkcie filtrovaným obrazom I a normalizáciou výsledku dostávame:

$$(I * f_\sigma)[x] = \frac{\sum_{n=-3\sigma}^{3\sigma} I[x-n]f_\sigma[n]}{\sum_{n=-3\sigma}^{3\sigma} f_\sigma[n]}. \tag{2.6}$$

Po zobecnení získame už dobre známu rovnicu Gaussovho filtra označovaného ako I^G:

$$I^G = \frac{\sum_{N(x)} f_\sigma(x)I(x)}{\sum_{N(x)} f_\sigma(x)}, \tag{2.7}$$

kde $N(x)$ je okolie bodu x do vzdialenosti 3σ. Z výslednej rovnice je zrejmé, že gaussovský filter nahrádza každý pixel vstupného obrazu váženým priemerom intenzít jeho okolia. Dochádza tak k celkovému rozmazaniu a strate detailov, čo z frekvenčného hľadiska chápeme ako potláčanie vysokých frekvencii. Gaussovský fiter je teda nízkofrekvenčným filtrom.

2.1.1 Deriche

Okrem už spomínanej metódy akcelerácie výpočtu s využitím separability existujú rôzne ďalšie spôsoby akcelerácie. Najrýchlejšou z nich je aproximácia rekurzívnymi filtrami, ktoré fungujú na princípe recyklácie už vypočítaných hodnôt pre určenie nasledujúcej hodnoty pri prechode obrazom. Presnosť aproximácie pritom závisí na počte znovu použitých hodnôt, v literatúre označované ako rád rekurzívneho filtra. Pre zachovanie symetricity filtra, je nutné obrazom prejsť dvakrát, tzv. kauzálny a antikauzálny prechod, v literatúre tiež označovaný ako extenzívny a antiextenzívny.

Asi najznámejším dodnes používaným predstaviteľom rekurzívnej aproximácie Gaussovho filtra je varianta od Rachid Dericheho [2], ktorý ju pôvodne navrhol pre svoj detektor hrán. Definoval ju sústavou ôsmych resp. dvanástich rovníc pre výpočet koeficientov rekurzívneho filtra druhého rádu. V súčasnosti existujú rozšírenia v podobe Dericheho filtrov vyšších rádov a aproximácie derivácii Gaussovej funkcie, no pre naše účely postačí pôvodný rekurzívny filter druhého rádu aproximujúci nultú deriváciu gaussovej funkcie, pre ktorú majú zmienené extenzívne a antiextenzívne rovnice nasledujúci tvar:

$$f_{ex}(x) = a_0 f(x) + a_1 f(x-1) - b_0 f_{ex}(x-1) - b_1 f_{ex}(x-2) \tag{2.8}$$

$$I^G(x) = f_{an}(x) = a_2 f_{ex}(x) + a_3 f_{ex}(x-1) - b_0 f_{an}(x-1) - b_1 f_{an}(x-2). \tag{2.9}$$

Všetky koeficienty sa počítajú len raz, na začiatku filtrácie, podľa predpisu:

$$b_0 = 2e^{-\alpha} \tag{2.10}$$

$$b_1 = -e^{-2\alpha} \tag{2.11}$$

$$a_0 = \frac{(1-e^{-\alpha})^2}{1+\alpha b_0 + b_1} \tag{2.12}$$

$$a_1 = a_0 e^{-\alpha}(\alpha - 1) \tag{2.13}$$

$$a_2 = a_0 e^{-\alpha}(\alpha + 1) \tag{2.14}$$

$$a_3 = a_0 b_1 \tag{2.15}$$

$$c_0 = c_1 = 1, \tag{2.16}$$

kde $\alpha = \frac{1.695}{\sigma}$. Koeficienty c_0 a c_1 sa v rovniciach filtrácie priamo neuplatňujú, no pre filtráciu sú nepostrádateľné, pretože určujú iniciálnu hodnotu pre oba typy prechodov. Uvedené rovnice sú oproti pôvodným rovniciam publikovaným autorom mierne upravené a konštanta 1.695 v prevode α na odpovedajúcu σ je experimentálne odvodená.

2.2 Od Gaussa k bilaterálnemu filtrovaniu

Bilaterálna filtrácia je technika pre vyhladzovanie obrazových dát pri zachovávaní hrán. Vznikla v roku 1995 pri experimentoch Auricha a Wueleho s nelineárnymi gaussovskými filtrami [3]. V súčasnosti ide o široko využívanú metódu, používanú v oblastiach filtrácie a editácie obrazu, štylizácie, odhadov optických tokov a mnoho ďalších. Matematicky je defino-

vaná ako

$$I^B(x) = \frac{\sum_{y \in N(x)} g_{\sigma_S}(x,y) g_{\sigma_R}\big(I(x), I(y)\big) I(y)\big)}{\sum_{y \in N(x)} \big(g_{\sigma_S}(x,y) g_{\sigma_R}\big(I(x), I(y)\big)}, \tag{2.17}$$

kde g_{σ_S} a g_{σ_R} sú jadrá gaussovského filtru, $N(x)$ je množina pixelov okolia počítaného pixelu a $I(x)$ je intenzita pixelu x.

Voľne formulované, ide podobne ako u Gaussovho filtra o náhradu každého obrazového bodu váženým priemerom jeho okolia. Jednotlivé váhy však nezávisia len na priestorovej vzdialenosti, ale aj od rozdielu intenzít okolia a centrálneho bodu. Mieru tejto závislosti parametrizujú σ_S a σ_R. Intuitívne, ak je σ_S priestorovým parametrom, bude vo výsledku určovať mieru vyhladenia. Obdobne pre intenzity σ_R určuje mieru zachovania hrán. V krajných prípadoch, kedy má σ_R veľkú hodnotu sa z bilaterálneho filtra stáva filter gaussovský a naopak pre σ_R blízke nule, sa filter vôbec neuplatní.

2.3 Spôsoby výpočtu bilaterálneho filtra

Do dnešného dňa bolo navrhnutých niekoľko desiatok aproximácii a vylepšení, my sa ale zameriame len na niektoré z nich. Menovite na metódu bilaterálneho gridu, jej odľahčenú variantu využívajúcu princíp intenzitných rezov, a pre nás asi najzaujímavejšiu rekurzívnu metódou. Pozornejšieho čitateľa určite napadla myšlienka využiť separabilitu jednotlivých gaussovských filtrov. Z princípu nelinearity bilaterálneho filtra to však nie je možné, no ako aproximácia je pri malých sigmách dobre použiteľná, a preto bola zaradená do tejto diplomovej práce ako štvrtá skúmaná metóda.

2.3.1 Bilaterálny grid

Je dôsledkom jedného z možných chápaní matematického predpisu bilaterálneho filtru. Ak by sme intenzity jednotlivých obrazových bodov chápali ako ďalší rozmer, t.j. v prípade 2D obrazu ako súradnicu hĺbky, resp. výšky, bilaterálna filtrácia, by sa redukovala na jednoduché gaussové filtrovanie. To vieme aproximovať napríklad Dericheho rekurzívnym filtrom v čase O(n), čím sa aj výpočet bilaterálneho filtra urýchli na O(n). Presnosť výsledku opäť závisí na použitej aproximácii gaussovho filtra, ktorá je už pri malom ráde pomerne vysoká.Táto zdanlivo ideálna metóda má však jeden vážny nedostatok, ktorým je až rádový nárast pamäťovej zložitosti.

Ten plynie z pridania ďalšej dimenzie, ktorá už pri použití jednoduchých dátových typov akým je napr. byte, predstavuje nárast pamäťovej zložitosti na 256-násobok pôvodného obrazu.

Tento nedostatok sa vo svojej publikácii [4] snažia potlačiť Jiawen Chen, Sylvain Paris a Fredo Durand vhodným podvzorkovaním vstupného obrazu, a to z hľadiska priestorovej aj rozsahovej domény. Toto podvzorkovanie sa deje už pri samotnej tvorbe bilaterálneho gridu, kedy sa pri prechode vstupného obrazu hodnota každého obrazového bodu pripočítaná k príslušnej, na nulu inicializovanej, bunke gridu. Jej súradnicu pritom určuje intenzita a pozícia konkrétneho obrazového bodu podľa nasledujúceho predpisu:

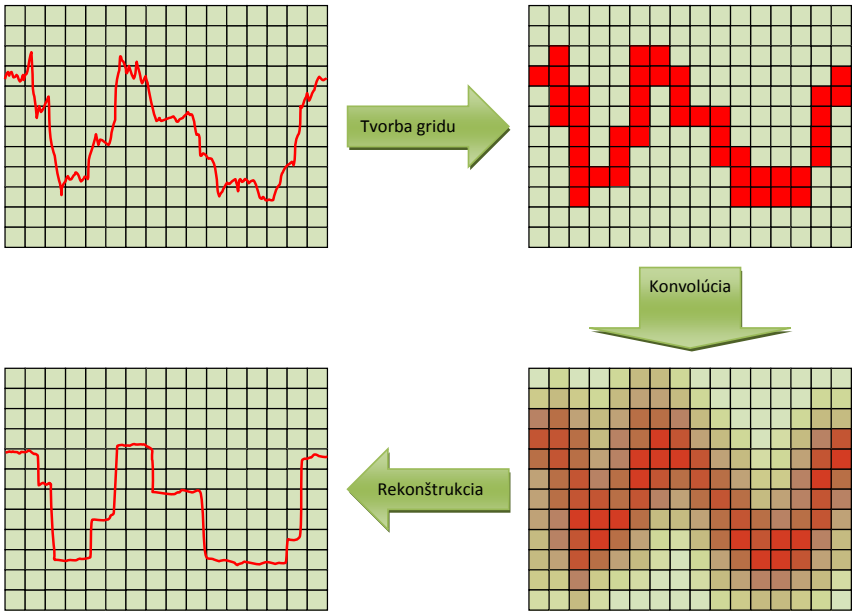
$$\Gamma[\lfloor x/\sigma_S \rfloor, \lfloor y/\sigma_S \rfloor, \lfloor z/\sigma_S \rfloor, \cdots, \lfloor I(x,y,z,\cdots)/\sigma_R \rfloor]. \tag{2.18}$$

Keďže sa do jednej bunky namapuje viac ako jeden bod, každá bunka gridu musí ešte okrem sumy intenzít uchovávať aj počet sčítaných hodnôt, ktorý sa vo finálnom kroku použije ako váha. Bunkou gridu teda rozumieme usporiadanú dvojicu, intenzita, váha. Jeden krok tvorby bilaterálneho gridu potom odpovedá súčtu dvoch dvojprvkových vektorov:

$$\Gamma[\lfloor x/\sigma_S \rfloor, \lfloor y/\sigma_S \rfloor, \lfloor z/\sigma_S \rfloor, \cdots, \lfloor I(x,y,z,\cdots)/\sigma_R \rfloor] \mathrel{+}= \big(I(x,y,z,\cdots,), 1.0\big) \tag{2.19}$$

Okrem podvzorkovania, ktoré už samo o sebe funguje ako hrubá aproximácia gaussa (box filter), sa pre presnejšie výsledky celý grid ešte prefiltruje gaussovým filtrom s malou, pevne nastavenou sigmou.

