VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

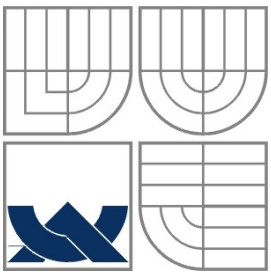
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VÝPOČET OPTICKÉHO POLE V GP-GPU

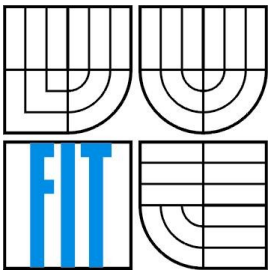
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. ERIK SRNEC
AUTHOR

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

VÝPOČET OPTICKÉHO POLE V GP-GPU

OPTICAL FIELD CALCULATIONS IN GP-GPU

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. ERIK SRNEC

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Dr. Ing. PAVEL ZEMČÍK

BRNO 2012

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2011/2012

Zadání diplomové práce

Řešitel: **Srnec Erik, Bc.**
Obor: Počítačová grafika a multimédia
Téma: **Výpočet optického pole v GP-GPU**
Optical Field Calculations in GP-GPU
Kategorie: Zpracování obrazu

Pokyny:

1. Prostudujte dostupnou literaturu na téma OpenCL a platformy pro OpenCL aplikace, zaměřte se na GP-GPU platformy. Prostudujte též nástroje pro OpenCL
2. Prostudujte algoritmy tvorby optického pole pro syntézu hologramů, případně algoritmy viditelnosti a zvažte možnosti implementace v OpenCL.
3. Navrhněte vhodný způsob reprezentace optického pole a implementace vybraného algoritmu.
4. Demonstrujte výstupy a vlastnosti implemetace.
5. Diskutujte dosažené výsledky a zvažte možnosti dalšího pokračování práce.

Literatura:

- Dle pokynů vedoucího

Při obhajobě semestrální části diplomového projektu je požadováno:

- Bez požadavků.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Zemčík Pavel, doc. Dr. Ing.**, UPGM FIT VUT
Datum zadání: 19. září 2011
Datum odevzdání: 23. května 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato práce popisuje poměrně novou techniku určenou na psaní vysoko paralelních programů, které název je OpenCL. Je určena jak pro GPU, tak i pro CPU a jiné paralelní procesory. Knižnice využívá architekturu procesoru, která obsahuje velké množství malých jader. Tyto jádra nejsou tak komplexní jako klasické procesory a proto se hodí pro výpočty, kterých je sice mnoho a jsou jednoduchá. Právě tato vlastnost by mohla za určitých podmínek urychlit výpočet hologramu, konkrétně výpočet optického pole. Samotný výpočet je sice jednoduchý, ale množství zpracovaných údajů je veliké a proto je pomalé. V práci nechybí taky základní pojmy vysvětlení optické a digitální holografie.

Abstract

This work describes a relatively new technique designed to write highly parallel programs, that name is OpenCL. It is intended for both GPU and CPU and other parallel processors. Libraries used by the processor architecture, which includes a large number of small cores. These cores are not as comprehensive as conventional processors and is therefore suitable for calculations, which are many and they are simple. It is this property could, under certain conditions, accelerate the calculation of the hologram, namely the calculation of the optical field. While the calculation itself is simple, but the amount of processed data is large and therefore slow. The work also contain the basic concepts of explanation of optical and digital holography.

Klíčová slova

OpenCL, holografia, hologram, GP-GPU, optické pole, paralelizmus

Keywords

OpenCL, holography, hologram, GP-GPU, optical field, parallelism

Citace

Erik Srnec: Výpočet optického pole v GP-GPU, diplomová práce, Brno, FIT VUT v Brně, 2012

Výpočet optického pole v GP-GPU

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením doc. Dr. Ing. Pavla Zemčíka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Erik Srnec
23. května 2012

Poděkování

Veľmi by som sa chcel poďakovať doc. Dr. Ing. Pavlovi Zemčíkovi za podporu a odbornú pomoc pri riešení tejto diplomovej práce. Taktiež by som chcel poďakovať všetkým ostatným, ktorí mi pomáhali.

