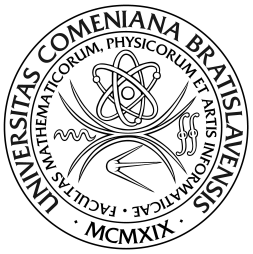
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY
UNIVERZITA KOMENSKÉHO V BRATISLAVE



RNDR. MICHAL ČERVEŇANSKÝ

AUTOREFERÁT DIZERTAČNEJ PRÁCE

NEGRAFICKÉ VÝPOČTY NA GPU V PROSTREDÍ OPENGL

na získanie vedecko-akademickej hodnosti
philosophiae doctor

v odbore doktorandského štúdia
9.2.1. Informatika

BRATISLAVA 2010

Dizertačná práca bola vypracovaná v dennej forme doktorandského štúdia na Katedre aplikovanej informatiky Fakulty matematiky, fyziky a informatiky Univerzity Komenského v Bratislave.

Predkladateľ: RNDr. Michal Červeňanský
FMFI UK
Mlynská dolina
842 48 Bratislava

Školiteľ: Prof. Ing. Miloš Šrámek, PhD.
FMFI UK
Mlynská dolina
842 48 Bratislava

Oponenti: Prof. Ing. Pavel Slavík, CSc.
Katedra počítačové grafiky a interakce, FEL ČVUT
Karlovo náměstí 13
121 35 Praha 2

RNDr. Silvester Czanner, PhD.
WMG
University of Warwick
Coventry
CV47AL
United Kingdom

Mgr. Matej Novotný, PhD.
FMFI UK
Mlynská dolina
842 48 Bratislava

Obhajoba dizertačnej práce sa koná . . . . . o . . . . . h
pred komisiou pre obhajobu dizertačnej práce v odbore doktorandského štúdia vymenovanou predsedom odborovej komisie . . . . . .

9.2.1 Informatika

na Fakulte matematiky, fyziky a informatiky Univerzity Komenského
Mlynská dolina
842 48 Baratislava

Predseda odborovej komisie

. . . . . .

Abstrakt

V tejto práci sa zaoberáme využitím moderných grafických akcelerátorov (GPU) v rôznych oblastiach spracovania obrazových a objemových dát. Dnešné GPU ponúkajú oproti bežným CPU vysoký výkon a poskytujú aj rozsiahlu funkcionalitu a programovateľnosť v OpenGL prostredí. Vďaka tomu sa tieto vlastnosti dajú využiť aj mimo hlavného určenia GPU—počítačovej grafiky.

GPU využívame v oblasti spracovania objemových dát prúdovým prístupom, ktorý efektívne využíva dostupné pamäťové prostriedky počítača. Zameriavame sa hlavne na spracovanie po rezoch. Predstavíme všeobecné techniky, ktoré je možné využiť pri spracovaní objemových dát na GPU. Na ich demonštráciu sme implementovali viaceré lokálne filtre spracovávajúce objemové dáta. V syntetických a reálnych testoch sme dosiahli výrazné urýchlenia oproti bežným výpočtom na CPU.

Ďalšou oblasťou využitia GPU sú paralelné dátovo závislé triangulácie. Výpočet dátovo závislých triangulácií je časovo náročný optimalizačný proces, a tak sme pre potreby využitia GPU navrhli paralelný algoritmus, ktorý vyhovuje obmedzeniam GPU. Výsledné dátovo závislé triangulácie využívame pri rekonštrukcii obrazu, konkrétne pri zväčšovaní. Predstavíme rôzne modifikácie základného algoritmu, ktoré zvyšujú výslednú kvalitu rekonštrukcie s nízkym dopadom na výpočtový čas. Z výsledkov a meraní vyplýva, že využitie dátovo závislých triangulácii je vhodnou voľbou pri rekonštrukcii obrazu.

Abstract

In this thesis we describe the usage of modern graphics accelerators (GPU) in various scientific fields. Today's GPUs offer high performance compared to a standard CPU and provide extensive functionality and programmability in the OpenGL environment. These features can be used also outside of rendering, which is the main domain of GPUs.

We use the modern GPUs in volume data processing based on streaming which efficiently utilizes the available computer memory resources. Here, we focus on the slice-based streamed processing. We present a general technique that can be used with volumetric data on the GPU in various applications. To demonstrate the technique we implemented several streamed local filters. We achieved a significant speed-up compared to the standard CPU computations in tests with both synthetic and real tomographic data.

The next area of GPU usage is data-dependent triangulation which involves a time-consuming optimization process. For a GPU implementation we proposed a parallel algorithm which satisfies the GPU constraints. The resulting triangulation is used in image reconstruction, particularly in image magnification. We introduce various modifications of the basic algorithm which improve the final quality of the reconstruction with low impact on computation time. The results and measurements show that the use of data-dependent triangulation is a good choice for image reconstruction.

Úvod

Moderné grafické karty (GPU – Graphics Processing Unit) sú v dnešnej dobe veľmi rozšírené a vyskytujú sa takmer v každom bežnom počítači. Ich výkon enormne vzrástol a presiahol aj výkon bežných CPU. So vzrastajúcim výkonom sa rozšírila funkcionalita GPU a hlavne programovateľnosť. Táto programovateľnosť nie je úplne bez obmedzení, ako je to pri CPU, ale je dostatočne flexibilná, aby aj v tomto atribúte konkurovala CPU. S rozširujúcou sa funkcionalitou a výkonom sa GPU začali využívať aj na výpočty rôzneho charakteru, nielen na renderovanie trojrozmerných scén, na čo sú primárne určené. Takéto všeobecné výpočty na grafických procesoroch (GPGPU – General-Purpouse Computation on Graphics Procesing Unit) [1] sú veľmi populárne z dôvodu niekoľkonásobného urýchlenia výpočtov oproti CPU, niekedy o dva alebo aj viac rádov. Využívanie výkonu GPU je populárne najmä vo vedeckých aplikáciách kvôli potrebe spracovania enormného množstva dát a možnej paralelizácie problémov. Aplikácie využívajúce výkon GPU sa postupne rozširujú z vedeckých oblastí aj do bežných oblastí, najmä do rôznych softvérov spracovávajúcich obraz a video.

Na využitie poskytovanej funkcionality GPU slúžia rôzne programovacie rozhrania. My využívame knižnicu OpenGL (*Open Graphics Library*) pre jej platformovú nezávislosť a širokú dostupnosť.

V tejto dizertačnej práci sa budeme zaoberať dvomi oblasťami spracovávajúcimi obrazové a objemové dáta. Prvou oblasťou je spracovanie a predspracovanie objemových dát pre rôzne segmentačné a vizualizačné techniky. Druhou oblasťou sú dátovo závislé triangulácie, ktoré využijeme v metódach spracovania obrazu, konkrétne v ich zväčšovaní.

Ciele dizertačnej práce

Hlavným cieľom dizertačnej práce je preveriť schopnosti moderných GPU pri všeobecnom spracovaní objemových dát. Konkrétne ide o návrh nových algoritmov, prípadne optimalizáciu existujúcich, na aktuálnu funkcionalitu GPU a využiť ju v čo najväčšej miere a tak dosiahnuť urýchlenie výpočtov oproti štandardným algoritmom. Pri spracovaní objemových dát je cieľom využitie prúdového prístupu, ktorý efektívne využíva pamäťové zdroje počítača, a tak umožnuje spracovanie rozsiahlych objemových dát aj na bežných počítačoch.

Prúdové spracovanie objemových dát

Objemové dáta obsahujú veľké množstvo údajov a bez podporných algoritmov na ich spracovanie a vizualizáciu je veľmi náročné sa v nich orientovať. Medzi hlavné algoritmy spracovávajúce objemové dáta patria algoritmy na odstraňovanie šumu, ostrenie hrán, detekciu hrán, označovanie regiónov, segmentáciu a mnohé ďalšie. Tieto algoritmy môžeme rozdeliť podľa prístupu k jednotlivým elementom dát na bodové, lokálne a globálne. Pod bodovými rozumieme algoritmy, ktoré spracovávajú iba hodnotu príslušného voxela (napr. prahovanie). Pod lokálnymi rozumieme algoritmy, ktoré potrebujú aj isté okolie príslušného voxela (napr. filtrovanie, detekcia hrán). Veľkosť tohto okolia závisí na danom algoritme. Globálne algoritmy potrebujú na výpočet výslednej hodnoty pre spracovávaný voxel informáciu o všetkých voxeloch (napr. Fourierova transformácia). Pri štandardnom spracovaní objemových dát sa načítajú do pamäte, kde k nim pristupuje algoritmus, vykonáva výpočty a generuje výstupné hodnoty. Nakoľko reálne objemové dáta vždy obsahovali relatívne veľa údajov v porovnaní s aktuálnymi pamäťovými možnosťami bežných počítačov, je toto spracovanie pamäťovo náročné a niekedy pomocou štandardných prístupov na bežne dostupných počítačoch až nemožné.