Posledným krokom výpočtu je tzv. slicing. Ide o symetrickú operáciu k operácii tvorby gridu. Opäť sa prechádza celý vstupný obraz na základe ktorého pristupujeme k jednotlivým bunkám gridu. Pre každý bod však nevyčítame len hodnoty zodpovedajúcej bunky, ale aj hodnoty jej okolia, z ktorých následne lineárne interpolujeme novú hodnotu pixelu. Váhu a sumu intenzít pritom interpolujeme zvlášť, a až po ich interpolácii interpolovanú intenzitu váhou normalizujeme. Pozornejší čitateľ si určite všimol problém s krajnými bodmi vo vzdialenosti σ_S resp. σ_R, u ktorých už neexistuje žiadne okolie potrebné pre interpoláciu. Autori tento problém prenechali na textúrovaciu jednotku grafickej karty, ktorá buď tieto body neinterpoluje, alebo ako ich okolie berie body zo začiatku obrazu. Režim závisí na aktuálnom nastavení grafickej karty a autori sa o ňom nezmieňujú.



Obr. 2.1: Postup filtrácie za pomoci bilaterálneho gridu

2.3.2 Rozklad na PBFIC

Ide o metódu navrhnutú trojicou Qungxiong Yangom, Kar-Han Tanom a Narendra Ahujalom [5], ktorá obdobne ako metóda predošlá vznikla na základe istého pohľadu na bilaterálnu filtráciu. Jej princíp pekne ilustruje postup, akým sa autori snažili osamostatniť priestorovú filtráciu od rozsahovej. Tento postup je postavený na základe povahy digitálnej reprezentácie obrazu, ktorá má obmedzený počet kvantizačných úrovní. Ak sa teda obmedzíme len na k, $k \in \{L_0, L_1, \cdots L_{N-1}\}$ úrovní intenzít, pre každú úroveň k a každý pixel $I(y)$ obrazu definujeme vrstvu:

$$J_k^B(x) = \frac{\sum_{y \in N(x)} g_{\sigma_S}(x,y) g_{\sigma_R}\big(k, I(y)\big) I(y)}{\sum_{y \in N(x)} g_{\sigma_S}(x,y) g_{\sigma_R}\big(k, I(y)\big)}, \tag{2.20}$$

autormi nazvanú ako Principle Bilateral Filtered Image Component (PBFIC). Ako z predpisu vidieť, rozsahová časť filtra sa úplne osamostatnila, vďaka čomu sa dá počítať samostatne. Výpočet predstavuje dobre známu Gaussovú filtráciu parametrizovanú σ_R, ktorá sa dá počítať v čase $O(n)$. Na

výsledok sa potom opäť aplikuje Gaussová filtrácie, tentokrát ale parametrizovaná σ_S. Celková časová zložitosť výpočtu jednej PBFIC je teda $O(n)$. Rovnako ako u bilaterálneho gridu aj tu sa normalizačné koeficienty a intenzity počítajú samostatne a normalizácia prebieha až v poslednom kroku výpočtu. Tým je lineárna interpolácia dvoch susedných PBFIC (J^B_k, J^B_{k+1}), podľa nasledujúcej rovnice:

$$I^B(x) = \big(L_{k+1} - I(x)\big)J^B_k(x) + \big(I(x) - L_k\big)J^B_{k+1}(x). \tag{2.21}$$

Je pochopiteľné, že z praktického hľadiska nemá zmysel používať všetky PBFIC vrstvy. Experimentálne sa ukázalo, že na dostatočne presné výsledky, t.j. takmer nepozorovateľné voľným okom stačia 4 až 8 PBFIC vrstiev podľa typu dát. Kompenzáciou za stratu presnosti je však podstatné skrátenie času výpočtu.

Pri detailnejšom pohľade na túto metódu sa iste neubránime pocitu, že ide o rovnakú metódu akou bola metóda využívajúca bilaterálny grid. Tento pocit je opodstatnený, avšak na rozdiel od metódy bilaterálneho gridu, táto metóda nepodvzorkováva priestorovú doménu, čím je presnejšia a zároveň nevyžaduje rádovo viac pamäte pre výpočet. Ďalším nemenej podstatným rozdielom je nižšia výpočtová náročnosť, kde sa výsledná hodnota bodu bez ohľadu na dimenzionalitu počíta prostou 1D lineárnou interpoláciou na rozdiel od metódy predošlej, ktorá vyžaduje trilineárnu, resp. quadlineárnu interpoláciu pre 2D, resp. 3D dáta.

2.3.3 Rekurzívny bilaterálny filter

Základný princíp rekurzívneho filtrovania sme si už priblížili v kapitole o gaussovej filtrácii. Už vieme, že jednoduchý gaussov filter vypočíta novú hodnotu centrálneho bodu na základe váženého priemeru jeho okolia. Každému bodu okolia teda priradí istú váhu odvodenú od vzdialenosti medzi centrálnym bodom a bodom pre ktorý váhu počítame. Z pohľadu rekurzívneho filtra, však túto váhu nepočítame priamo, ale postupne ju kumulujeme, ako prechádzame obrazom. Výpočet rekurzívneho filtra teda veľkosť okolia nijako neovplyvňuje. Jediné, čo sa s rastúcou σ_S mení, sú jeho koeficienty. Túto skutočnosť môžeme tiež chápať aj tak, že rekurzívny filter ráta vždy s okolím veľkosti celého obrazu. Rekurzívny bilaterálny filter vyplýva práve z tohto princípu, pričom sa súčasne kumuluje priestorová aj intenzitná vzdialenosť. Z praktického hľadiska tak stačí len upraviť výpočet koeficientov a_k, b_k pôvodného gaussovho rekurzívneho filtra.

Túto myšlienku publikoval Qingxiong Yang [6] v roku 2012. Súčasťou publikácie je aj modifikácia metódy autorom nazvaná Gradient domain bi-

lateral filtering (GDBF), ktorá opäť iba zmenou výpočtu koeficientov a_k a b_k odstraňuje schodovité artefakty prejavujúce sa najmä v gradientných oblastiach obrazu (viď odstavec 2.4.1). Výsledné rovnice použitého GDBF prvého rádu majú tvar:

$$b_0 = \frac{1}{\sigma_R} \tag{2.22}$$

$$b_1 = e^{-\sqrt{2}\sigma_S} \tag{2.23}$$

$$a_0 = a_1 = e^{-|I(x)-I(x-1)|b_0} b_1 \tag{2.24}$$

pre koeficienty a

$$I_{ex} = I_{ex}(x)(1-a_0) + I_{ex}(x-1)a_0 \tag{2.25}$$

$$I_{an} = I_{an}(x)(1-a_1) + I_{an}(x-1)a_1 \tag{2.26}$$

$$I^B(x) = \frac{I_{an}(x) + I_{ex}(x)}{2} \tag{2.27}$$

pre samotnú filtráciu.

2.3.4 Ďalšie metódy

Ako sme už uviedli v úvode, navrhovaných aproximácii a vylepšení bilaterálnej filtrácie existuje veľké množstvo. Aktuálne sú za state-of-art okrem bilaterálneho gridu považované metódy gaussovských KD-stromov a mriežky permutostenov.

Metóda KD-stromov vyplynula zo snahy urýchliť Gaussovu filtráciu bilaterálneho gridu. Okrem časovej optimalizácie zároveň odstraňuje aj jej problém s exponencionálnym rastom pamäťovej náročnosti, pri raste dimenzionality vstupných dát. Základnou myšlienkou tejto metódy je prevod týchto obecne N-dimenzionálnych dát na upravené KD-stromy, autormi nazývanými gaussovské KD-stromy, s pomocou ktorých sa potom konvolúcia okolia každého obrazového bodu mení na vyhľadávanie k najbližších susedov tohto stromu. Pre toto vyhľadávanie teraz už v 2D priestore, využívajú metódy mote-carlo a teóriu pravdepodobnosti, a tým zaručujú prístup k potrebným bodom v konštantnom čase. Vstupné dáta sa pri tomto prevode naviac podvzorkovávajú, čím sa ešte viac redukujú požiadavky na potrebnú pamäť.

Metóda permutostenov podobne ako predošlá metóda gaussovské KD-stromov vychádza z bilaterálneho gridu, ktorý modifikuje redukciou pamäťovej náročnosti. Hlavnou ideou je podvzorkovanie priestoru vstupných

dát rozšíreného o dimenziu intenzity do pravidelnej mriežky mnohostenov. Vrcholy týchto mnohostenov pritom spĺňajú podmienku permutostenu, kedy každému vrcholu zodpovedá práve jedna, v rámci mnohostenu jedinečná permutácia celých čísel určených rádom permutostenu. Napríklad, pre permutosten 2. rádu existujú celkom len dve permutácie: $(1, 2)$ a $(2, 1)$. Výsledný permutosten bude teda mať len dva vrcholy, a degraduje na úsečku. Pre permutosten 3. rádu je k dispozícii celkom šesť permutácií: $(1, 2, 3)$, $(1, 3, 2)$, $(2, 1, 3)$, $(2, 3, 1)$, $(3, 1, 2)$, $(3, 2, 1)$, čo vo výsledku predstavuje kocku, ... atď.

Vrcholy permutostenov sa fyzicky ukladajú do štruktúry mriežky a slúžia ako kontrolné body pre nasledujúce fázy: vypĺňania mriežky, konvolúcie a finálnej rekonštrukcie výsledku. Pre každý bod filtrovaného obrazu sa vyhradí samostatná pozícia v mriežke, ktorá sa určuje na základe permutácie okolitých kontrolných vrcholov v podobe hashu pre hashovaciu tabuľku. Významne sa tak šetrí pamäťou pri zachovaní vysokej presnosti výsledku.

Podrobný popis spolu s matematickým základom uvedených metód, by značne presahoval rozsah tejto práce, a tak čitateľa len odkážeme na publikované odborné články autorov [7], [8].

2.4 Aplikácie

2.4.1 Potláčanie šumu

Potláčanie šumu je jedným z hlavných využití bilaterálneho filtrovania. Neúmerne vysoká hodnota priestorovej sigmy však naopak dodatočný šum môže ešte vnášať, a to v podobe zvýraznených ostrých hrán v miestach, kde sa predtým žiadne hrany nenachádzali. Typickým príkladom je filtrácia gradientných plôch, kde po aplikácii bilaterálneho filtra s veľkou σ_S vnikajú schodovité artefakty. Príčina plynie zo základnej rovnice bilaterálneho filtra, podľa ktorej sa rozmazávajú len tie body, ktoré sú si priestorovo aj intenzitne blízke. Pri gradiente, ktorý rozsahovo presahuje σ_R, však vznikajú ostro vyhranené hranice medzi jednotlivými úrovňami intenzít, ktoré sa prejavujú s rastúcou σ_S. V extrémnom prípade, kedy priestorové rozmazanie spriemeruje všetky body danej úrovne, tak môžeme toto rozdelenie pozorovať voľným okom v podobe plôch s uniformným ofarbením.

Uvedený nedostatok odstraňuje napríklad doplnok od Buades et al. publikovaný v článku [9] a naopak niektoré algoritmy ako napríklad štylizácia tento nedostatok využíva.

2.4.2 Zvyšovanie kvality videa

Za predpokladu, že po sebe nasledujúce snímky zobrazujú tú istú statickú scénu, sú všetky zmeny intenzity obrazových bodov spôsobené výlučne šumom. Ten je možné odstrániť jednoduchým spriemerovaním gaussovským filtrom v časovej osi. V praxi ale statické scény vo videu majú len mizivé zastúpenie a uvedené riešenie v pohyblivých scénach vytvára neprijateľné stopy za pohybujúcimi sa objektami tzv. ghosting. Bennett a McMillan vo svojom článku [10] nahradili pôvodný Gaussov filter za bilaterálny, ktorý upravili jednoduchým nahradením priestorového vyhladzovania za vyhladzovanie v čase medzi snímkami videa. Docielili tak vysoko kvalitných výsledkov pri statických aj dynamických scénach bez spomínaných artefaktov.