Obsah

1 Úvod

Už od nepamäti sa ľudstvo snaží zreprodukovať rôzne objekty, či už statické alebo dynamické v pohybe a majú na to rôzne dôvody. Začalo to kamennými tabuľami, na ktoré sa muselo pracne tepať a obrázky boli tým pádom jednoduché bez presnejších rysov. Pokračovalo to papyrusmi, ktoré boli prenosnejšie, lepšie a jednoduchšie sa na ne písalo aj kreslilo. Všetky tieto a iné spôsoby však majú značnú nevýhodu, pretože zobrazujú len statický text alebo obrázky. Až v 20. storočí sa začalo zachytávanie a reprodukcia dynamického obrazu. Najskôr analógovým spôsobom, no v dnešnej dobe je zobrazovanie predovšetkým pomocou digitálnych obrazoviek. Aj keď si môžeme nahrať a pustiť ľubovolné video, stále vidíme obraz v rovine. Preto je snaha o vytvorenie realistického trojrozmerného obrazu. Momentálne je to realizovateľné dvoma spôsobmi. Prvým sú tzv. 3D zobrazovacie zariadenia. Táto technológia ale nie je veľmi pohodlná, pretože aby sme videli trojrozmerný obraz musíme mať na sebe okuliare a keďže sa využíva nedokonalosť ľudských očí, týmto spôsobom sa unavujú a niektorí môžu mať bolesti hlavy. Druhou možnosťou je využitie holografie. Má niekoľko výhod oproti prvej možnosti, ale aj nevýhod. Medzi výhody patrí skutočné vyobrazenie modelu, takže je viditeľný zo všetkých uhľov, nie sú nutné žiadne dodatočné pomôcky a pre ľudí nenastávajú nežiadúce vedľajšie efekty. Hlavná nevýhoda spočíva vo výpočtoch potrebných na záznam a reprodukciu objektu, ktoré sú veľmi náročné pre dnešné počítače. Práve akcelerácia týchto výpočtov bude predmetom ďalších kapitol.

Najskôr predstavím OpenCL[1], dostupné nástroje a platformy. OpenCL je výsledok skupiny Khronos , ktorá vytvára otvorené štandardy a API predovšetkým na paralelné a grafické výpočty. Do tejto skupiny patria firmy ako Apple, AMD, Intel, Nvidia, Ericsson, Sun a iné. Medzi doposiaľ vytvorenými najznámejšími štandardmi patrí napr. OpenGL, OpenVG alebo WebGL [1]. Všetky využívajú architektúru GPU na urýchlenie výpočtov, ktoré sú však určené na vykresľovanie grafických prvkov aplikácií a neumožňujú využitie pre bežné aplikácie. Práve toto špecifické určenie sa zmenilo príchodom nového štandardu s názvom OpenCL. Dnešné GPU obsahujú veľké množstvo malých procesorových jednotiek, ktoré sa využívajú napr. na renderovanie obrázkov, dekódovanie videa a iné paralelné grafické výpočty. Tento výkon a paralelizmus sa výborne hodí na spracovanie veľkého množstva jednoduchých výpočtov pri vytváraní optického pola a následného hologramu. V tejto práci sa budem predovšetkým venovať GP-GPU[2], teda paralelnými výpočtami na GPU.

Vysvetlenie holografie, vytváranie a spracovanie hologramu pomocou počítačov bude predmetom nasledujúcej kapitoly. Princíp holografie bol objavený maďarským fyzikom Dennisom Gaborom v roku 1947 za čo dostal v 1971 nobelovú cenu [2]. Jednalo sa o náhodný objav, ktorý vznikol pri vylepšovaní elektrónového mikroskopu. Od vynájdenia ubehlo niekoľko rokov, kým sa mohol tento objav vyskúšať v praxi. Na vytvorenie hologramu je potrebné koherentné žiarenie vysielané na model, ktorý chceme reprodukovať a také žiarenie prišlo až s vynálezom laseru, ktorý splňoval potrebné požiadavky. Odvtedy sa princíp síce nezmenil, ale použité technológie a spôsoby sa zmenili a to k lepšiemu, čo prináša možnosti zobrazenia väčších a rozsiahlejších scén, ktoré sú rýchlejšie vytvorené.