Odstránením problému nedostatku pamäti sa venuje prúdové spracovanie dát, kde je objem rozdelený na časti a tieto sú spracovávané postupne – prúdovo. Do pamäti sa nahrá prvá časť dát, spracuje sa, uloží (zobrazí) výsledok a pokračuje sa ďalšou časťou pokiaľ sa nespracujú celé dáta. Pri tomto procese je v pamäti uložená len časť dát, ktorá nezaťažuje natoľko pamäť a tak dovoľuje spracovanie aj takých dát, ktoré sa nezmestia celé do operačnej pamäte počítača. Častým prístupom je rozdelenie dát na bloky (kocky/kvádre) s následným spracovaním [2]. Toto delenie je výhodné hlavne pre vizualizáciu dát, kde je potrebný „náhodný“ prístup k dátam. Štandardne sa dáta spracovávajú postupuje po voxeloch od začiatku dát po koniec. Pri takomto spracovaní je výhodnejšie spracovanie po rezoch [3]. Hlavným z dôvodov je potreba duplikovania voxelov na hraniciach blokov pri lokálnych operáciách. Táto duplicita výrazne zvyšuje pamäťové nároky. Špeciálnym blokom je aj rez, kde rozmer v smere Z je rovný jednej. Pri spracovaní po rezoch táto duplicita v smeroch X,Y odpadá a v smere Z je automaticky dodržaná nahraním potrebného počtu rezov do pamäte. Tento proces je výhodnejší pre bodové a lokálne operácie. Prúdové spracovanie po rezoch sa dá aplikovať aj na viaceré algoritmy globálneho charakteru, ale s potrebou ukladania medzivýsledku na disk, čo predstavuje isté spomalenie [3].

Jednotlivé algoritmy potrebujú rôzny počet rezov v závislosti na vykonávanej operácii (pri bodovej je to jeden rez). Na výpise 1 je zobrazené rozhranie, ktoré sme navrhli pre prúdového spracovania objemových dát, kde si metóda (ozn. process) na začiatku spracovania interne ukladá potrebný počet rezov, následne spracuje príslušný rez a vráti ho aplikácii. Takýmto spôsobom sa postupne spracujú všetky rezy. Uvedený prístup je univerzálny a nezávislý na hardvérovej optimalizácii (CPU, SSE, GPU).

Prúdové spracovanie dát s využitím GPU

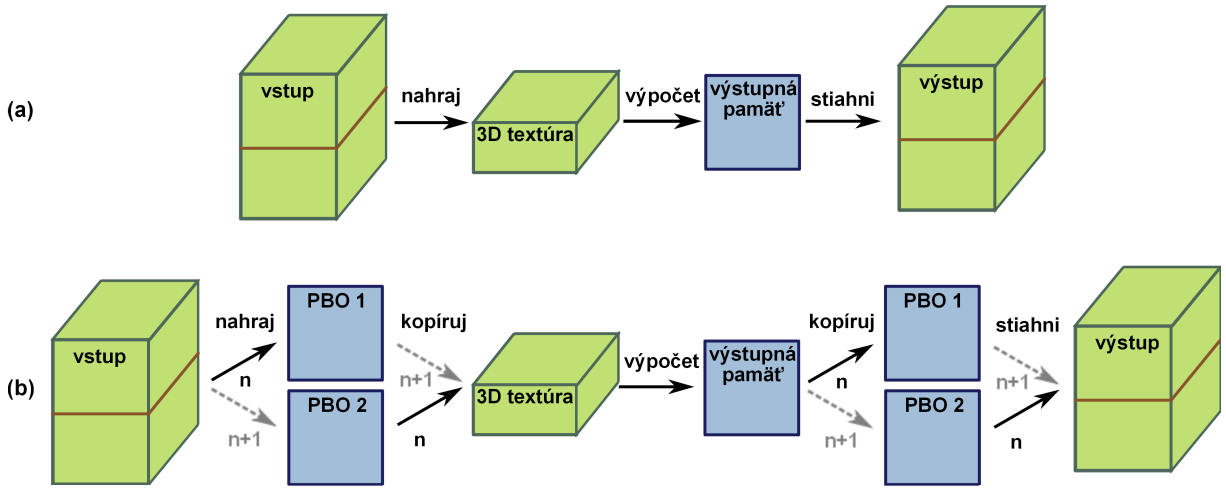
V tejto časti uvedieme základný postup a jeho rozšírenia, ktoré sme využili pri spracovaní objemových dát, ktoré lepšie využívajú funkcionalitu dnešných GPU.

```
1  if(process.open())
2  {
3    while(process.isReadyForUpload() || process.isReadyForDownload())
4    {
5      if(process.isReadyForUpload())
6      {
7        ...
8        process.uploadSlice(sliceIn);
9      }
10     if(process.isReadyForDownload())
11     {
12       process.downloadSlice(sliceOut);
13       ...
14     }
15   }
16   process.close();
17 }
```

Výpis 1: Rozhranie pre prúdové spracovanie objemových dát po rezoch.

Základný postup
Pri štandardnom postupe spracovania objemových dát je na začiatku nahratý potrebný počet rezov z CPU do 3D textúry na GPU. Následne prebieha spracovanie vstupného rezu. Spracovaný rez je uložený na disk prípadne použitý na ďalšie spracovanie. Samotné spracovávanie je vykonávané vo fragmentovom programe, kde vstupom je 3D textúra s potrebným počtom rezov a výstupom spracovaný rez.

Asynchrónny prenos dát
Vyššie uvedený priamočiary postup je možné aplikovať aj na starších GPU. Aktuálne modely GPU však umožňujú využiť asynchrónny prenos dát medzi CPU a GPU pamäťou bez potreby synchronizácie. Pri synchrónnych volaniach dochádza k plytvaniu výpočtovými zdrojmi či už CPU alebo GPU a dochádza k spomaleniu výpočtov. Na obrázku 2(a) je zobrazený štandardný prenos dát bez využitia asynchrónnej technológie. Asynchrónnu technológiu prenosu dát je možné využiť pomocou OpenGL rozšírenia *ARB_pixel_buffer_object* (ozn. *PBO*). Jeden PBO zabezpečuje asynchrónny prenos dát, ale GPU musí čakať na prenos týchto dát čím pracuje neefektívne. Z tohto dôvodu je potrebné použiť viac PBO. Jeden PBO nahráva dáta na GPU priamo do video pamäti bez obmedzenia prebiehajúcich výpočtov. GPU počas tohto nahrávania spracováva dáta uložené v druhom PBO. Pri skončení výpočtu na aktuálnom PBO nasleduje výpočet na ďalšom a pôvodný PBO sa môže použiť na asynchrónne nahrávanie dát. Tento proces pokračuje až do skončenia výpočtu. Obdobne sa postupuje aj pri sťahovaní dát z GPU. Asynchrónny prenos dát a výsledný výpočtový proces je zobrazený na obrázku 2(b). Týmto postupom sa odstráni pôvodná synchrónna funkcionalita prenosu dát medzi CPU a GPU. Takýto proces prenosu dát je vhodný práve pre prúdové spracovanie dát, prípadne na veľké prenosy dát.



Obrázok 2: Zobrazenie rôznych verzií prenosu dát. Pôvodná technológia pomocou synchrónnych volaní (a). Prúdová verzia asynchrónneho prenosu dát (b).

Spracovanie viacerých rezov súčasne

V tejto časti si predstavíme techniku, ktorú využijeme na spracovanie viacerých rezov súčasne v jednom fragmentovom programe. Je založená na OpenGL rozšírení *ARB_draw_buffers*, ktoré umožňuje zapisovať do viacerých renderovacích oblastí súčasne (textúr). Táto technológia sa označuje ako *Multiple Render Targets – MRT*. Pomocou tohto rozšírenia je možné spracovať viacero rezov súčasne. Pri spracovaní viacerých rezov súčasne sa odľahčí rasterizačná jednotka, ktorá tak nemusí počítať textúrové súradnice a dodatočné interpolované hodnoty pre všetky spracovávané rezy, ale iba pre jeden. Taktiež sa zníži počet OpenGL volaní. Pri použití tejto techniky je potrebné vykonať minimálne zmeny v OpenGL volaniach a zväčšiť rozmer 3D textúry v smere Z o počet spracovávaných rezov súčasne znížený o jeden. Taktiež je potrebné adekvátne zvýšiť počet 2D textúr reprezentujúcich spracovávané rezy. Pri výpočtoch vo fragmentových programoch, ktoré využívajú čítanie dát iba v rámci jedného rezu (napr. bodové operácie) je tento výpočet opakovaný pre každý spracovávaný rez. Výraznejšia zmena nastane vo fragmentovom programe využívajúcom dáta aj zo susedných rezov (napr. výpočet konvolúcie v smere Z). Pri štandardnom postupe by sa pre každý spracovávaný rez načítali hodnoty zo susedných rezov, pričom by vznikali duplicitné čítania dát v závislosti od počtu potrebných rezov. Nakoľko čítanie z textúry je časovo náročná operácia, je vhodné toto duplicitné čítanie dát odstrániť. Jednou z možností je predčítať potrebné dáta, pokiaľ to výpočet umožňuje (fixný počet dát, dostatok registrov na GPU), prípadne využiť iné optimalizácie. Tieto optimalizácie je potrebné prispôsobiť konkrétnym algoritmom.