2.4.3 Zvyšovanie kvality statických snímkov vytvorených pri nízkej úrovni osvetlenia

Autormi tejto metódy sú Eisemann and Durand, ktorí ju publikovali v článku [11] v roku 2004. Využíva techniku kombinovania dvoch snímkov: s bleskom, ktorý je presvetlený, no má vysoký pomer signál-šum, a bez blesku, ktorý je síce zašumený, no zodpovedá reálnemu osvetleniu. Kombinovanie zabezpečuje tzv. cross-bilateral filter. Ide o bilaterálny filter, v ktorom intenzitné váhy počítame z obrazu, ktorým kombinujeme. Matematicky vyjadrené:

$$I^B(x) = \frac{\sum_{y \in N(x)} g_{\sigma_S}(x,y) g_{\sigma_R}\big(D(x), D(y)\big).I(y)}{\sum_{y \in N(x)} g_{\sigma_S}(x,y) g_{\sigma_R}\big(D(x), D(y)\big)}. \tag{2.28}$$

Pre samotné zlepšenie kvality najskôr oba snímky prefiltrujeme štandardným bilaterálnym filtrom. Ten zaručí odstránenie vysokých frekvencii, šumu, a zároveň zaistí, že nedôjde k skresleniu jasovej zložky. Tie použijeme pre výpočet reziduálov s pôvodným obrazom, čím dostaneme časť štruktúry každého obrazu vo forme detailov. Datail získaný zo snímku bez blesku zahodíme, pretože máme k dispozícii detailnejšiu štruktúru zo snímku s bleskom. Následne cross-bilateralnym filtrom skombinujeme normalizovanú jasovú zložku zo snímky bez blesku s detailom so snímky s bleskom. Po aplikovaní farbnej zložky snímku s bleskom dostaneme finálny výsledok.

2.4.4 Tone mapping

Slúži pre premapovanie HDR (high dynamic range) obrazu pre monitory, ktoré HDR natívne nepodporujú. Naivná metóda, ktorá len u lineárne mapuje intenzitný rozsah súčasne odstraňuje aj niektoré detaily pôvodného HDR obrazu. Riešením je uchovať detaily pred samotným premapovaním a neskôr ich do premapovaného výsledku vložiť. Problémom predošlých metód bola buď vysoká časová náročnosť, alebo halo artefakty – metódy využívajúce pre extrakciu detailu Gaussov filter. Oba problémy súčasne rieši nahradenie Gaussovho filtra filtrom bilaterálnym, ktorý na rozdiel od gaussovho nerozmazáva hrany, čím nevytvára spomínané halo artefakty (Durand a Dorsey [12]).

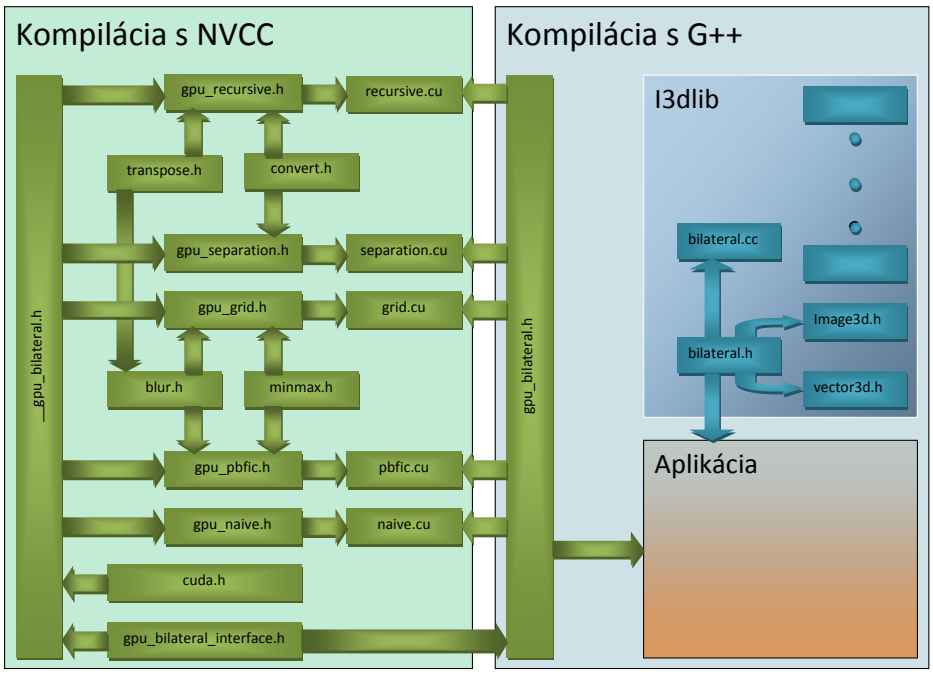
Kapitola 3

Implementácia

3.1 Štruktúra

Celú implementáciu môžeme rozdeliť na dve hlavné časti. Prvú časť predstavuje modul pre knižnicu i3dlib, ktorý okrem bilaterálnej filtrácie implementuje aj metódy pre meranie chybovosti. Menovite ide o metriku signal-to-noise-ratio (SNR) a mean-square-error (MSE), ktoré boli použité pre celú experimentálnu časť práce. Dôvodom pre integráciu všetkých metód do knižnice i3dlib, je najmä možnosť testovania v cieľovom prostredí, t.j. na reálnych dátach bez predspracovania a s využitím dátových štruktúr tejto knižnice. Nepríjemným následkom tohto postupu je nepriama modifikácia zadania, kedy na miesto integrácie vybraných metód je nutné nechcené metódy z knižnice odstrániť.

Druhú časť tvoria jednotlivé metódy, ktoré sú rozdelené do samostatných hlavičkových súborov a rovnako ako CPU varianta, sú vo forme šablón. Keďže štandardný gcc kompilátor nedokáže kompilovať zdrojové súbory so syntaxou CUDA, nebolo možné tieto súbory priamo použiť v i3dlib. Situáciu naviac komplikuje zatiaľ neexistujúca podpora architektúry CUDA touto knižnicou. Pre každú metódu preto existuje pomocný kernel, ktorý už je možné priamo volať ako funkciu z natívneho c/c++. Týmto spôsobom je zároveň zaručená flexibilita implementácie, kde pre adaptáciu na inú knižnicu stačí upraviť, resp. vytvoriť nové wrapper kernely. Každý wrapper kernel pozostáva zo série jednoriadkových funkcií, v rámci ktorých sa inštanciuje konkrétna metóda pre konkrétny dátový typ a presnosť. V stávajúcej implementácii, sú aktuálne podporované všetky formáty, ktoré knižnica i3dlib podporuje.



Obr. 3.1: Štruktúra implementácie spolu so znázornením vzájomných väzieb a závislostí na prekladači.

3.2 Bilaterálny grid

3.2.1 Tvorba gridu

Najpomalšou časťou tejto metódy je tvorba samotného gridu. Z princípu je zrejmé, že nezarovnaným a kolíznym prístupom do globálnej pamäte sa jednoduchým spôsobom nedá vyhnúť. Vstupné dáta by bolo treba najskôr v rozsahu každej bunky gridu zoradiť podľa intenzity, rovnaké intenzity sčítať a až potom uložiť do gridu na príslušné miesto. Vo výsledku by sa vylúčili konflikty prístupu do globálnej pamäte, no nezarovnaný prístup by v menšej miere ostal. Uvedený postup bol experimentálne vyskúšaný, no oproti stávajúcej metóde je v priemere až 3,5 násobne pomalší.

Úvahy teda smerovali k minimalizácii už zmienených problémov prístupu do globálnej pamäte. V zásade prichádzali do úvahy dva rôzne spôsoby dátovej paralelizácie. Prvý by členil dáta spôsobom, jedno vlákno na bunku gridu, čím by sa vylúčili konflikty prístupu, no nezarovnaný

prístup by sa prejavoval hneď dvakrát. Prvýkrát pri načítaní zo vstupu a druhýkrát pri zápise do gridu.

Druhý spôsob paralelizácie, aktuálne používaný, člení vstupné dáta systémom jedno vlákno na pixel, čím sa eliminuje nezarovnaný prístup pri načítaní, no zvýši sa počet konfliktov pri zápise do gridu. Tento zdanlivo horší spôsob sa experimentálne ukázal ako výhodnejší, pretože využíva v plnej miere cache globálnej pamäte, ktorá sa snaží minimalizovať práve nami riešený problém konfliktov. Ďalším opatrením minimalizujúcim problémové prístupy je dátová reprezentácia gridu. Tá pre 2D vstupné dáta predstavuje 3D maticu, kde ako os X chápeme intenzity, ako os Y pôvodnú os X a ako os Z pôvodnú os Y. Analogicky pre 3D vstupné dáta. Takto definovaná štruktúra zaručí dobrú koherenciu dát pri sekvenčnom spracovaní vstupného obrazu.

Opäť experimentálne sa aktuálne používaná metóda ukázala v priemere o viac než $15\,\%$ rýchlejšia ako prvá navrhovaná metóda.

3.2.2 Konvolúcia gridu

Podľa pôvodného článku by sa mal grid pred rekonštrukciou konvolvovať tzv. 5-tap konvolučným jadrom, ktoré predstavuje nasledujúci kernel pre separačnú metódu:

$$[0.0625, 0.25, 0.375, 0.25, 0.0625] \tag{3.1}$$

Z dôvodu väčšej flexibility a možnosti experimentovať, výsledný filter bilaterálneho gridu implementovaný na GPU dovoľuje užívateľovi určiť toto jadro explicitne. Pre CPU verziu je toto jadro fixne nastavné na tvar:

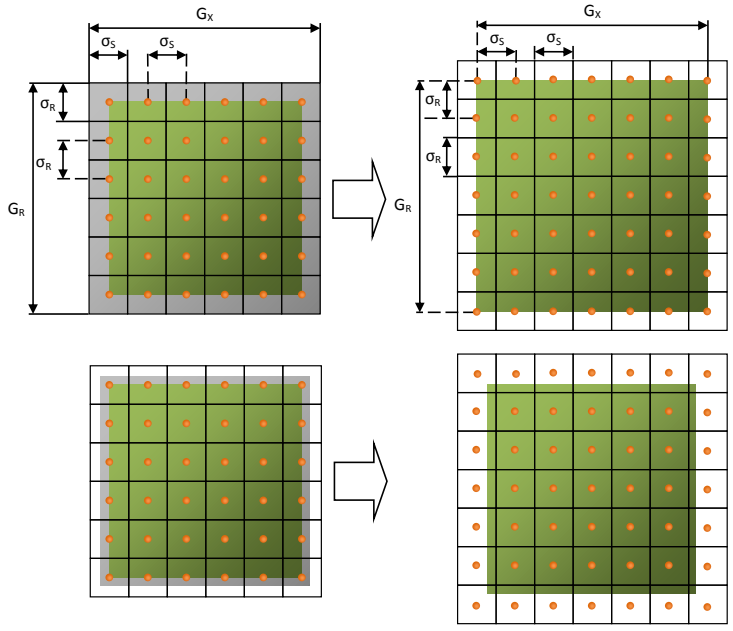
$$[0.1, 3.6788, 10.0, 3.6788, 0.1], \tag{3.2}$$

čo predstavuje jadro pre $\sigma = 1$, ktoré kvalitatívne vykazovalo lepšie výsledky než to, autormi navrhované. Výhodou fixného jadra je možnosť pamäťovej a časovej optimalizácie algoritmu, i keď len pre CPU variantu, kde táto optimalizácia zredukovala pamäťovú komplexitu z 2N na N. Dôvodom vylúčenia GPU optimalizácie je potreba uchovávať hodnotu troch obrazových bodov ako desatinné číslo, naviac doplnenú o váhu. To predstavuje 12 až 24 dodatočných registrov, resp. 48 až 96 bajtov zdieľanej pamäte na vlákno. Tieto nároky výrazne prekračujú limity bežných grafických kariet. Pre ilustráciu, karty s c.c. 2.0 majú obmedzený počet dostupných registrov na 32768 a veľkosť zdieľanej pamäte na $48\,$kiB. Tento limit sa vzťahuje na každý blok vlákien. Pri maximálnej utlizácii máme tak k dispozícii len $\lfloor \frac{32768}{1536} \rfloor = 21$ registrov a $\lfloor \frac{48\,kiB}{1536} \rfloor = 32\,B$ na vlákno.