1 OpenCL (Open Computing Language) – štandart pre paralelné programovanie heterogenných počítačových systémov

2 General-purpose computing on graphics processing units – technika použitia grafickej karty na tvorbu aplikácií pôvodne určených pre počítačové procesory

2 OpenCL

Táto kapitola zobrazuje prehľad o frameworku[3] OpenCL, nástrojoch a platformách, na ktorých je možný beh OpenCL. Ako bolo vyššie spomenuté, OpenCL je momentálne určené predovšetkým na využitie v GPU, ale už nachádza priamu podporu aj v CPU, prípadne iných paralelne spracovávajúcich procesoroch. Táto univerzálnosť a otvorenosť celého rozhrania je kľúčová pre široké nasadenie a využitie. Už v minulosti aj nedávnej súčastnosti boli snahy o vytvorenie rôznych architektúr, frameworkov, či API[4]. Za architektúry je to napríklad Larrabee od firmy Intel, ktorá však zanikla ešte pred predstavením [3]. Ďalšia už trochu staršia architektúra je CUDA od firmy Nvidia [4]. Nie je to však len architektúra, ale Nvidia vydala aj vlastný framework nazvaný CUDA C, ako už názov naznačuje, ide o rozšírenie jazyka C resp. C++ o možnosť vyžitia tejto architektúry. Ponúka podobné možnosti ako OpenCL, priníp aj štrukuktúra sú veľmi podobné, ale CUDA C nie je podporovaná inými výrobcami, kvôli uzavretosti. Okrem toho môže byť na CUDA implementovaný API DirectCompute od firmy Microsoft [5] alebo práve framework OpenCL. Firma AMD prišla s „ATI Stream Technology“ [6] (v súčasnosti premenovanou na AMD Accelerated Parallel Processing (APP) Technology [7]) architektúrou, ktorá je akýmsi prepojením grafickej karty a procesorom v takom zmysle, že grafická karta má na starosti paralelnú časť výpočtov a procesor zas sekvenčné vykonávanie príkazov. Táto architektúra však nemá také široké spektrum ako CUDA a podporuje len OpenCL. Minoritnú časť tvoria aj rôzne API pre programovacie jazyky určené na paralelné programovanie na GPU ako napríklad menej známe BrookGPU [8] alebo Sh [9]. Kvôli tejto rozmanitosti, prípadne uzavretosti sa paralelné programovanie na GPU zatiaľ veľmi nerozšírilo a roztrieštenosť má vyriešiť práve framework OpenCL. Keďže sú v súčastnosti najviac dostupné GP-GPU architektúry CUDA, ATI Stream Technology a Intel Visual Computing, tak ďalšie časti budú prezentovať hlavné súčasti týchto architektúr.