Balenie voxelov
Táto technika je založená na využití viackanálových – farebných – textúr. Najčastejšie využívané sú štvorkanálové textúry reprezentujúce farebné hodnoty RGB (Red, Green, Blue) a priehľadnosť A (Alpha). Ako už bolo spomenuté, čítanie z textúr je časovo náročná operácia a preto je vhodné počet čítaní z textúr minimalizovať. Jednou možnosťou je zoskupovať vstupné dáta po štvoriciach. Týmto sa docieli získanie štyroch hodnôt na jedno čítanie z textúry, ktoré reprezentujú štyri voxely zo vstupných dát. Balenie je možné vykonať v smere X alebo Y na vstupnom reze. Tento proces si vyžaduje zarovnanie posledných voxelov v riadku nulami pokiaľ veľkosť spracovávaného objemu nie je v smere X násobkom 4. Objem prenosu dát medzi CPU a GPU je takmer nezmenený až na prípadné zarovnanie. Balenie dát si nevyžaduje špeciálnu úpravu OpenGL volaní. Výraznejšie zmeny nastanú vo fragmentových programoch, hlavne pre výpočty v smere X, kde je potrebné brať ohľad na balenie voxelov.

V tejto časti sme uviedli viaceré všeobecné techniky, ktoré rozširujú základný prístup prúdového spracovania objemových dát. Tieto techniky je potrebné prispôsobiť danému algoritmu a jeho špecifikám. Uvedené techniky je možné navzájom kombinovať za účelom dosiahnutia urýchlenia spracovávania dát.

Filtre

Počítačová grafika a hlavne jej oblasť spracovanie obrazu využíva rôzne operácie za účelom vylepšenia obrazu (odstránenie šumu, ostrenie hrán a iné). V oblasti spracovania a vizualizácie objemových dát sa tieto procesy tiež využívajú, pričom sú aplikované na jednotlivé rezy samostatne alebo priamo na celé dáta vo všetkých troch smeroch. Týmto operáciám spracovávajúcim obraz (objem) za účelom získania nového obrazu (objemu) hovoríme filtre a výpočet je označovaný ako filtrácia. Medzi hlavné filtre, ktoré sú využívané v spracovaní obrazu patrí výpočet konvolúcie. Konvolúcia môže byť využitá na rôzne účely, pričom pomocou nej vieme v spracovávanom obraze odstrániť šum, vyhladiť ho, prípadne ostriť hrany a iné. Tento proces je algoritmicky jednoduchý, ale v dôsledku veľkého objemu dát je časovo náročný, a preto je vhodné využiť výpočtový výkon GPU na jeho urýchlenie. Ďalším výpočtom, ktorým sme sa zaoberali je filter detekujúci tubulárne štruktúry (cievy), ktorý je používaný v medicínskych aplikáciách.

Vyššie uvedené všeobecné techniky sme využili pri implementácii separovateľnej a neseparovateľnej konvolúcie na GPU. Taktiež sme implementovali filter detekujúci tubulárne štruktúry. Konkrétne implementačné detaily sú uvedené v dizertačnej práci.

Merania

Vyššie prezentované techniky a ich implementácie využívajúce výkon GPU sme testovali navzájom. Všetky implementácie používali asynchrónny prenos dát. Na výhodnosť danej techniky ma veľký vplyv rozmer testovaných dát, prípadne veľkosť konvolučného jadra, pričom niektoré techniky urýchľovali výpočet výraznejšia a iné menej. Na základe meraní sme za najvýhodnejšiu

implementáciu pre výpočet separovateľnej a neseparovateľnej konvolúcie vybrali techniku spracovávajúcu štyri rezy súčasne spolu s technikou balenia voxelov. Pre filter detukujúci tubulárne štruktúry bola najvýhodnejšia základná varianta, čiže spracovanie jedného rezu bez balenia voxelov. Tieto varianty sme naimplementovali do balíka *f3d* [4], konkrétne do knižnice *f3dfilter*.

Všetky prezentované testy boli vykonané na počítači s nasledujúcimi parametrami: Intel QuadCore 2.4GHz, 8GB RAM, 500GB HDD, NVIDIA 9800 GTX s 512MB VRAM, Ubuntu 8.04. Výsledky meraní rýchlosti sú uvedené v sekundách. Všetky testy boli spúšťané trikrát a vo výsledkoch sú prezentované minimálne hodnoty. Pri konvolučných jadrách s menším rozmerom je bežné, že nameraná hodnota je vo veľkej miere ovplyvnená pevným diskom. Pomalý pevný disk spôsoboval, že rozdiely medzi rôznymi hardvérovými variantmi a veľkosťami jadier boli minimálne. Pre testy boli preto vstupné dáta generované pomocou špeciálneho nástroja bez diskových operácií a spracované dáta neboli ukladané na disk, ale do pseudosúboru (/dev/null), aby sa zamedzilo jeho vplyvu na merania.

Knižnica *f3dfilter* obsahuje aj iné implementácie uvedených filtrov, a to CPU verziu a optimalizovanú verziu využívajúcu inštrukčnú sadu SSE hlavného procesora. Preto sme sa pri testovaní zamerali aj na merania medzi týmito riešeniami oproti našej OpenGL implementácii (ozn. *GL*).

V tabuľke 1 sú uvedené výsledky pre neseparovateľnú a separovateľnú konvolúciu. Z výsledkov pre neseparovateľnú konvolúciu vyplýva, že OpenGL verzia je 8.1-krát rýchlejšia oproti SSE verzii a 275.1-krát rýchlejšia oproti CPU verzii v priemernom prípade. Vysoký výkon GPU je vidieť na výsledkoch, kde iba v dvoch prípadoch bola verzia SSE rýchlejšia. Pre separovateľnú verziu je SSE verzia rýchlejšia vo viacerých prípadoch, čo je spôsobené časom potrebným na inicializáciu GPU a na prenos dát. V priemernom prípade je však OpenGL verzia 3.1-krát rýchlejšia oproti SSE verzii a 20.7-krát rýchlejšia oproti CPU verzii.

V tabuľke 2 sú uvedené výsledky pre filter detekujúci tubulárne štruktúry. Z nameraných hodnôt vyplýva, že OpenGL verzia ja najrýchlejšia. Tento stav je spôsobený hlavne implementáciou v jednoduchej presnosti, pričom ostatné verzie využívajú dvojitú presnosť výpočtu s plávajúcou desatinnou čiarkou. Preto je OpenGL verzia výpočtu v takom veľkom nepomere k ostatným verziám.

Nakoľko GPU ťaží nielen zo svojho hrubého výkonu, ale aj z rozsiahlej paralelizácie, je potrebné spomenúť paralelizáciu aj pri CPU a SSE verziách podporovaných novými viacjadrovými procesormi. Túto paralelizáciu sme využili v aplikácii spracovávajúcej objemové dáta na vizualizáciu tubulárnych štruktúr. Táto aplikácia (s diskovými operáciami) filtruje vstupné objemové dáta s desiatimi rôzne veľkými Gaussovými jadrami. Na každom predfiltrovanom objeme prebieha dodatočná detekcia tubulárnych štruktúr pomocou výpočtu matíc druhých derivácií a výpočtu vlastných čísel, na základe ktorých je vypočítaná hodnota zvýraznenia cievy [5]. Následne sú vytvorené výstupné dáta reprezentujúce maximá z jednotlivých výpočtov (procesov), ktoré sú uložené na disk alebo vizualizované (obrázok 3). Pri testovaní sa menilo zastúpenie SSE a OpenGL verzií algoritmov na výpočte separovateľnej konvolúcie tak, že postupne boli SSE verzia nahrádzané OpenGL verziami od najväčších jadier k najmenším. Na detekciu tubulárnych štruktúr boli použité CPU, SSE a OpenGL verzie, pričom všetky spustené procesy

	Separovateľná konvolúcia					Neseparovateľná konvolúcia				
GL	**3**	**5**	**9**	**15**	**31**	**3^3**	**5^3**	**9^3**	**15^3**	**31^3**
128^3	0,170	0,160	0,170	0,180	0,190	0,150	0,180	**0,260**	**0,490**	**2,440**
256^3	0,290	0,300	0,340	0,310	**0,320**	0,360	**0,470**	**1,070**	**2,780**	**17,750**
512^3	1,210	1,230	1,230	1,440	**1,600**	**1,790**	**2,740**	**7,550**	**21,130**	**142,640**

SSE	**3**	**5**	**9**	**15**	**31**	**3^3**	**5^3**	**9^3**	**15^3**	**31^3**
128^3	**0,030**	**0,040**	**0,060**	**0,070**	**0,150**	**0,060**	**0,150**	0,490	2,500	16,740
256^3	**0,140**	**0,170**	0,310	0,650	1,100	**0,330**	0,770	4,840	20,670	147,040
512^3	1,010	1,670	2,820	5,170	8,960	2,320	7,050	39,420	166,850	1225,130

CPU	**3**	**5**	**9**	**15**	**31**	**3^3**	**5^3**	**9^3**	**15^3**	**31^3**
128^3	0,090	0,090	0,220	0,480	0,910	0,340	0,970	4,000	21,270	302,440
256^3	0,420	0,690	1,500	3,350	6,280	2,570	7,650	39,800	169,180	2153,580
512^3	3,590	5,480	13,440	26,930	79,650	20,800	68,510	319,570	1287,290	51115,141

Tabuľka 1: Časy výpočtov hardvérových verzií separovateľnej a neseparovateľnej konvolúcie. V stĺpcoch sú uvedené veľkosti konvolučných jadier, v riadkoch veľkosti spracovávaných objemov a použité hardvérové verzie. Červenou farbou sú označené najrýchlejšie výpočty medzi rôznymi hardvérovými verziami.