3.2.3 Rekonštrukcia výsledku

Z dôvodu už zmieneného nedostatku pri interpolácii okrajových bodov pri spätnej rekonštrukcii výsledku, musíme prvý krok, tvorbu gridu, poupraviť. Táto úprava bude nasledujúce.

Pre plne pokrytý grid veľkosti G_R, G_X, G_Y, (G_Z) platí, že sa na všetky bunky gridu namapoval rovnaký počet obrazových bodov ${\sigma_S}^D\sigma_R$, kde D odpovedá dimenzionalite vstupných dát. Vstupné data potom museli mať rozmer $G_X.\sigma_S$, $G_Y.\sigma_S$, $G_Z.\sigma_S$ a počet intenzitných úrovní $G_R.\sigma_R$. Ak rozšírime celý bilaterálny grid v keždej dimenzii o jednu bunku, získame z hľadiska vstupných dát σ_S voľých bodov v každej dimenzii a σ_R intenzitných úrovní. Aby sme tieto voľné body po okrajoch gridu rovnomerne rozdistribuovali, musíme vstupné data do gridu ukladať od pozície $\frac{\sigma_S}{2}$ resp. $\frac{\sigma_R}{2}$. Keďže ide o horný limit zaplnenia gridu, pre ľubovoľné vstupné dáta bude tento posun $O_S \geq \frac{\sigma_S}{2}$ resp. $O_R \geq \frac{\sigma_R}{2}$. Ak ako referenčné body okolia interpolácie výslednej hodnoty budeme používať stredy buniek bilaterálneho gridu, je garantované, že ku každému bodu bude toto okolie existovať. Tento postup pekne znázorňuje nasledujúca ilustrácia.



Obr. 3.2: Odstránenie problému interpolácie okrajových bodov: Zelenou sú znázornené vstupné dáta, oranžové body reprezentujú referenčné body interpolácie a sivá reprezentuje oblasť dát, pre ktoré nie sme schopní rekonštruovať výslednú hodnotu, v tomto prípade bilineárnou interpoláciou.

Celý výpočet sa presunul do globálnej pamäte, čím odpadá celá réžia textúr. Oproti autormi navrhovanej implementácii tak výsledné spomalenie odpovedá výpočtu interpolácie v jednej dimenzii, bez prístupu do globálnej pamäte. To predstavuje operácie v podobe dvoch násobení a dvoch sčítaní, čo pre GPU predstavuje inštrukcie s najväčšou dátovou priepustnosťou.

Vlákna sú pri rekonštrukcii organizované po štyroch pre 2D, resp. po ôsmych pre 3D vstupné dáta. Každé vlákno najskôr vyčíta jeden obrazový bod, z ktorého určí svoj index do gridu. Tento prístup do globálnej pamäti je zarovnaný. Následne každé vlákno z vypočítaného indexu vyčíta vždy dvojicu susedných buniek, ktoré lineárne interpoluje a výsledok uloží do zdieľanej pamäte. Tento prístup už zarovnaný nie je, no z hľadiska veľkosti dát, ktoré vyčíta jedno vlákno (4xfloat 16 B, 4xdouble 32 B), a dobrej koherencie dát, ide stále o prístup s minimálnym spomalením. Nasleduje séria dvoch, prípadne troch lineárnych interpolácii nad zdieľanou pamäťou, kde každý krok rieši vždy polovica vlákien z predošlého kroku. Poslednú, výslednú, hodnotu tak počíta a na výstup zapisuje jediné vlákno.

Priestorová náročnosť tejto metódy je lineárne závislá na σ_R, no kvadraticky až kubicky na σ_S. Preto nechávame primeranú voľbu oboch týchto parametrov na úsudku užívateľa.

3.3 PBFIC

Metódu intenzitných rezov PBFIC z hľadiska implementácie môžeme chápať ako bilaterálny grid, v ktorom je fixne nastavená σ_S a intenzitná dimenzia. Odpadá tak problém nezarovnaného a kolízneho prístupu do globálnej pamäte, ktorý bolo nutné riešiť pri tvorbe a rekonštrukcii bilaterálneho gridu. S výhodou sme tiež využili degradáciu 3D až 4D gridu na 2D PBFIC rez, pre ktorý na rekonštrukciu výsledku jedného obrazového bodu stačí lineárna interpolácia a jediné vlákno. Dátová priepustnosť tak stúpla štvor až osem-násobne.

Posledným neriešeným spomalením tak ostava krok konvolúcie, pre ktorý sme využili už implementovanú Dericheho aproximáciu Gaussovho filtra modulu `blur.h`. Ten poskytuje dostatočne presné výsledky v lineárnom čase. V rámci experimentov, bol do tohto modulu implementovaný aj jednoduchý rekurzívny box filter, ktorý však pre malé σ_R vytvára vizuálne neprijateľné artefakty. Rýchlostne však prekonáva Dericheho aproximáciu a pri dostatočne vysokom počte rezov (8 a viac), poskytuje uspokojivé výsledky PSNR > 40 dB. Už zmienené artefakty sú dôsledkom zaokrúhľovacích chýb, ktoré vznikajú pri kumulácii váh a následnom prie-

merovaní – základný princíp box filtra.

Podobne ako pri bilaterálnom gride, je aj táto implementácia rozdelená na tri kroky: krok tvorby, konvolúcie a rekonštrukcie. Oproti publikovanej verzii, kde sa v rámci kroku tvorby rezov počítajú všetky rezy súčasne, t.j. vstupným obrazom sa prechádza len raz, prezentovaná implementácia vytvára tieto rezy postupne.

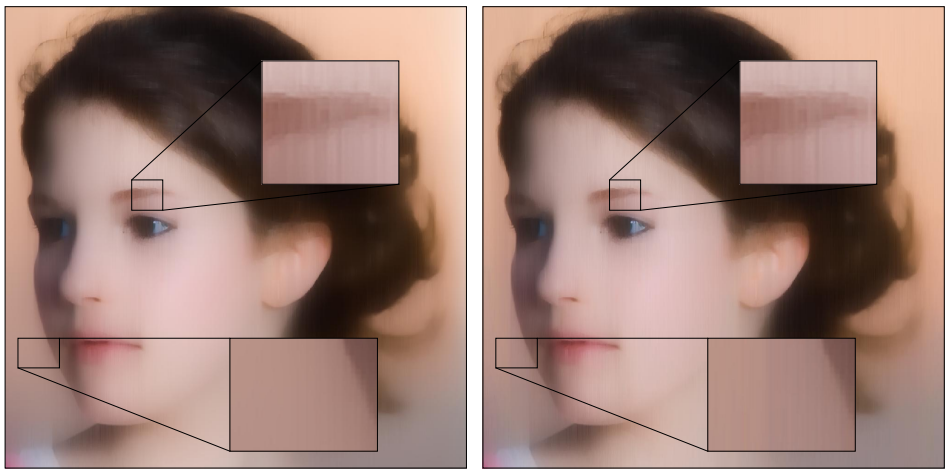
Dôvodom pre toto výrazné spomalenie je významná úspora potrebnej pamäte a s tým súvisiaca schopnosť filtrovať veľké medicínske dáta. Celý postup filtrácie potom prebieha predpočítaním nultého rezu, s referenčnou intenzitou rovnou nule, ktorý slúži ako spodná hranica lineárnej interpolácie v poslednom kroku filtrácie. Následne sa iteruje cez jednotlivé intenzity, pre ktoré sa počítajú zodpovedajúce rezy, horné hranice lineárnej interpolácie. Na konci každej iterácie sa potom rez reprezentujúci spodný limit zahodí a nahradí sa práve vypočítaným rezom. Z tohto postupu je zrejmé, že celkový počet potrebných rezov je minimálne dva. Operácie zahodenia a výmeny nad rezmi fyzicky dáta nemažú, ani nekopírujú. Rovnaký výsledku sa dosahuje ukazovateľovou aritmetikou.

3.4 Rekurzívny bilaterálny filter

Navrhnutá GPU implementácia priamo vychádza z autorom zverejnenej CPU implementácie s istými modifikáciami. Asi najvýznamnejšou je redukcia celkového počtu premenných a počtu pomocných zásobníkov zo šiestich na tri, a po ďalšej optimalizácii na dva. Touto optimalizáciou rozumieme úpravu samotného algrotimu, kedy pre antiextenzívny a každý ďalší prechod dátami neuvažujeme pre výpočet σ_R pôvodný vstupný obraz, ale aktuálne vypočítaný medzivýsledok. Dôsledkom tejto modifikácie je okrem pochopiteľnej trojnásobnej redukcie potrebnej pamäte aj istá degradácia kvality výsledku v podobe zvýraznenia, už v pôvodnej metóde prítomných, artefaktov. Tie sa však prejavujú len pri vysokých hodnotách σ_R, viď Obr.3.3.

Podľa autora metódy sa všetky koeficienty a_k dajú predpočítať do pomocnej tabuľky. Z praktického hľadiska to však už pre 8-bitové vstupné dáta predstavuje 1 kiB (presnosť float), resp. 2 kiB(presnosť double) dodatočnej pamäte. Veľkosť tejto tabuľky navyše nie sme schopní dopredu odhadnúť vzhľadom na dopredu neznámy intenzitný rozsah vstupu. Využitie globálnej pamäte, či už v podobe konštantnej, alebo priamo, pre túto tabuľku neprichádza v úvahu. Hlavným dôvodom je nezarovnaný prístupu a v prípade veľkej variancie intenzít, aj nedostatok pamäte. Koeficienty

a_k sa preto počítajú v hlavnom cykle filtra pre každý pixel a pre každý prechod zvlášť. Okrem drobných optimalizácii zbytok implementácie zodpovedá publikovanému algoritmu. Ďalšie globálne optimalizácie, ktoré ovplyvňujú aj túto metódu, sú uvedené v ďalšej kapitole s názvom Optimalizácie.



Obr. 3.3: Príklad artefaktov spôsobených vysokou hodnotou σ_R. Spodný detail poukazuje na zvýraznenie chýb optimalizovanej varianty, ktorá sa v tomto prípade prejavila skokovitou zmenou intenzity. Druhý detail len poukazuje na celkovú chybovosť metódy.