2.1 CPU - GPU

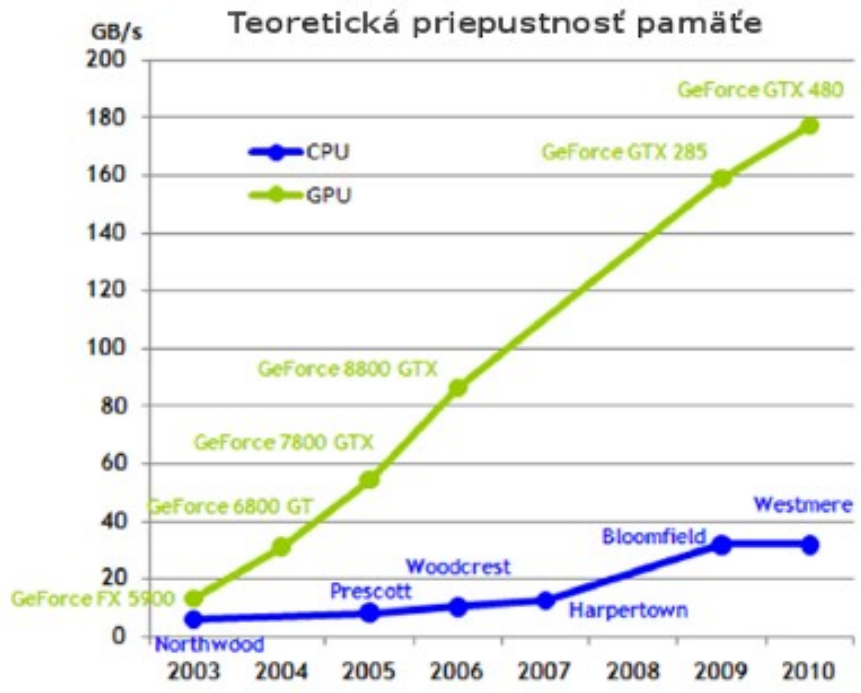
V dnešnej dobe používa drvivá väčšina počítačov viacjadrové CPU. Počet jadier je rôzny, bežne sa pohybuje od dvoch do štyroch, v serverových je počet obvykle osem až viac jadier na CPU. Paralelizmus je teda možný aj na klaisckých procesoroch, ale malý počet fyzických jadier je vhodný len na určité operácie a úlohy. Prípadne sa dajú tieto fyzické jadrá rozdeliť na väčší počet virtuálnych jadier, táto možnosť však nie je veľmi efektívna práve kvôli rozdeleniu, ktoré je riešené softvérovo a musí sa použiť časť výkonu procesora. Na takéto paralelné programovanie sa zrejme najčastejšie využíva API OpenMP [10] Naopak GPU boli už od začiatku vymyslené a špecializované na grafické vykresľovanie a s tým spojené špecifické úlohy ako napríklad renderovanie, počítanie veľkého počtu pixelov, vertexov a pod., čo sú procesy spracovávané paralelne. Postupne sa vyvinuli do dnešných vysoko paralelných, viacvláknových a viacjadrových GPU s veľkou dátovou priepustnosťou pamäte. Toto všetko je možné hlavne vďaka mnohým výpočtovým jednotkám umiestneným na grafickom čipe, čo zobrazuje obrázok 2.1.

3 Framework – viacero API spojených do jedného celku

4 API (Application Programming Interface) – rozhranie pre programovanie aplikácií

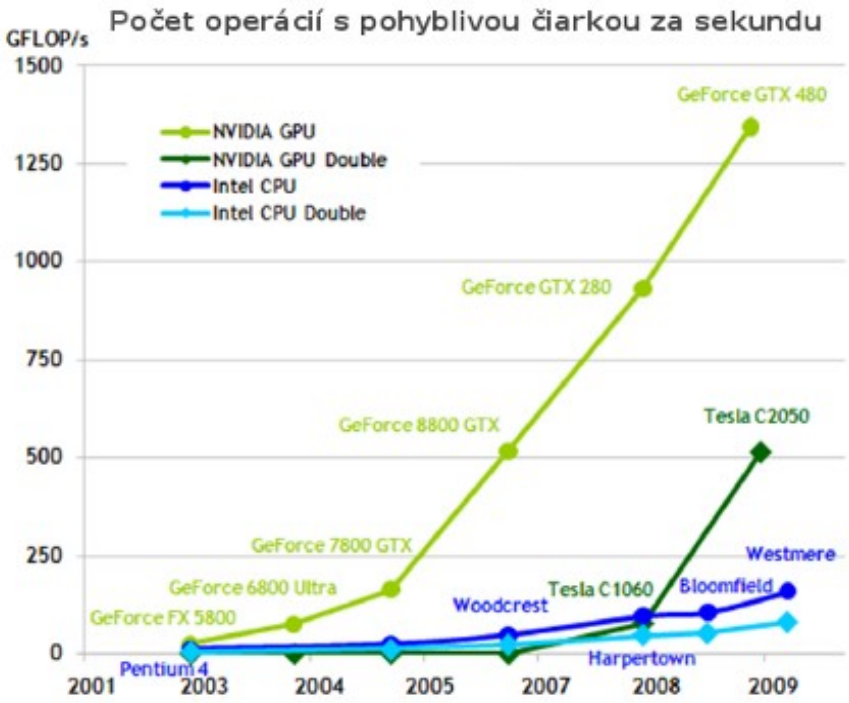
Obrázok 2.1: Porovnanie počtu tranzistorov na CPU a GPU. Obrázok prevzatý z [11].