	Tubulárne štruktúry			
	128^3	**256^3**	**512^3**	**1024^3**
GL	**0,220**	**0,870**	**5,800**	**44,550**
SSE	0,740	5,800	47,820	384,080
CPU	1,300	9,820	96,430	773,200
GSL	3,610	27,930	226,980	1832,360

Tabuľka 2: Časy výpočtov hardvérových verzií filtra detekujúceho tubulárne štruktúry. V stĺpcoch sú uvedené veľkosti spracovávaných objemov, v riadkoch rôzne hardvérové verzie. Červenou farbou sú označené najrýchlejšie výpočty.

využívali iba jednu z týchto verzií. Na testovanie sme použili CT snímku dolných končatín s rozmermi $512 \times 512 \times 1950$ a 12 bitovej fixnej presnosti, prekonvertovanú do 32 bitovej presnosti s pohyblivou desatinnou čiarkou, čo predstavovalo 1.9GB dát. Pri spracovaní desiatimi filtrami išlo teda o súčasné spracovanie 19GB dát. V tabuľke 3 sú uvedené výsledky z meraní, kde $RGT_{CPU}MW$ predstavuje originálnu verziu testu s využitím CPU algoritmu na detekciu tubulárnych štruktúr, $RGT_{SSE}MW$ so SSE verziou a $RGT_{GL}MW$ s OpenGL verziou výpočtu.

Ako najvhodnejšia kombinácia sa ukázalo využitie jedného SSE a deväť OpenGL procesov pre výpočet separovateľnej konvolúcie a SSE verzia detekujúca tubulárne štruktúry, čo predstavovalo 1.49 násobné urýchlenie oproti verzii využívajúcej iba SSE algoritmy.

	$RGT_{CPU}MW$	**$RGT_{SSE}MW$**	**$RGT_{GL}MW$**
10S0G	402,14	421,92	374,38
9S1G	356,86	373,71	325,69
8S2G	333,21	341,13	316,42
7S3G	315,16	317,72	**298,02**
6S4G	317,43	305,56	298,21
5S5G	309,59	295,90	306,29
4S6G	**303,53**	287,09	308,15
3S7G	310,85	283,54	330,14
2S8G	307,61	284,34	337,41
1S9G	316,43	**282,25**	354,40
0S10G	316,22	283,42	347,41

Tabuľka 3: Zobrazenie kombinácií SSE a OpenGL verzií programov na výpočet separovateľnej konvolúcie v reálnej aplikácii. Stĺpec RGTMW označuje výpočet s využitím rôznych hardvérových verzií na detekciu tubulárnych štruktúr. Jednotlivé riadky predstavujú kombináciu (pomer) medzi SSE a OpenGL verziou výpočtu konvolúcie programu, kde prvá číslica predstavuje počet spustených SSE programov a druhá číslica počet OpenGL programov na štvorjadrovom procesore. Červenou farbou sú zvýraznené najrýchlejšie výpočty

Obrázok 3: Maximálna intenzitová projekcia s farebne zvýraznenými cievami na základe ich priemeru. Modrá - zelená - červená farba korešponduje s tenkými - strednými - hrubými štruktúrami.

Paralelné dátovo závislé triangulácie a rekonštrukcia obrazu

Optimálne triangulácie sú často využívané vo vedeckých a technických aplikáciách rôznych odvetví. V tejto časti sa zameriavame na získavanie lokálne optimálnych triangulácií iteračným spôsobom (optimalizáciou). Tie môžu byť použité v rôznych geometricky definovaných problémoch ako napríklad simulácii konečných prvkov alebo rekonštrukcii obrazu. Špeciálne sa zameriavame na oblasť rekonštrukcie obrazu, najmä na zväčšovanie s dôrazom na zachovanie hrán. Takéto zväčšovanie je možné riešiť pomocou špeciálneho prípadu optimálnych sietí a to dátovo závislými trianguláciami (*data-dependet triangulations - DDT*) [6]. Hlavnou výhodou tejto techniky je schopnosť prispôsobiť výslednú triangulovanú sieť vstupným dátam.

Rekonštrukcia obrazu je jedna z viacerých aplikačných oblastí, ktoré využívajú dátovo závislé triangulácie. Dôvodom výberu tejto aplikačnej oblasti bola možnosť vizuálneho zobrazenia výsledkov. Na vytváranie dátovo závislých sietí je použitý Lawsonov optimalizačný proces. Efektívna implementácia tejto techniky na GPU si vyžaduje jej paralelizáciu. Možnosti GPU nie sú priamo prispôsobené takémuto typu výpočtov a dátovým štruktúram. Hlavnou požiadavkou pri paralelnom návrhu bolo obísť tieto hardvérové obmedzenia a navrhnúť taký algoritmus, ktorý bude implementovateľný na GPU.

Dátovo závislé triangulácie

Najbežnejšie používanými technikami v spracovaní obrazu sú konvolučné metódy. Úplne odlišný prístup založený na geometrickej rekonštrukcii zaviedol Dyn [6] s využitím dátovo závislých triangulácií. Tieto triangulácie sú prispôsobené vstupným hodnotám, čo vedie k po častiach lineárnej interpolácii. Na rozdiel od iných metód generujúcich triangulácie, dátovo závislé triangulácie prispôsobujú hrany vstupným dátam a organizujú trojuholníkové štruktúry do siete, ktorá zachováva črty zo vstupných dát. Kvalita výslednej triangulácie je definovaná pomocou špeciálnej *cenovej funkcie* v optimalizačnom procese.

V práci sa zameriavame výlučne na hranovo založené dátovo závislé triangulácie, v ktorých sú oceňované iba hrany triangulácie. Každá vnútorná hrana je závislá na štyroch vrcholoch, ktoré tvoria štvoruholník (vytvorený z dvoch susedných trojuholníkov), kde daná hrana reprezentuje jeho diagonálu. Existuje niekoľko ďalších prístupov, ktoré priraďujú cenu iným komponentom triangulácií (vrcholom, hranám, trojuholníkom), v závislosti na rôznych geometrických a negeometrických kritériách [7, 8, 6]. Avšak my budeme používať Sederbergovu cenovú funkciu, ktorá je závislá na štyroch vrcholoch [9].

Väčšina existujúcich optimalizačných prístupov je založená na bežnej topologickej operácii nazývanej *preklopenie hrany*. Každá vnútorná hrana je zjednotením dvoch susedných trojuholníkov, ktoré formujú štvoruholník. Ak je tento štvoruholník konvexný a nedegenerovaný, tak môže byť daná hrana nahradená druhou diagonálou štvoruholníka – preklopením.

Najpopulárnejšou technikou na získavanie dátovo závislých triangulácií je Lawsonova optimalizácia [6] (tiež nazývaná lokálna optimalizácia), ktorá využíva operáciu preklápania hrán. Táto technika spracováva hrany iteračne a vyhodnocuje ich lokálnu optimalitu.

Paralelné dátovo závislé triangulácie

V tejto časti prezentujeme paralelnú verziu dátovo závislých triangulácií založenú na Lawsonovom optimalizačnom procese. Pri návrhu paralelného algoritmu sme zobrali do úvahy možnosti a obmedzenia GPU.

Na začiatku potrebujeme definovať potrebnú terminológiu. *Susedmi* hrany *e* v triangulácii $T(\mathbf{V})$ označujeme množinu hrán z $T(\mathbf{V})$, ktoré tvoria spolu s hranou *e* trojuholník. Množina susedov hrany *e* je označená ako región stupňa 1 (1-*región*). Na základe tohto označenia definujeme n-*región* ($n > 1$) hrany *e* z $T(\mathbf{V})$ ako prienik nasledujúcich hrán:

- hrany z $(n-1)$-regiónu hrany *e* a
- susedov hrán z $(n-1)$-regiónu hrany *e*.

Lokálna optimalita hrany e v dátovo závislej triangulácii závisí od výberu cenovej funkcie. Preto definujeme región vplyvu (*region of influence - ROI*) hrany e ako množinu tých hrán, ktorých zmena (preklopenie) ovplyvní lokálnu optimalitu hrany e. Pre Sederbergovu cenovú funkciu je región vplyvu stupňa 2 a obsahuje 12 hrán.