3.5 Modul blur

Pozostáva z dvoch častí. Prvá implementuje gaussovský filter dvoma metódami, s využitím vlastnosti separability a Dericheho rekurzívnou metódou. Druhá časť predstavuje boxfilter implementovaný výlučne rekurzívnou metódou. Separačná metóda bola do tohto modulu zaradená čisto z experimentálnych dôvodov. Na rozdiel od Dericheho metódy, ktorá Gaussov filter len aproximuje, metóda separability vracia vždy presné výsledky. Umožňuje tak jednotlivé metódy využívajúce vo svojich výpočtoch gaussov filter testovať do akej miery je presnosť ich výstupu závislá na tomto filtri.

Modul je primárne určený pre metódu bilaterálneho gridu a PBFIC, časovo kritickým je však len pre metódu rezov PBFIC, kde predstavuje podľa počtu PBFIC 60 až 100 % celkového behu metódy. Napriek istej špecializácii pre tento účel je modul stále dobre použiteľný aj pre iné aplikácie.

Keďže sú jednotlivé metódy viackrokové, t.j. iterujú k finálnemu výsledku, pre výpočet potrebujú minimálne dva zásobníky. Jeden pre aktuálny výsledok medzikroku a druhý pre ukladanie výsledku kroku počítaného. Pre zaistenie čo najmenšej pamäťovej náročnosti sa tak namiesto alokácie druhého pomocného zásobníka používa pamäť vstupných dát. Toto riešenie je zároveň dôvodom, prečo je vstup a výstup predávaný v podobe ukazovateľa na ukazovateľ.

3.6 Modul convert

Ide o jednoduchý modul poskytujúci grafickou kartou akcelerované pretypovanie skalárneho poľa hodnôt. Ako parametre vyžaduje ukazovateľ na vstupné pole, rozmery tohto poľa, až do štyroch dimenzii, a ukazovateľ na výstupné pole. Pôvodnou snahou bolo typovú konverziu vykonávať in situ, t.j. na mieste, bez použitia pomocného zásobníka, no nemožnosť sychronizácie blokov a s tým súvisiaci race condition problém viedol na aktuálne riešenie so sekundárnym zásobníkom, v podobe výstupného poľa. Primárny účel tohto modulu je poskytnúť plnú presnosť metódy rekurzívneho a separačného bilaterálneho filtra, viď nasledujúca kapitola.

3.7 Optimalizácie

Pre optimalitu prístupu do globálnej pamäte, zjednodušenie implementácie, no najmä pre čo najmenšiu pamäťovú náročnosť, rekurzívna a separačná metóda počas svojho výpočtu používa pomocný zásobník rovnakého dátového typu, akého sú vstupné dáta. Jednotlivé výpočty sú však počítané s užívateľom zadanou presnosťou. V zásade je podporovaná presnosť jednoduchá – float, prípadne dvojnásobná – double.

Pre zdôraznenie významu priestorovej úspory, si ako vstup predstavme 3D šedotónny dáta o rozmeroch 1024x1024x256 formátu bezznamienkový byte. Ak by sme chceli použiť plnú presnosť, t.j. presnosť typu double, potrebovali by sme pomocný zásobník veľkosti osemkrát väčšej oproti vstupným dátam. To v tomto príklade predstavuje nárast potrebnej pamäte o 2 GiB, teda z 256 MiB na 2.25 GiB. Pre väčšinu mainstreamových grafických kariet je už tento nárast neúnosný. Výkonnejšie grafické karty ho však ešte zvládajú. S jedným pomocným zásobníkom sme však stále obmedzení len na použitie naivnej metódy bilaterálneho filtra. Ak by sme chceli využiť rekurzívnu, resp. separačná metóda potrebujeme dva zásobníky s plnou presnosťou. To aj pri šetrení miestom v podobe znovupoužitia

vstupných dát ako zásobníka, po pretypovaní, predstavuje nárast o ďalších 1.75 GiB. Celkovo teda potrebujeme 4 GiB pamäte, čo už zvládajú len špičkové, poprípade špecializované grafické karty akými sú TESLA, GTX Titan a podobne. Naopak, pri stávajúcej implementácii budú požadované nároky len 512 MiB.

Nevýhodou tohto riešenia je okrem nutnej násobnej konverzie pre každý spracovávaný obrazový bod najmä strata presnosti. Tá závisí od použitého dátového typu vstupných dát a činí $\frac{1}{2^{n+1}}$, na konverziu, kde n predstavuje bitovú veľkosť použitého dátového typu. Pre dátový typ float je n rovné veľkosti mantisy, t.j. 24 bitov. Nasleduje tabuľka maximálnych chýb spôsobených typovou konverziou pre obe optimalizované metódy:

	2D			
typ	char	short	long	float
separabilita	1/512	1/128k	1/8G	1/32M
rekurzívna metóda	3/256	3/64k	3/4G	3/16M
	3D			
typ	char	short	long	float
separabilita	1/128	1/32k	1/2G	1/8M
rekurzívna metóda	5/256	5/64k	5/4G	5/16M

Pre 2D vstupné dáta je táto optimalizácia pamäťovo bezvýznamná, no vďaka lepšej koherencii vstupných dát významným spôsobom znižuje počet I/O operácii globálnej pamäte, čo sa v konečnom dôsledku prejavuje časovou úsporou v priemere až o $20\,\%$.

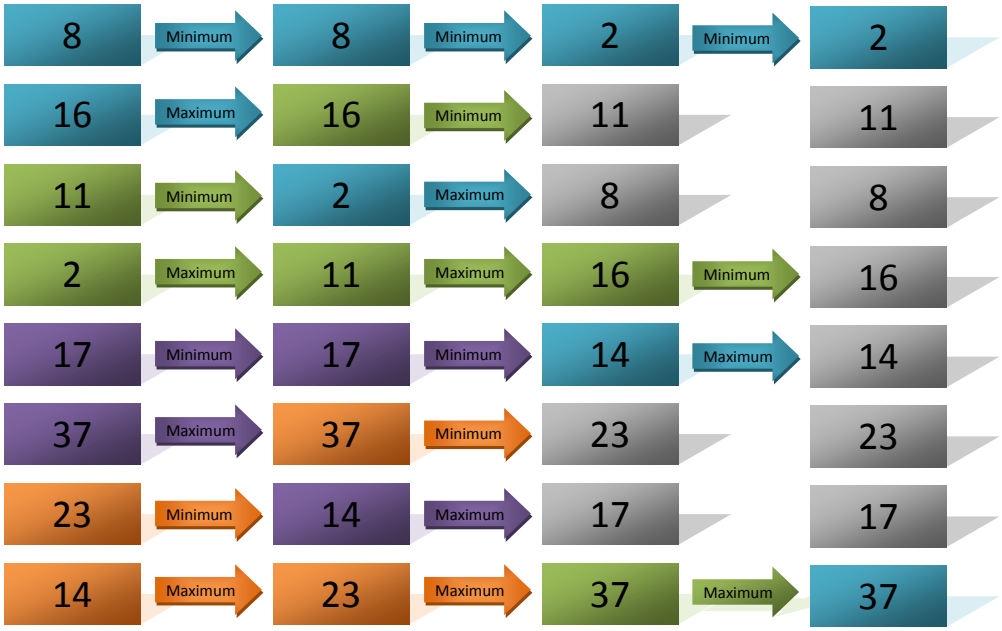
Kvôli povahe filtrovaných dát a dôraze na presnosť výsledkov boli metódy, ktoré túto optimalizáciu umožňujú obohatené o nepovinný konfiguračný parameter, ktorý umožňuje túto optimalizáciu potlačiť. Implicitne je jeho hodnota nastavená na `USE_DEFAULT_FLAGS` (hodnota 0), čo zodpovedá optimalizácii metódy. Pre jej potlačenie je nutné tento parameter nastaviť na hodnotu `DISABLE_OPTIMISATIONS`, kedy sa vstupné dáta pred samotným filtrovaním pretypujú na požadovanú presnosť a po filtrácii sa opäť pretypujú naspäť na pôvodný typ o čo sa stará v kapitole 3.6 spomínaný modul convert.

3.8 Minmax

Ďalšiu optimalizáciu poskytuje modul minmax.h, ktorý implementuje paralelné vyhľadávanie minimálnej a maximálnej hodnoty v čase $log(n)$. Znalosť minima a maxima vstupných dát umožňuje podstatne navýšiť presnosť

výpočtu metód PBFIC a bilaterálneho gridu. U druhej zmienenej metódy naviac redukuje významnou mierou pamäťové nároky. Obe optimalizácie využívajú, pre 3D obrazové dáta typickú, charakteristiku histogramu, ktorý je obyčajne veľmi úzky a často nepokrýva ani $20\,\%$ rozsahu intenzít.

Algoritmus pracuje na princípe divide and conquer, čiže v prvom kroku sa vstupné dáta virtuálne rozdelia do súvislých blokov veľkosti dvoch prvkov, v ktorom každé vlákno určí minimálnu a maximálnu hodnotu, ktorú umiestni na prvú resp. poslednú pozíciu aktuálneho bloku. Z praktického hľadiska tento krok predsatuje jediné porovnanie a prípadnú výmenú prvkov ak je prvý prvok väčší ako druhý, v bloku posledný. Nasledujúceho kroku sa opäť zúčastnia všetky vlákna a porovnanie prebieha analogicky s prvým krokom. Rozdielom je prerozdelenie vlákien na dve rovnako veľké skupiny, kde prvá skupina bude operovať nad minimálnymi a druhá nad maximálnymi prvkami predošlého kroku. Opäť každé vlákno operuje nad dvojprvkovým blokom dát a opäť ide len o jediné porovnanie, prípadne výmenu dvoch prvkov. Výsledkom druhého kroku je množina štvorprvkových blokov, kde na začiatku je prvok minimálny a na konci prvok maximálny. Prvky medzi minimom a maximom sú hodnotou vždy menšie, rovné maximu a väčšie rovné minimu, čím sa nekvalifikujú do ďalšej iterácie algoritmu a zo vstupných dát vypadávajú. Celý algoritmus znázorňuje nasledujúca ilustrácia:



Obr. 3.4: Farby prvkov zodpovedajú konkrétnemu vláknu a sivé prvky budú po poslednej iteracií odstránené.

3.9 Modul transpose

Modul transponovania slúži pre transpozíciu jedno až štvorrozmerných dát. Vždy ide o jednoduchú transpozíciu dvojice rozmerov, kde sa ľubovoľná dimenzia zamieňa s y-ovou dimenziou. Výber y-ovej dimenzie úzko súvisí s organizáciou vlákien a zarovnaným prístupom do globálnej pamäte najmä u rekurzívych metód konvolúcie a bilaterálnej filtrácie.

Problémom tohto prístupu je obmedzenie veľkosti dimenzie bloku, ktorá je pre c.c. < 2.0 512 a pre c.c. ≥ 2.0 1024. Navyše ak zvolíme tento blok príliš veľký, pre rozmerovo malé dáta, resp. dáta, ktorých veľkosť nie je zarovnaná na veľkosť bloku, bude veľká časť vlákien neaktívna z dôvodu prístupu mimo alokovanú pamäť. Experimentálne vyšla ako najvhodnejšia veľkosť x-ovej dimenzie bloku 256. Pre 2D dáta ide o dostatočne malú hodnotu, vzhľadom k tomu, že 2D dáta bývajú rozmerovo oveľa väčšie než 3D dáta. Pri 3D dátach môžeme rozšíriť pôvodný 1D blok o ďalšiu di-

menziu, čím sme schopní paralelne spracovávať riadky, v rôznej hĺbke. Podľa rovnice 1.1 pre GPU s c.c. 2.0 vychádza pre maximálnu utilizáciu 6 blokov na multiprocesor, čo pri 14 multiprocesorch predstavuje 84 blokov a teda 21504 vlákien. Pre maximálnu utilizáciu teda musia mať vstupné 2D dáta všetky rozmery aspoň veľkosť 21504 pixelov, pre 3D stačí, aby mali veľkosť bloku, čiže 256. Paradoxne tak má výpočet nad 3D dátami vyššiu dátovú priepustnosť než výpočet nad 2D dátami, čo názorne ilustrujú grafy výsledkov v kapitole 4.