Z obrázku je taktiež vidno, že GPU sa viac ako o riadenie toku a ukladanie dát do medzipamäte starajú o ich spracovanie. To je ďalšia vec, ktorá je výhodnejšia oproti klasickým CPU.
Na obrázku 2.2 je rozdiel teoretickej priepustnosti pamäte v GB za sekundu medzi CPU firmy Intel a GPU firmy Nvidia za posledných sedem rokov. Za túto dobu je výrazný nárast priepustnosti v GPU a to viac ako desaťnásobný, v niektorých prípadoch dokonca 30 násobný, ak porovnáme GeForce FX 5500 [12] a GeForce GTX 480 [13]. Zatiaľ čo priepustnosť CPU sa zvýšila len zhruba päťnásobne na hodnotu okolo 25 GB/s, čo je hodnota GPU z pred šiestich rokov.



Obrázok 2.2: Teoretická priepustnosť pamäte. Obrázok inšpirovaný z [11].

Na obrázku 2.3 je porovnanie počtu operácií s pohyblivou desatinnou čiarkou vyjadrenou GFLOPS za sekundu opäť medzi produktami firiem Intel a Nvidia. Z grafu vidno výrazný rozdiel medzi GPU a CPU aj v double rozsahu.

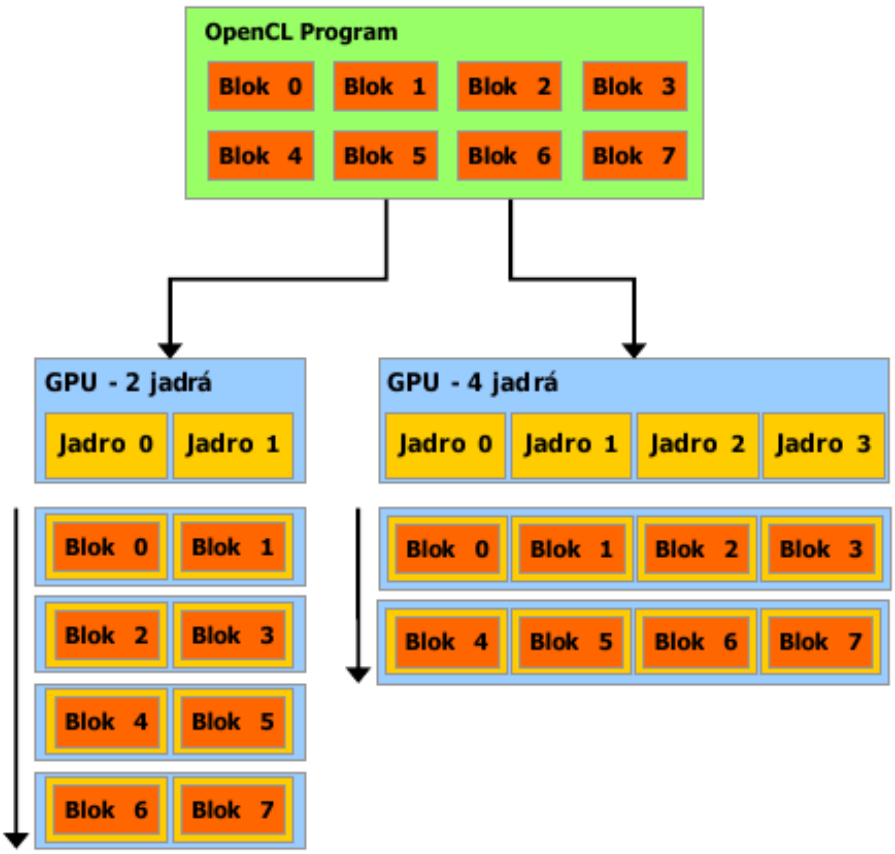


Obrázok 2.3: Počet operácií s pohyblivou čiarkou. Obrázok inšpirovaný z [11].