V paralelnej verzii algoritmu by mali byť hrany spracované súčasne z čoho plynú isté obmedzenia. Predstavujeme iteračný algoritmus, ktorý pozostáva v každej iterácií z nasledujúcich troch krokov: *vytváranie kandidátov, akceptovanie a zamietanie kandidátov, preklápanie hrán*.

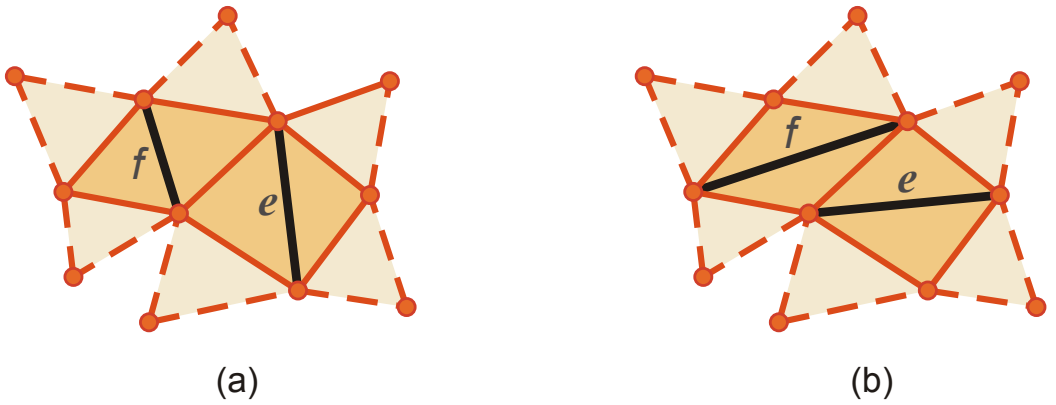
Vytváranie kandidátov
V prvom kroku jednotlivých iterácií je vyčíslená cenová funkcia pre každú hranu triangulácie a hrany sú rozdelené do dvoch množín. Prvá množina obsahuje lokálne neoptimálne *kandidátske* hrany, ktorých preklopenie má za následok zníženie ceny celej triangulácie. Ostatné hrany ostávajú nepreklopené v aktuálnej iterácii algoritmu. Vyčísľovanie cenovej funkcie sa navzájom neovplyvňuje a preto môže byť vykonávané paralelne na všetkých hranách.

Akceptovanie a zamietanie kandidátov
Predstavme si situáciu s dvoma kandidátskymi hranami e a f, kde f je v ROI hrany e a naopak. Ak preklopíme jednu z týchto hrán, lokálna optimalita druhej hrany môže byť taktiež zmenená, nastane konflikt (obrázok 4). Na vyvarovanie sa takýchto konfliktov je potrebné vybrať z množiny kandidátov takú podmnožinu nezávislých hrán, ktorých ROI sa neprekrývajú, takže môžu byť preklopené naraz.

Inak povedané, množina kandidátov je rozdelená na dve disjunktné množiny

- *zamietnuté* hrany, ktoré nebudú preklopené a
- *akceptované* hrany, ktoré môžu byť preklopené paralelne.

Klasifikácia hrán je založená na porovnaní identifikačných čísel hrany (ID). Každá hrana má svoje jedinečné ID, ktoré je jej pridelené na začiatku algoritmu. Ak je hrana preklopená, uchováva si toto ID aj po preklopení. Tento proces je iteratívny a pokračuje pokiaľ existujú hrany v množine kandidátov.



Obrázok 4: Hrany e a f v konfliktnej situácií pred a po paralelnom preklopení (hrana e by mala byť neovplyvnená hranou f a naopak).

Preklápanie hrán
V poslednom kroku algoritmu sú paralelne preklopené všetky hrany z množiny akceptovaných hrán vytvorenej v predchádzajúcom kroku algoritmu. Taktiež sú aktualizované príslušné dátové štruktúry spracovávanej hrany a hrán z jej ROI.

Celý iteračný proces je vykonávaný pokiaľ nie je dosiahnutá lokálne optimálna triangulácia. Pseudo kód algoritmu je uvedený na výpise 5.

Tento paralelný algoritmus pre dátovo závislé triangulácie sme navrhli na vykonávanie vo fragmentovej jednotke tak ako väčšina GPGPU výpočtov. Vďaka tomu je možné vykonávať tento algoritmus aj na starších GPU. Konkrétnym implementačným detailom sa venuje naša práca [10].

Vylepšenia

Prezentované riešenie výpočtu dátovo závislej triangulácie založené na paralelizovanom Lawsonovom optimalizačnom procese je všeobecné a umožňuje vypočítať lokálne optimálnu triangulaciu s použitím rôznych cenových funkcií na GPU.

Prvotné testy tohto prístupu ukázali isté obmedzenia v porovnaní s neparalelnou CPU verziou algoritmu. Po prvé, sieť optimalizovaná pomocou GPU mala zvyčajne vyššiu cenu než sieť optimalizovaná pomocou CPU algoritmom. Po druhé, v rekonštruovaných obrázkoch sa objavili rušivé artefakty, ktoré sa často nachádzali v hranových oblastiach.

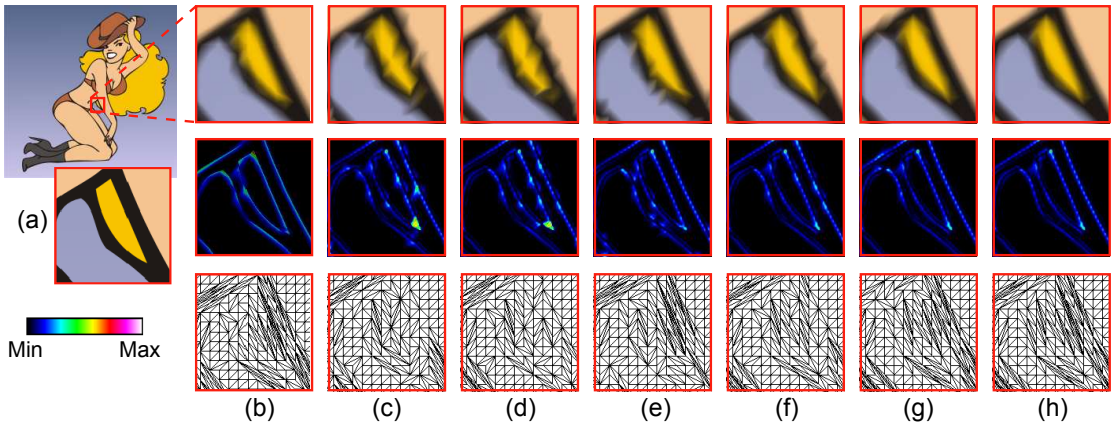
Na potlačenie týchto artefaktov sme navrhli metódy pracujúce v obrazovom priestore. Taktiež sme navrhli modifikácie algoritmu za účelom potlačenia artefaktov a zníženia ceny triangulácie pracujúce vo výpočtovom priestore.

```
1   repeat
2     AcceptedSet.clear()
3     CandidateSet.clear()
    // – Krok 1 –
4     for ( každú hranu e z triangulácie )
5       if ( e nie je lokálne optimálna )
6         CandidateSet.add(e)
7       endif
8     endfor
    // – Krok 2 –
9     while ( existuje hrana e z CandidateSet )
10      if ( existuje akceptovaná hrana f v ROI hrany e )
11        CandidateSet.remove(e)     // zamietni e
12      else
13        if ( e.ID < min ( ID všetkých kandidátskych hrán z ROI hrany e ) )
14          AcceptedSet.add(e)     // akceptuj e
15          CandidateSet.remove(e)
16        else
17              // e ostáva kandidátom, nezmenená
18        endif
19      endif
20    endwhile
    // – Krok 3 –
21    for ( každú hranu e )
22      if ( e z AcceptedSet )
23        flip(e)
24        zaktualizuj dátové štruktúry(e)
25      else
26        if ( existuje akceptovaná hrana v 1-regióne hrany e ) // susedná hrana
27          zaktualizuj dátové štruktúry(e)
28        endif
29      endif
30    endfor
31  until lokálne optimálna triangulácia nie je vypočítaná
```

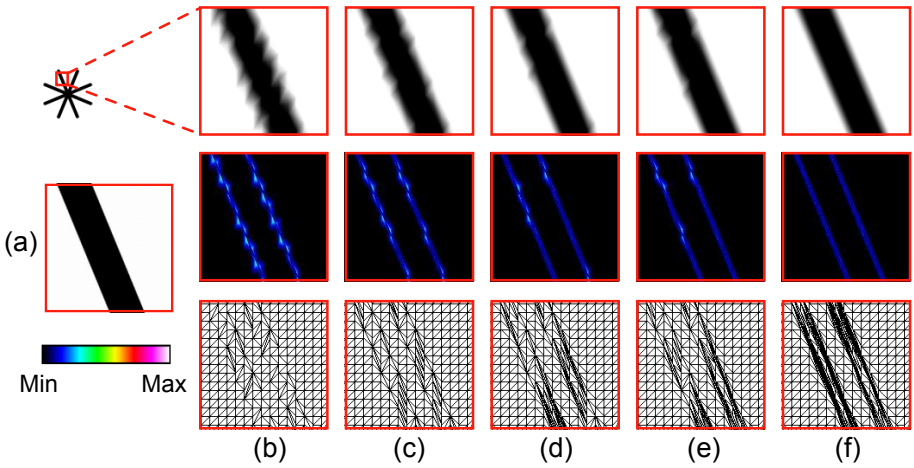
Výpis 5: Pseudokód paralelnej verzie Lawsonovho optimalizačného algoritmu, vyhovujúceho možnostiam GPU.