Z dôvodu častého volania funkciami metód filtrov ide o jednu z časovo kritických častí kódu. Hlavným cieľom pri implementácii tohto modulu bola teda časová optimalita. Tá je dosiahnutá vhodnou voľbou bloku a s využitím zdieľanej pamäte pre zarovnaný prístup do globálnej pamäte, ktorá predstavuje úzke hrdlo. Vhodným blokom rozumieme blok veľkosti 16x16, ktorý je univerzálnym z hľadiska c.c grafickej karty. Dosiahnutú časovú optimalitu dobre ilustruje nasledujúca tabuľka, pre ktorú boli použité dva testovacie obrazy veľkosti jedného megapixelu (1000x1000). Prvý vo formáte 8-bit, šedotónny, druhý 24 bit RGB.

RGB			Grayscale		
YXZW	XZYW	XWZY	YXZW	XZYW	XWZY
0,11ms	0,07ms	0.8ms	0.06ms	0.05ms	0.05ms

Ako vidieť, aj najhorší prípad, 0.11ms, predstavuje pre najrýchleší, rekurzivný filter, pri 1 MiB vstupe len asi 2 % z celkového behu.

Kapitola 4

Výsledky a porovnanie

4.1 Presnosť metód

4.1.1 Metrika

Pre meranie presnosti výstupu jednotlivých metód je ako referenčný výsledok, tzv. ground-truth, použitý výstup naivnej CPU metódy s presnosťou double. Ako metrika presnosti je použitý pomer špička-špička signálu a šumu v literatúre označovaný ako PSNR (peak signal to noise ratio). PSNR sa určuje na základe ďalšej metriky označovanej MSE (mean square error), pre ktorú platí:

$$MSE = \frac{\left(I_{REF}(x) - I(x)\right)^2}{X_{Dim}.Y_{Dim}.Z_{Dim}.\tau}, \tag{4.1}$$

ide teda o sumu štvorcov všetkých odchýlok od referencie, ktorá sa normalizuje veľkosťou dát. Prirodzene, teda porovnávané a referenčné dáta musia mať rovnaké rozmery. Hodnota τ v tomto predpise predstavuje normalizačnú konštantu, ktorá zodpovedá maximálnej intenzite vstupných dát.

Reciprokáciou MSE dostávame PSNR, ktoré sa ešte zvykne prevádzať do logaritmickej mierky s jednodkou decibel podľa nasledujúceho predpisu:

$$PSNR = 10.\log_{10} MSE^{-1}, \tag{4.2}$$

Pre RGB sa PSNR počíta z modifikovaného MSE, kde sa štandardným spôsobom určí MSE pre každý kanál zvlášť a výsledné MSE_R, MSE_G, MSE_B sa potom priemerujú váženým priemerom s váhami zodpovedajúcimi výpočtu intenzity z RGB podľa štandardu PAL, resp. NTSC:

$$MSE = 0.2989.MSE_R + 0.5870.MSE_G + 0.1141.MSE_B \tag{4.3}$$

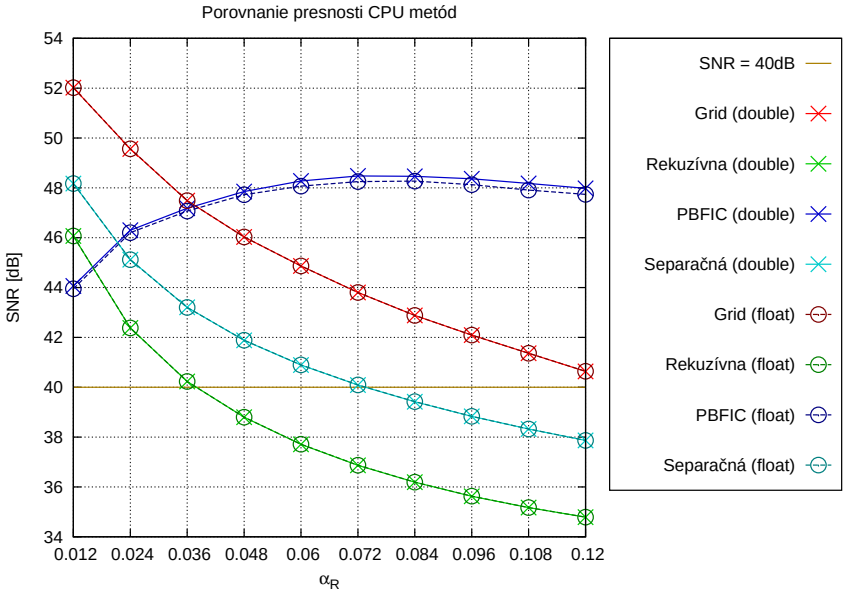
Intuitívne, čím je rozdielnosť dát menšia, tým väčšia je hodnota PSNR a tým kvalitnejšie sú porovnávané dáta. Za ľudským okom len ťažko rozpoznateľné rozdiely sa považuje PSNR > 40 dB.

4.1.2 Metodika merania

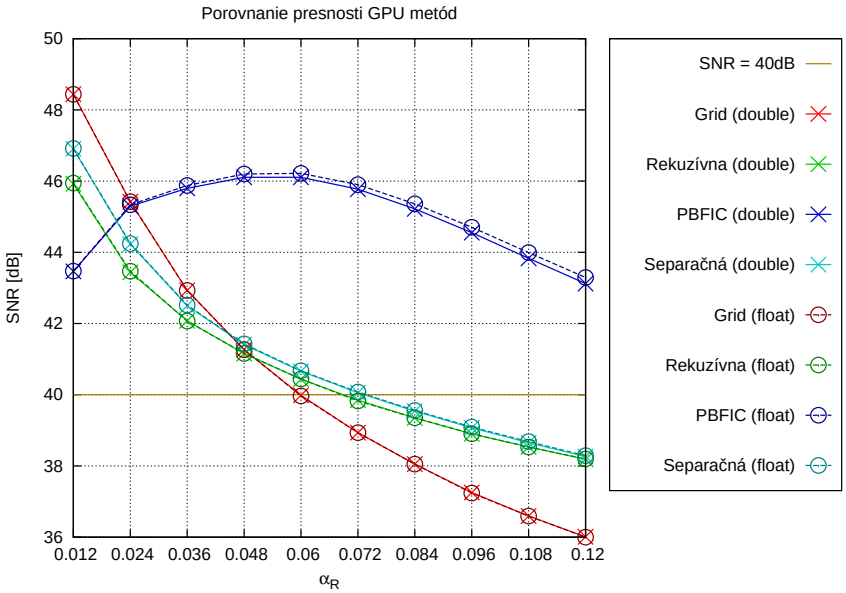
Meranie presnosti metód prebiehalo v dvoch krokoch. V prvom sme zafixovali veľkosť σ_S na hodnotu 5.0 a variabilnou bola σ_R v rozsahu 0.012 až 0.12 s krokom 0.012. V druhom sa sigmy vymenili, t.j. fixná bola σ_R s hodnotou 0.036 a variabilnou bola σ_S s hodnotami v rozsahu 1.0 až 10.0 s krokom veľkosti 1.0. V oboch prípadoch boli testované všetky CPU aj GPU metódy v jednoduchej a dvojitej presnosti. Pri výbere testovacích dát bolo snahou čo najviac potlačiť závislosť presnosti na ich type a zároveň zastúpiť všetky typy najčastejšie používané v cieľovom prostredí, Ústave spracovania biomedicínských dát MU. Ako testovacie dáta bola teda použitá nasledujúca sada 2D a 3D obrazov:

- 352x288, RGB, 8b
- 1000x666, RGB, 8b
- 1024x768, RGB, 8b
- 1000x1000, RGB, 8b
- 600x600, RGB, 8b
- 1600x800, RGB, 8b
- 2400x2400, šedotónny, 8b
- 32x32x32, šedotónny, 8b
- 512x512x5, šedotónny, 8b
- 512x512x5, šedotónny, 16b
- 512x512x20, šedotónny, 16b

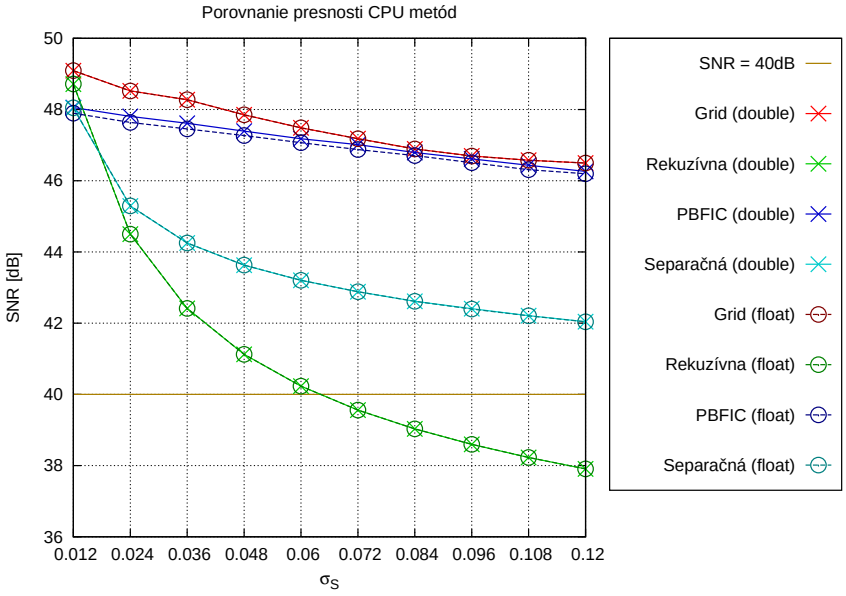
Výsledné PSNR sa potom získalo aritmetickým priemerom dielčích výsledkov, a je znázornené v nasledujúcich grafoch:



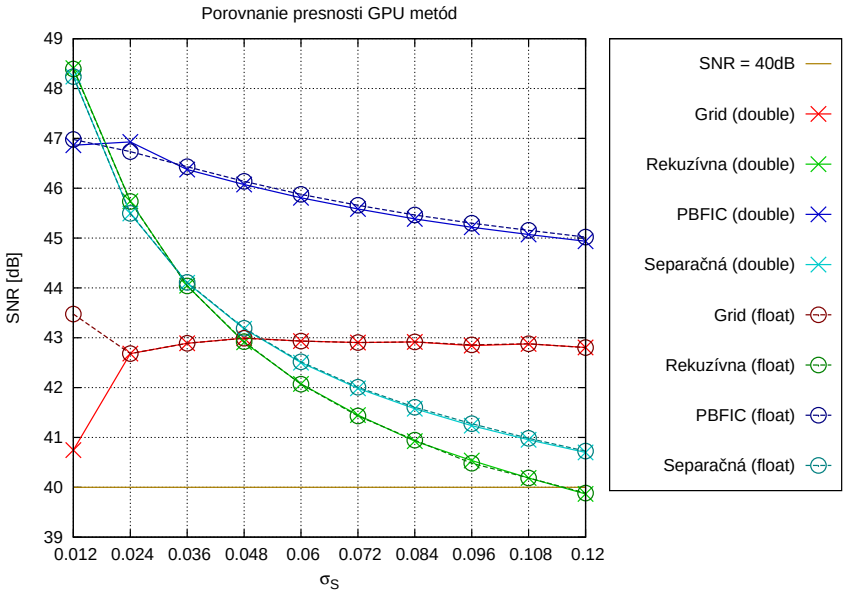
Graf 4.1



Graf 4.2



Graf 4.3



Graf 4.4

4.1.3 Výsledky

Pri metóde PBFIC je pokles PSNR pre malú σ_R spôsobený už v kapitole 3.3 zmienenými, zaokrúhľovacími chybami. Na rozdiel od box filtru sa však tieto chyby neprejavujú viditeľnými artefaktami obrazu. Počiatočný pokles PSNR u bilaterálneho gridu v grafe 4.4 je zas výsledkom nepresnosti merania, kedy sa pre enormné pamäťové nároky nepodarilo časť meraní vôbec uskutočniť. Celkový pokles presnosti s rastúcimi sigmami prisudzujeme faktu, že metódy implementujú len aproximácie bilaterálneho filtra, a je preto pochopiteľné, že s rastúcim efektom filtrovania sa výsledky aproximácie zhoršujú. K výraznejšiemu poklesu rekurzívnej metódy tiež prispieva vylepšenie odstraňujúce schodovité artefakty, ktoré sú súčasťou referenčnej naivnej metódy.