Vylepšenia vo výpočtovom priestore

Modifikácie algoritmu uvedené nižšie sme navrhli za účelom potlačenia viditeľných artefaktov a zníženia ceny triangulácie. Vplyv modifikácií na výsledky sú prezentované na obrázkoch 6 a 7. Stredný riadok v týchto obrázkoch zobrazuje diferenčný obrázok (v porovnaní s originálnym obrázkom) vypočítaný v perceptuálne lineárnom farebnom priestore L*,u*,v*.



Obrázok 6: Záber na 1200% zväčšenie rasterizovaného vektorového obrázka. Zvýraznená oblasť zachytáva časť s najväčším výskytom artefaktov. Originálny obrázok (a), CPU DDT algoritmus (b), Basic (c), MaxGain (d), ExpRoi (e), ExpRoiMaxGain (f), MeshSobMax (g), ExpRoiMaxGainMeshSobMax (h) modifikácie.

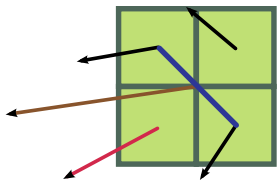


Obrázok 7: Rekonštruovaný obrázok pri 1000% zväčšení s využitím rôznych modifikácií: originálny obrázok (a), Basic (b), MeshCanny (c), MeshSobAvg (d), MeshSobMax (e), ExpRoiMaxGainMeshSobMax (h).

Expanzia ROI
Táto modifikácia rozširuje oblasť ROI pri procese výberu kandidátov. V základnom prístupe je tento región veľkosti 2, čím pokrýva dvanásť hrán. Tento región sme rozšírili o dodatočné hrany na región stupňa 3. Táto modifikácia (označenie *ExpRoi*) čiastočne potlačila artefakty a výsledná cena je nižšia. Obrázok 6(e) zobrazuje vizuálny prínos v porovnaní so základným algoritmom označeným ako *Basic*.

Maximalizácia prírastku
Hlavným cieľom tejto modifikácie (označenie *MaxGain*) je zníženie ceny triangulácie. Na rozdiel od základného algoritmu, kde sú akceptované hrany vyberané výhradne na základe ID hrany, v tejto modifikácií je hodnotená veľkosť príspevku preklopenia hrany k minimalizácii ceny triangulácie. Počas iterácie nad množinou kandidátov je vybraná hrana s najväčším prírastkom (rozdiel medzi cenou pred preklopením a cenou po preklopení hrany) z ROI aktuálne spracovávanej hrany. Obrázok 6(d) ukazuje výsledky v porovnaní so základným GPU algoritmom.

Predspracovanie siete
Táto modifikácia je založená na úprave iniciálnej triangulácii. Jej cieľom je prispôsobenie vstupnému obrázku. Jedná sa hlavne o diagonálne hrany iniciálnej triangulácie, ktoré sú upravené podľa smeru vysokofrekvenčných oblastí vo vstupnom obrázku. Na zisťovanie významných vysokofrekvenčných oblastí sme použili rôzne hranové detektory. V prvom prípade bol použitý Cannyho hranový detektor (ozn. *MeshCanny*, obrázok 7(c)). Ďalší prístup odhaduje gradienty pomocou Sobelovho operátor, kde bola diagonálna hrana orientovaná podľa maximálneho gradientu (ozn. *MeshSobMax*, obrázok 7(e)) zo štvorice pixelov zodpovedajúcej diagonálnej hrane, alebo podľa priemerného gradientu (ozn. *MeshSobAvg*, obrázok 7(d)). Situácia je znázornená na obrázku 8.



Obrázok 8: Zobrazenie umiestnenia hranovej diagonály (modrá) podľa maximálneho gradientu (červená) alebo priemerného gradientu (hnedá) na základe gradientov v jednotlivých pixeloch (čierna) vyrátaných pomocou Sobelovho operátora.

Vyššie uvedené modifkácie je možné navzájom kombinovať s cieľom dosiahnuť kvalitnejšej rekonštrukcie. Jednou z kombinácií je využitie maximalizácie prírastku a rozšíreného ROI, označenej ako *ExpRoiMaxGain*. Ďalšou je kombinácia troch modifikácií a to maximalizácie prírastku, rozšírená oblasť a predspracovanie siete označenej ako *ExpRoiMaxGainMesh* s príslušnou príponou *Canny, SobMax, SobAvg*. V zhode s očakávaniami tieto posledné algoritmy generujú trianguláciu s najnižšou cenou a s minimálnym počtom vizuálnych artefaktov. Výsledky rekonštrukcií z niektorých kombinácií sú zobrazené na obrázkoch 6 a 7.

Vylepšenia v obrazovom priestore

Na zvyšovanie kvality rekonštrukcie sme použili aj techniky pracujúce v obrazovom priestore, pričom neovplyvňujú prebiehajúci výpočet. Pri týchto technikách dochádza ku kombinácií viacerých obrázkov. Tieto obrázky sú výsledkom DDT rekonštrukcie na rôzne transformovaných vstupných obrázkoch. Viacej o týchto technikách je uvedené v dizertačnej práci.

Kombinačná technika

Ďalšiu techniku, ktorú sme navrhli za účelom zvyšovania kvality rekonštrukcie je kombinačná technika, ktorá využíva rekonštrukcie založené na DDT spolu s konvolučnými technikami. Výhodou konvolučných techník je relatívne rýchle spracovanie oproti DDT technikám s primeranou kvalitou. Avšak kvalita DDT techník vo vysokofrekvenčných oblastiach je vyššia až na špecifické prípady. Z tohto dôvodu je výhodné kombinovať obe techniky, kde v homogénnych oblastiach bude použitá konvolučná technika a v nehomogénnych oblastiach DDT technika. Celý proces sa skladá zo šiestich fáz, pričom každá fáza generuje čiastkové výsledky – obrázky, ktoré sú použité v ďalších fázach. Viacej o tejto technike je možné nájsť v dizertačnej práci.

Výsledky

V tejto časti predstavíme výsledky z meraní výpočtového času a kvality rekonštrukcie pre modifikácie uvedené v predošlej časti. Na testovanie sme použili dve skupiny obrázkov. Prvá skupina obsahovala dvanásť obrázkov z reálneho života, ktoré boli vybrané z bežne používaných sád testovacích obrázkov pri spracovaní obrazu. Druhá skupina obsahovala osem umelo vytvorených obrázkov rasterizovaných vo vektorovom editore. Obe skupiny boli zmenšované na štvrtinovú, respektíve osminovú veľkosť z pôvodnej veľkosti iteratívnym zmenšovaním na polovicu veľkosti pomocou bilineárneho filtrovania. Zmenšené obrázky mali rozmery od 64×64 do 250×243 pixelov. Tieto zmenšené obrázky boli zväčšené na pôvodnú veľkosť rôznymi rekonštrukčnými technikami. CPU verzia algoritmu bola implementovaná v jazyku C++. GPU verzie algoritmov využívali OpenGL rozhranie a výpočtové jednotky boli programované v jazykoch glsl a Cg. Testy a merania sme vykonali na systéme vybavenom procesorom Intel Pentium 4 3.0GHz, 1GB RAM s nasledujúcimi GPU: NVIDIA GeForce 8800 GTS so 640MB pamäte, NVIDIA GeForce 9600GT s 512MB pamäte a ATI Radeon HD3850 s 512MB pamäte.

V tejto časti sú uvedené základne merania, podrobnejšie výsledky je možné nájsť v dizertačnej práci.

Výpočtový čas

Predstavené modifikácie boli testované na viacerých GPU, čo bolo možné platformovou nezávislosťou implementácie využívajúcej glsl jazyk. Získané údaje sú uvedené v tabuľke 4. Uvedené výsledky sú v sekundách a sú priemerom z troch meraní. Z predstavených výsledkov vidieť že GPU verzia je 6—8 krát rýchlejšia ako CPU verzia. GeForce 9600 je rýchlejšia v priemere o 35 percent než GeForce 8800 vo všetkých prípadoch. Avšak situácia je odlišná pre GPU

Výpočtový čas (s)	GeForce 8800GTS	GeForce 9600GT	Radeon HD3850
CPU	7,262	7,262	7,262
Basic	0,876	0,593	0,914
ExpRoi	1,176	0,893	1,760
MaxGain	0,865	0,582	0,996
ExpRoiMaxGain	1,230	0,864	4,024
MeshCanny	0,873	0,590	0,908
MeshSobMax	0,884	0,591	0,909
MeshSobAvg	0,878	0,595	0,909
MaxGainMeshSobMax	0,870	0,579	0,979
ExpRoiMeshSobMax	1,185	0,887	1,743
ExpRoiMaxGainMeshSobMax	1,181	0,858	3,975

Tabuľka 4: Priemerné časy výpočtov meraných na rôznych GPU (glsl implementácia).

firmy ATI. Rozdiely závisia od konkrétnej modifikácie, kde je spomalenie 0.1 až 3.5 násobné v porovnaní s GeForce 8800 GPU. Výrazné spomalenie sa týka najmä modifikácii *ExpRoiMaxGain*, ktoré využívajú fragmentové programy s vyšším počtom inštrukcií. Tieto rozdiely sa dajú vysvetliť rozdielnou architektúrou GPU od firiem NVIDIA a ATI, čo vedie k rozdielnej efektívnosti rôznych operácií.

Kvalita

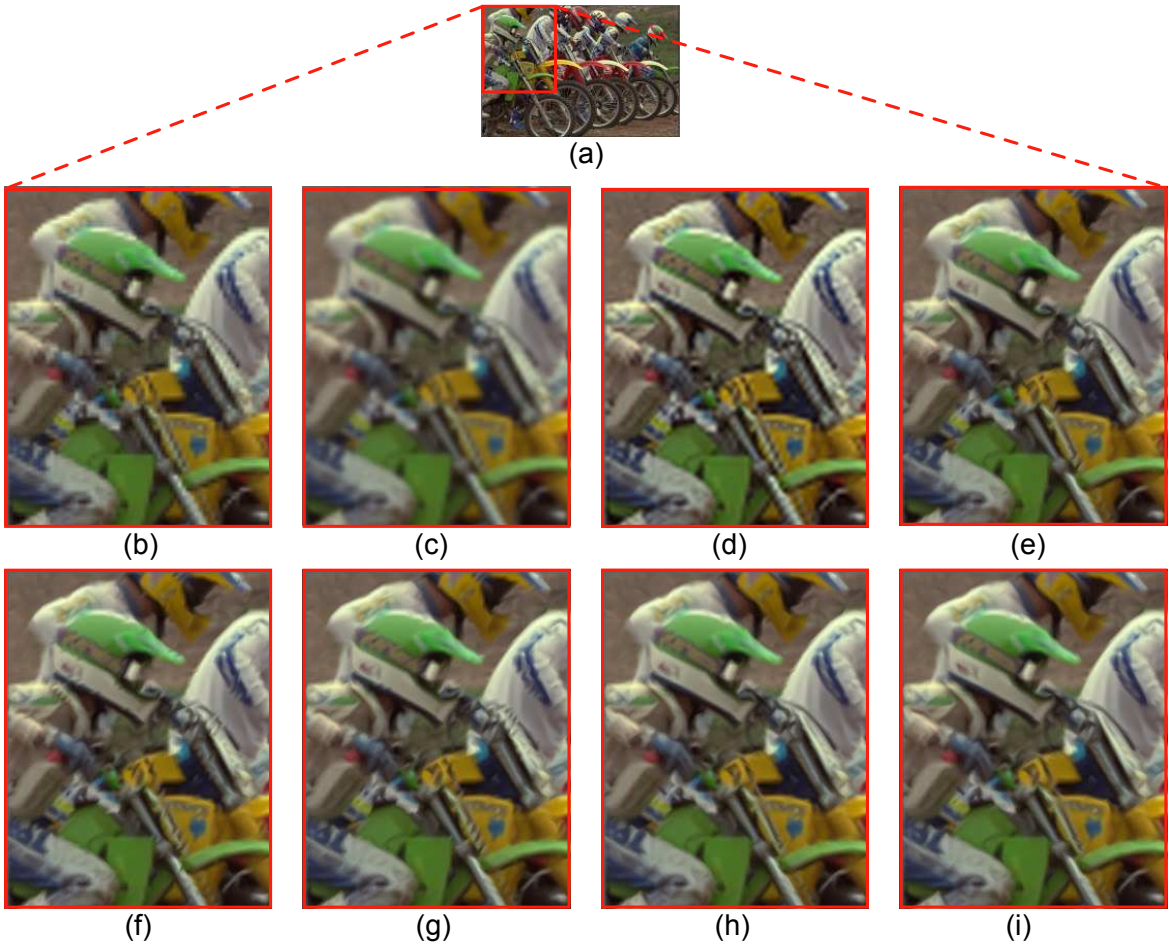
Vizuálnu kvalitu výsledkov sme hodnotili pomocou niekoľkých perceptuálnych metrík, ktoré sú bežne používané v oblasti spracovania obrazu na hodnotenie kvality. Vybranými metrikami boli: korelácia, krížová korelácia, stredná kvadratická chyba (MSE), odstup signál-šum (SNR), špičkový odstup signál-šum (PSNR), univerzálny koeficient kvality obrazu (UIQI), index štrukturálnej podobnosti (SSIM). Okrem vyššie spomínaných modifikácií sme použili štandardné rekonštrukčné techniky na porovnanie kvality rekonštrukcie: bilineárny, b-splinový, Lanczos filter. Priemerné hodnoty získané z meraní výsledkov sú uvedené v tabuľke 5. Vyššie hodnoty znamenajú vyššiu kvalitu okrem MSE, kde nižšia hodnota znamená lepšie rekonštrukčné výsledky.

Ohodnotenie kvality obrazu ľudským vizuálnym systémom nie je možné merať pomocou perceptuálnych metrík. Z tohto dôvodu sú na obrázku 9 zobrazené niektoré výsledky rôznych rekonštrukcií. Pre konvolučné techniky sú viditeľné blokové artefakty v oblastiach vysokých frekvencií.

Meranie kvality – reálna skupina obrázkov							
priemerné hodnoty	**Korelácia%**	**Krížová kor.%**	**MSE**	**SNR(dB)**	**PSNR(dB)**	**UIQI**	**SSIM**
bilinear filter	96,0128	94,8248	294,6944	17,7384	24,1055	0,3549	0,6368
b-spline filter	95,3048	93,3421	378,2071	16,5979	22,9650	0,2739	0,5958
Lanczos filter	96,3680	95,6046	249,8571	18,5692	24,9363	0,3972	0,6619
CPU DDT	96,1801	95,2550	267,7697	18,2405	24,6076	0,3712	0,6527
Basic	96,1418	95,1506	272,882	18,1580	24,5238	0,3830	0,6487
ExpRoi	96,1573	95,1779	270,689	18,1989	24,5647	0,3848	0,6500
MaxGain	96,1485	95,1618	272,204	18,1713	24,5372	0,3833	0,6491
ExpRoiMaxGain	96,1587	95,1725	270,879	18,1914	24,5572	0,3847	0,6499
MeshCanny	96,1503	95,2385	269,6155	18,2017	24,5696	0,3643	0,6512
MeshSobMax	96,1706	95,2919	266,6022	18,2560	24,6239	0,3664	0,6532
MeshSobAvg	96,1647	95,3088	267,7879	18,2344	24,6023	0,3654	0,6525
ExpRoiMeshSobMax	96,1747	95,3222	266,0252	18,2659	24,6338	0,3667	0,6535
MaxGainMeshSobMax	96,1875	95,2785	264,4245	18,2972	24,6652	0,3675	0,6545
ExpRoiMaxGainMeshSobMax	96,1905	95,2870	263,9944	18,3075	24,6754	0,3676	0,6547

Meranie kvality – vektorová skupina obrázkov							
priemerné hodnoty	**Korelácia%**	**Krížová kor.%**	**MSE**	**SNR(dB)**	**PSNR(dB)**	**UIQI**	**SSIM**
bilinear filter	97,8058	97,1848	347,9350	21,1548	23,5693	0,7315	0,9008
b-spline filter	97,0695	95,8742	507,1912	19,6697	22,0842	0,6226	0,8785
Lanczos filter	98,2447	98,0020	256,9001	22,4676	24,8820	0,6045	0,9088
CPU DDT	98,2257	97,7630	271,7720	22,2660	24,6804	0,7927	0,9194
Basic	98,1522	97,6275	286,916	22,0282	24,4229	0,7757	0,9156
ExpRoi	98,1715	97,6450	283,838	22,0616	24,4563	0,7772	0,9165
MaxGain	98,1570	97,6162	287,492	22,0196	24,4143	0,7760	0,9159
ExpRoiMaxGain	98,1982	97,6887	278,899	22,1660	24,5607	0,7782	0,9179
MeshCanny	98,1747	97,7063	279,1330	22,1577	24,5722	0,7121	0,9171
MeshSobMax	98,2002	97,7780	272,1485	22,2593	24,6737	0,7133	0,9185
MeshSobAvg	98,1972	97,7638	273,6493	22,2364	24,6508	0,7131	0,9183
ExpRoiMeshSobMax	98,2232	97,8118	267,3245	22,3284	24,7429	0,7141	0,9196
MaxGainMeshSobMax	98,2597	97,8665	259,1128	22,4350	24,8494	0,7148	0,9213
ExpRoiMaxGainMeshSobMax	98,2763	97,9157	255,4667	22,4962	24,9107	0,7155	0,9222

Tabuľka 5: Priemerné kvalitatívne výsledky merané perceptuálnymi metrikami pre reálne a vektorové obrázky.