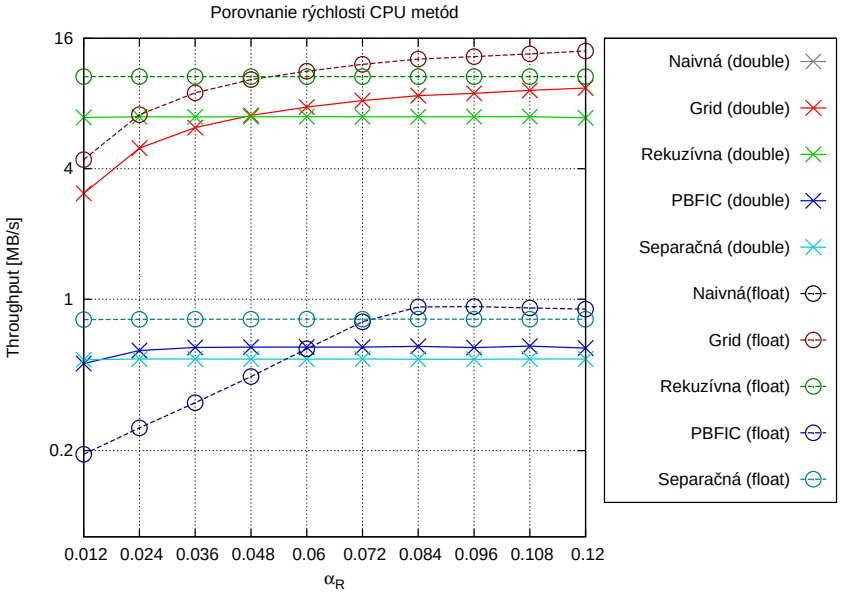
4.2 Rýchlosť metód

4.2.1 Metrika

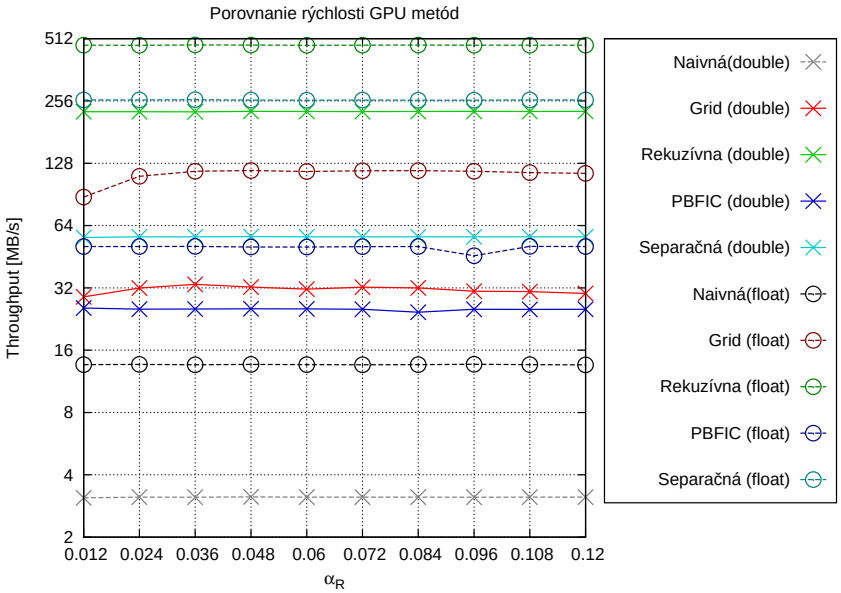
Rýchlosť v tejto podkapitole budeme chápať ako množstvo výsledných dát získané za časový interval jednej sekundy, tiež označované ako priepustnosť. Voľba tejto metriky potláča veľkú rozdielnosť výsledkov meraní v závislosti na veľkosti vstupných dát. Hodnotu priepustnosti teda získavame podielom veľkosti dát a potrebného času pre ich spracovanie.

4.2.2 Metodika merania

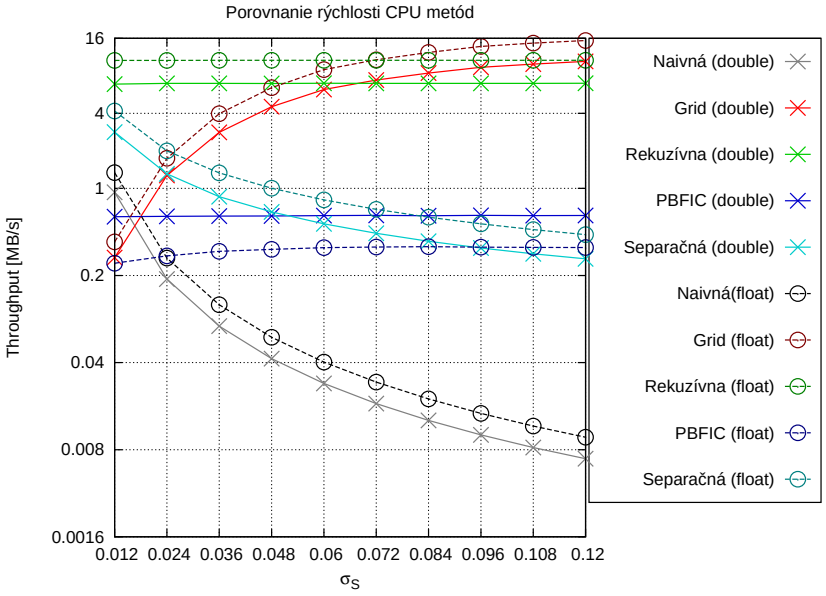
Obdobne ako v meraní presnosti aj pri meraní rýchlosti sme meranie rozdelili do niekoľkých krokov. Celkovo sú tri, pričom v prvých dvoch sme postupovali analogicky s meraním presnosti spolu s nastavením σ_R a σ_S. Posledný, tretí krok, kedy boli zafixované obe sigmy na hodnoty 6.0 pre σ_S, resp. 0.06 pre σ_R a variabilná bola veľkosť obrazu, sme z dôvodu závislosti merania na dimenzionalite dát rozdelili do ďalších dvoch podkrokov. V prvom sa meria závislosť priepustnosti od veľkosti 2D dát, v druhom od veľkosti 3D dát. Tieto veľkosti sa pritom pohybovali v rozmedzí 200x200 až 2000x2000 s krokom 200x200 pre 2D a 16x16x16 až 160x160x160 s krokom 16x16x16 pre 3D obraz. V oboch prípadoch išlo o šedotónne dáta s bitovou hĺbkou 8 bitov pre 2D resp. 16 bitov pre 3D variantu. Výsledky meraní sú znázornené v nasledujúcich grafoch.



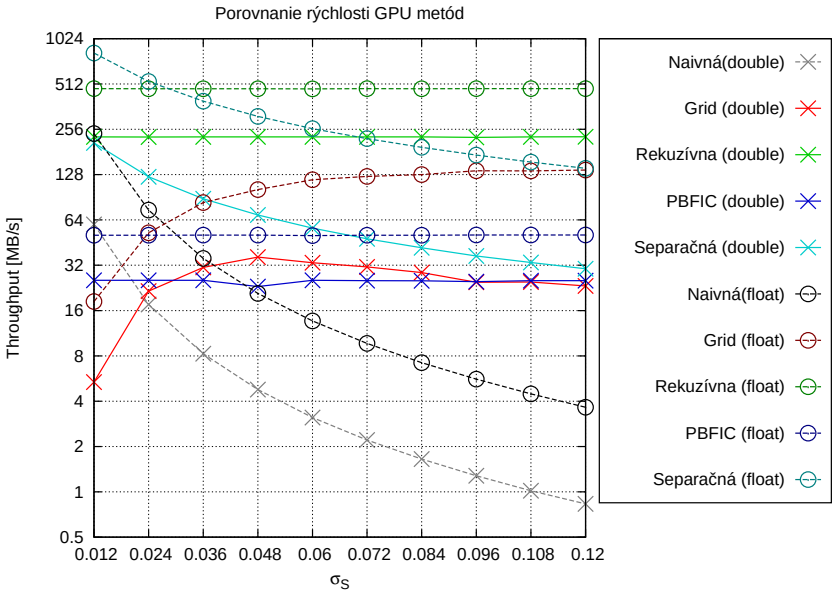
Graf 4.5



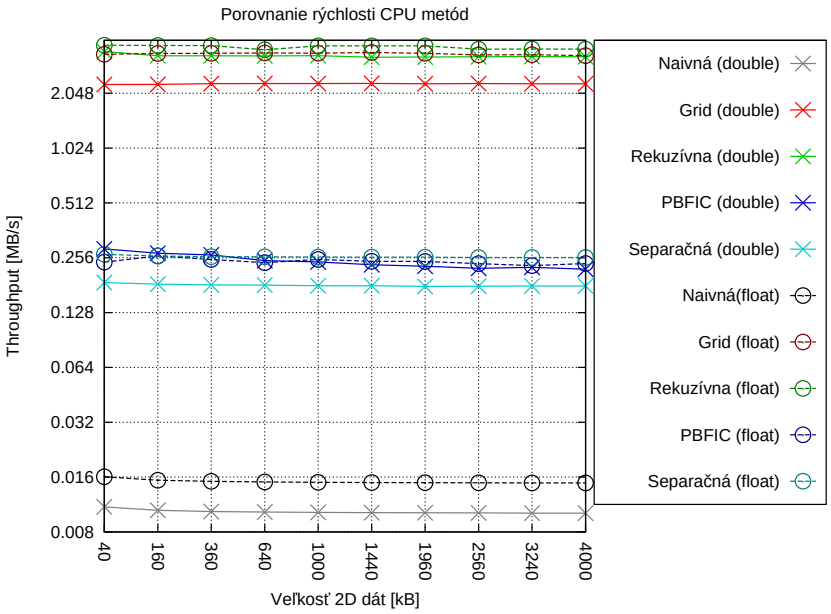
Graf 4.6



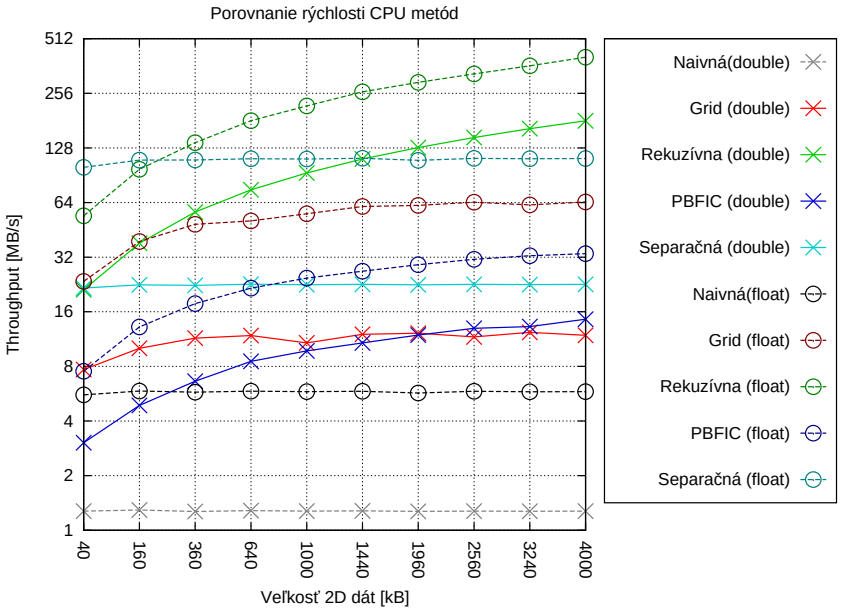
Graf 4.7



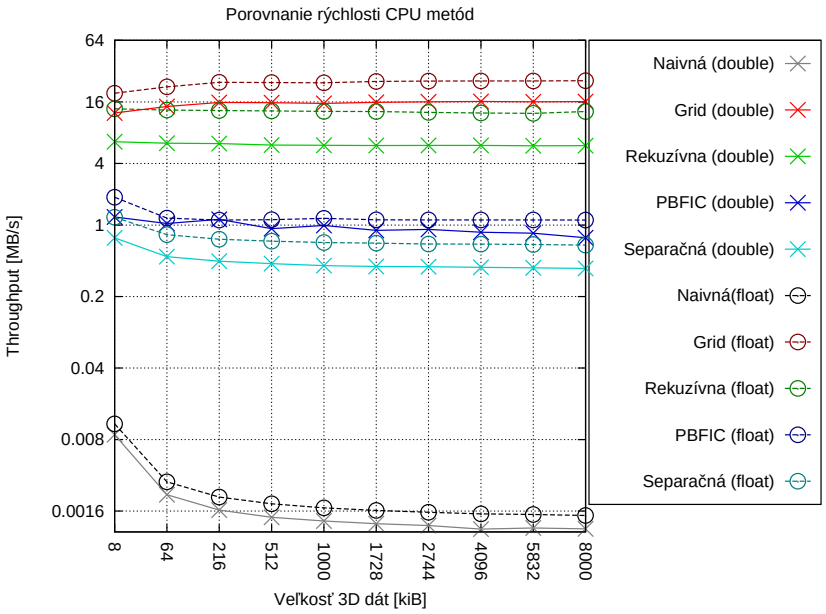
Graf 4.8



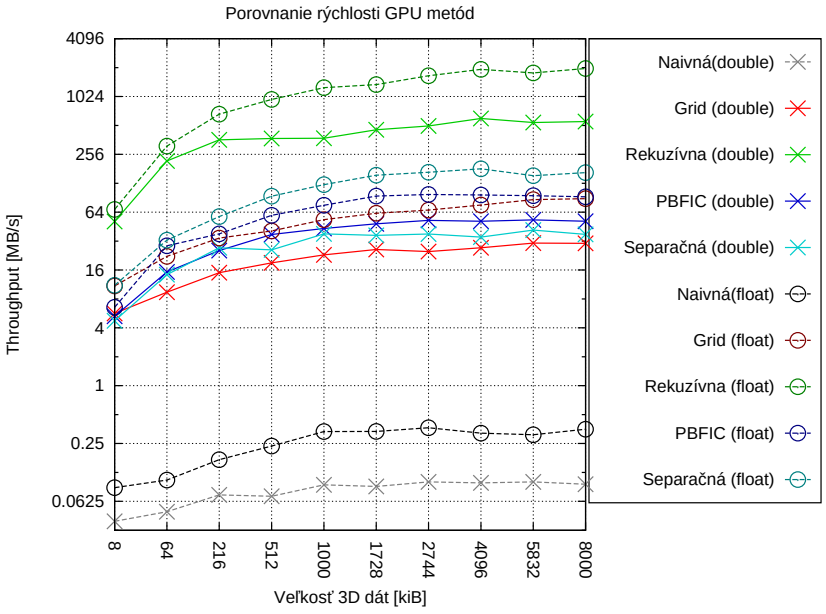
Graf 4.9



Graf 4.10



Graf 4.11



Graf 4.12

4.2.3 Výsledky

Z grafov 4.5 a 4.6 vidieť, že všetky metódy okrem bilaterálneho gridu sú rýchlosťou nezávislé na parametri σ_R. Dôvodom je skutočnosť, že σ_R nijakým spôsobom neovplyvňuje veľkosť konvolučného jadra filtrácie. Toto tvrdenie platí aj pre metódu bilaterálneho gridu a to aj navzdory stúpajúcej tendencii jej krivky. Tá je spôsobená nepriamou úmerou veľkosti intenzitnej dimenzie gridu od σ_R, čo vidieť z rovnice 2.19. Z praktického hľadiska tak σ_R nepriamou úmerou určuje veľkosť celého gridu, čím znižuje, alebo zvyšuje objem dát ktoré je nutné v rámci kroku konvolúcie gridu spracovať.

Rovnaké zdôvodnenie správania sa krivky bilaterálneho gridu nachádza uplatnenie aj pre grafy 4.7 a 4.8, kde je táto závislosť naviac umocnená rastúcou dimenzionalitou vstupných dát. Opäť plynie z rovnice 2.19. Krivky naivnej a separačnej metódy svojim priebehom zodpovedajú predpokladanej zložitosti $O(n^2)$, resp. $O(n^3)$, podľa toho či filtrujeme 2D alebo 3D dáta. Príčinou je kvadratický až kubický nárast veľkosti konvolučného jadra s rastúcou σ_S. Rovnako predpokladaný, konštantný priebeh majú všetky rekurzívne metódy, resp. metódy PBFIC, ktoré pre svoj výpočet využívajú rekurzívne počítanú gaussovú filtráciu a teda sú od σ_S nezávislé.

4.3 Testovacia zostava

Pre účely všetkých meraní bol využitý server melete, ktorý má nasledujúce hardwarové parametre:

- Procesor: Intel Core 2 Duo Q6600, 2.4 GHz, Kentsfield
- Pamäť: 8 GB RAM
- Grafická karta: GeForce GTX 470 (c.c. 2.0)

Záver

Výsledkom tejto diplomovej práce, je komplexný prehľad troch vybratých metód bilaterálnej filtrácie, doplnených o experimentálnu metódu aproximujúcu túto filtráciu, s využitím separability, spolu s ich GPU implementáciami. Práca do detailov rozoberá algoritmy týchto metód, ich prednosti a nedostatky, pre ktoré potom v implementačnej časti navrhuje a neskôr aj imeplementuje optimalizácie. Ide predovšetkým o optimalizácie s ohľadom na pamäťové nároky, ktoré bolo nutné pre konečné nasadenie, spracovanie objemných 3D medicínskych dát, čo najviac minimalizovať. Dôraz sa však kládol aj na optimalitu z hľadiska rýchlosti, čo sa vo výsledných implementáciach prejavilo znížením časových nárokov do miery, kedy filtrácia metódou bilaterálneho gridu a PBFIC rezov týchto dát môže prebiehať interaktívne (doba odozvy pod 100 ms), pre rekurzívnu metódu dokonca v reálnom čase.

Výsledky práce sú podporené experimentálnou časťou, v ktorej je okrem porovnania rýchlosti metód, ilustrovaný aj minimálny dopad vykonaných optimalizácií na presnosť výsledku. Tá vo väčšine prípadov, pre rozumne veľké hodnoty vstupných parametrov, významne neklesá pod hranicu PSNR 40 dB.

Ako možný smer ďalšieho vývoja pripadá v úvahu optimalizácia pre špecializované grafické karty c.c.≥ 3.0, kde by bolo možné naplno využiť ich rozšírenú funkcionalitu. Napríklad inštrukcie typu shuffle umožňujú medzivláknovú komunikáciu na úrovni registrov, čo by umožnilo čas a pamäťovú náročnosť výpočtu filtrácie ďalej redukovať. Za zaujímavú môžeme považovať aj myšlienku kombinovania techník aktuálnych state of art metód. Predovšetkým podvzorkovania vstupu a delenie rozsahu intenzít, ktoré využíva napr. bilaterálny grid resp. metóda PBFIC rezov. Výsledkom by mohla byť hybridná metóda, ktorá by v sebe spájala výhody jednotlivých algoritmov.

Literatúra

[1] NVIDIA CORPORATION. *NVIDIA CUDA C Programming Guide*. 2010. Dostupné na: <http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html>.

[2] DERICHE, R.. *Recursively Implementing The Gaussian and Its Derivatives*. [b.m.]: INRIA, Unité de Recherche Sophia-Antipolis, 1993. 1893.

[3] AURICH, V., WEULE, J.. Non-Linear Gaussian Filters Performing Edge Preserving Diffusion. *DAGM-Symposium*. 1995. Str. 538–545.

[4] CHEN, J., PARIS, S., DURAND, F.. Real-time edge-aware image processing with the bilateral grid. *ACM Trans. Graph.* 2007, roč. 26, č. 3. ISSN 0730-0301.

[5] YANG, Q., TAN, K., AHUJA, N.. Real-time edge-aware image processing with the bilateral grid. *Computer Vision and Pattern Recognition*. 2009. Str. 557–564. ISSN 1063-6919.

[6] YANG, Q. Recursive Bilateral Filtering. In *ECCV - European Conference on Computer Vision*. 2012. Str. 399–413.

[7] ADAMS, A., GELFAND, N., DOLSON, J., LEVOY, M.. Gaussian kd-trees for fast high-dimensional filtering. *ACM Trans. Graph*. 2009. Str. 21.

[8] ADAMS, A., BAEK J., DAVIS A, M.. Fast High-Dimensional Filtering Using the Permutohedral Lattice. *Comput. Graph. Forum*. 2010, roč. 29, č. 2. Str. 753–762.

[9] BUADES, A., COLL, B., MOREL, J., M.. The staircasing effect in neighborhood filters and its solution. *Trans. Img. Proc.* 2006, roč. 15, č. 6. Str. 1499–1505. ISSN 1057-7149.

[10] BENNETT P., E., MCMILLAN, L.. Video enhancement using per-pixel virtual exposures. *ACM Trans. Graph.* 2005, roč. 24, č. 3. Str. 845–852.

[11] EISEMANN, E., DURAND, F.. Flash photography enhancement via intrinsic relighting. *ACM Trans. Graph.* 2004, roč. 23, č. 3. Str. 673–678. ISSN 0730-0301.

[12] DURAND, F., DORSEY, J.. Fast bilateral filtering for the display of high-dynamic-range images. *ACM Trans. Graph.* 2002, roč. 21, č. 3. Str. 257–266.