Obrázok 9: Výsledky rekonštrukcie pri 400% zväčšení. Originálny obrázok (a), bilineárny filter (b), b-spline filter (c), Lanczos filter (d), CPU DDT rekonštrukcia (e), Basic (f), ExpRoiMaxGain (g), SobMax (h), ExpRoiMaxGain MeshSobMax (i) modifikácie.

Záver

Moderné GPU ponúkajú výpočtový výkon, ktorý významne prevyšuje výkon bežných procesorov počítača. Funkcionalita, ktorú ponúkajú sa neustále rozširuje a nové možnosti programovacieho rozhrania sa neustále vyvíjajú pre potreby programátorov. Dnešné osobné počítače obsahujú väčšinou moderné GPU, ktoré sú využívané najmä na počítačové hry a inak je ich výkon nevyužitý. V súčasnosti vznikajú rôzne tzv. GPGPU aplikácie, ťažiace z vysokého výkonu GPU. V našej práci sme sa preto zamerali na výpočty s využitím GPU, ktoré nesúvisia primárne so zobrazovaním. Špeciálne sme sa venovali dvom oblastiam a to prúdovému spracovaniu objemových dát a rekonštrukcii obrazu založenej na paralelných dátovo závislých triangu láciách.

V prvej oblasti sme navrhli využitie GPU pri spracovaní objemových dát. Dnešné GPU sa v tejto oblasti využívajú hlavne na renderovanie dát. Existujú však algoritmy využívajúce GPU aj na ich spracovanie, ale ich nevýhodou je vysoká pamäťová náročnosť, ktorá obmedzuje použitie týchto algoritmov na bežne dostupných počítačoch. Preto sme sa zamerali na prúdové spracovanie objemových dát, špeciálne na spracovanie po rezoch, kde je v pamäti počítača (GPU) len nevyhnutne potrebné množstvo dát. Tento prístup efektívnejšie využíva pamäťové zdroje počítača a umožňuje spracovanie objemových dát veľkých rozmerov na bežných počítačoch. Navrhli sme jednoduché aplikačné rozhranie, ktoré je univerzálne a je ho možné využiť viacerými dostupnými hardvérovými prostriedkami počítača ako je CPU, prípadne jeho inštrukčná sada SSE a moderné GPU. Ďalej sme uviedli viaceré všeobecné techniky využívajúce funkcionalitu súčasných GPU pri spracovaní objemových dát. Tieto techniky sme využili pri implementovaní algoritmov spracovávajúcich objemové dáta na GPU v prúdovom režime. Jednalo sa o dva filtre, všeobecný filter na výpočet separovateľnej a neseparovateľnej konvolúcie a filter detekujúci tubulárne štruktúry v objemových dátach. Implementované filtre sme porovnávali s rovnakými algoritmami využívajúcimi CPU, prípadne SSE. Dosiahli sme výrazne urýchlenie, ktoré záviselo od konkrétnych algoritmov. Implementované algoritmy sme využili aj v reálnej aplikácii, v ktorej sa tiež dosiahlo výrazné urýchlenie spracovania dát. V tejto aplikácii sa využívali všetky dostupné zdroje počítača (GPU, SSE), čo sa ukázalo ako výhodné riešenie pri spracovaní objemových dát.

V druhej oblasti sme využili výkon GPU pri výpočte dátovo závislých triangulácií. Pre tento výpočet sme navrhli všeobecný paralelný algoritmus založený na Lawsonovom optimalizačnom procese. Pomocou paralelnej verzie je možné efektívnejšie využiť aj dnešné viacjadrové CPU. Tento paralelný algoritmus je univerzálny, takže pri optimalizácii je možné použiť rôzne cenové funkcie. Výsledná triangulácia bola využitá pri rekonštrukcii obrazu, konkrétne pri zväčšovaní. Základná verzia algoritmu bola vykonávaná 8-krát rýchlejšie oproti sekvenčnej verzii na CPU. Pri rekonštrukcii obrazu sme sa zamerali na kvalitu výslednej rekonštrukcie, najmä na korektnú rekonštrukciu hranových oblastí, na ktoré je pozorovateľ najviac senzitívny. Navrhli sme viaceré modifikácie základného algoritmu s cieľom zvýšiť výslednú kvalitu rekonštrukcie. Tieto modifikácie ovplyvňovali priamo výpočet algoritmu, prípadne spracovávali už rekonštruovaný obraz. Viacerými modifikáciami sme dosiahli výraznejšie zvýšenie kvality rekonštrukcie s malým dopadom na výsledný čas. Merania preukázali, že predstavený algoritmus a jeho modifikácie sú konkurencieschopné v porovnaní s klasickými rekonštrukčnými algoritmami založenými na konvolučných technikách.

Literatúra

[1] GPGPU. GPGPU General-Purpose Computation on Graphics Hardware [Internet]. `http://www.gpgpu.org`, 2010. [cited Apr. 20, 2010].

[2] C. Charles Law, William J. Schroeder, Kenneth M. Martin, and Joshua Temkin. A multi-threaded streaming pipeline architecture for large structured data sets. In *VIS '99: Proceedings of the conference on Visualization '99*, pages 225–232, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[3] Andrej Varchola, Anton Vaško, Viliam Solčány, Leonid I. Dimitrov, and Miloš Šrámek. Processing of volumetric data by slice- and process-based streaming. In *AFRIGRAPH '07: Proceedings of the 5th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa*, pages 101–110, New York, NY, USA, 2007. ACM.

[4] Miloš Šrámek and Leonid I. Dimitrov. f3d – a file format and tools for storage and manipulation of volumetric data sets. In *In 1-st International Symposium on 3D Data Processing, Visualization and Transmission, Padova, Italy, June 19-21, 2009*, pages 368–372, 2002.

[5] Alejandro F. Frangi, Wiro J. Niessen, Koen L. Vincken, and Max A. Viergever. Multiscale vessel enhancement filtering. In *MICCAI'98 - Medical Image Computing and Computer-Aided Intervention*, pages 130 – 137, 1998.

[6] N. Dyn, D. Levin, and S. Rippa. Data Dependent Triangulations for Piecewise Linear Interpolation. *IMA Journal of Numerical Analysis*, 1(10):137–154, 1990.

[7] L. Alboul, G. Kloostreman, C. Traas, and R. V. Damme. Best data-dependent triangulations. *Journal of Computational and Applied Mathematics*, 119(1-2):1–12, 2000.

[8] J. Brown. Vertex Based Data Dependent Triangulations. *Computer Aided Geometric Design*, 8(3):239–251, 1991.

[9] X. Yu, B. S. Bryan, and T. W. Sederberg. Image Reconstruction Using Data-Dependent Triangulation. *Computer Graphics and Applications*, 21(3):62–68, 2001.

[10] Michal Červeňanský, Zsolt Tóth, Juraj Starinský, Andrej Ferko, and Miloš Šrámek. Parallel GPU-based data-dependent triangulations. *Computers & Graphics*, 34:125–135, 2010.

Vedecké aktivity

Publikácie

Michal Červeňanský, Zsolt Tóth, Juraj Starinský, Andrej Ferko, Miloš Šrámek. *Parallel GPU-based data-dependent triangulations*, In Computers & Graphics 34, 2010, pages 125-135, doi:10.1016/j.cag.2010.01.001

Július Parulek, Miloš Šrámek, Michal Červeňanský, Marta Novotová, Ivan Zahradník, *A Cell Architecture Modeling System Based on Quantitative Ultrastructural Characteristics*, In the book on "Systems Biology-by Humana Press, Springer Verlag, Series: Methods in Molecular Biology, Vol. 500, Ivan V. Maly (Ed.), ISBN: 978-1-934115-64-0, pages 289–313, © Springer-Verlag Berlin Heidelberg 2009

Miloš Šrámek, Leonid I. Dimitrov, Matúš Straka, Michal Červeňanský. *The f3d tools for processing and visualization of volumetric data.* Journal of Medical Informatics and Technologies, pages MIP-71-MIP-79, 2004

Matúš Straka, Michal Červeňanský, Miloš Šrámek, Alexandra La Cruz, Eduard Gröller, Arnold Köchl, Dominik Fleischmann. *The VesselGlyph: Focus & Context Visualization in CT-Angiography.* IEEE Visualization 2004 Conference Proceedings, Austin, Texas, Oct 2004

Kvalifikačné práce

Michal Červeňanský. Objemové zobrazovanie v reálnom čase s použitím grafického hardvéru. *Práca k dizertačnej skúške*, FMFI Univerzita Komenského, Bratislava, 2007.

Michal Červeňanský. Vybrané techniky objemového zobrazovania s použitím komerčných grafických akcelerátorov . *Rigorózna práca*, FMFI UK, Bratislava, 2006

Michal Červeňanský. Využitie komerčných grafických akcelerátorov pre spracovanie a vizualizaciu objemových dát. *Diplomová práca*, FMFI UK, Bratislava, 2004

Grantová činnosť

APVV - Tools for processing and visualization of tomographic and confocal data - APVV-20-056105, (2006-2009)