Z predchádzajúcich obrázkov a grafov je viditeľný rozdiel medzi CPU a GPU. Tým, že sa nevenuje GPU riadeniu a správe pamäte, ale viac výpočtom, umožňuje oveľa väčšiu priepustnosť a počet operácií s pohyblivou čiarkou, čo robí GP-GPU tak sľubnou technológiou. Toto si uvedomujú aj výrobcovia procesorov a preto je v súčasnosti trendom spájanie CPU a GPU do jedného heterogénneho celku. Samozrejme nejde primárne len o možnosť paralelných výpočtov, ale hlavne zníženie energetickej spotreby a samozrejme väčší podiel na trhu. Pre zákazníkov je však len dobré, keď výrobcovia ponúkajú aj takéto nové technológie.

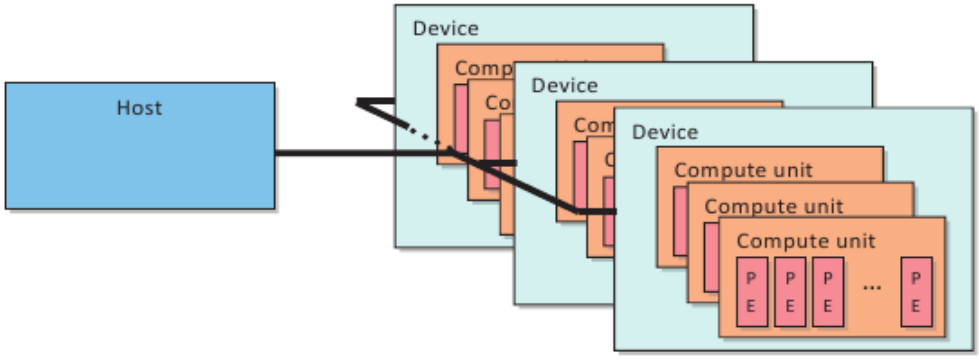
2.2 OpenCL na hardvérovej úrovni

Nástup viacjadrových CPU a GPU so sebou prináša aj problém v podobe veľkého rozdielu v počte jadier. Je potrebné písať aplikácie, ktoré dokážu v plnej miere využiť všetky dostupné jadrá na čipe. Tento problém však rieši samotná architektúra, na ktorej môže byť OpenCL prevádzkovaná. Je to riešené pomocou troch abstraktných vrstiev a to hierarchiou skupín vlákien, hierarchiou zdielanej pamäte a ich synchronizáciou. Tieto abstrakcie poskytujú paralelizmus s malými dátami a vláknový paralelizmus, ktoré sú vnorené do paralelizmu zhlukov dát a paralelizmu úloh. To vedie programátora k rozdeleniu údajov na väčšie celky obsahujúce menšie časti, ktoré môžu byť nezávisle paralelne spracované v blokoch vlákien a tie ešte rozdeliť do jednotlivých vlákien v bloku. Tento rozklad umožňuje automatickú škálovateľnosť, takže každý blok vlákien môže byť spustený na niektorom z dostupných jadier a je jedno, v akom poradí sa bude blok púšťať, či už postupne alebo naraz. Preto je skompilovaný program v OpenCL bez problémov spustiteľný na akomkoľvek paralelnom procesore a prispôsobí sa danému počtu jadier automaticky, resp. podľa nastavenia v programe. Automatické rozdelenie možno vidieť na obrázku 2.4.



Obrázok 2.4: Rozdelenie viacvláknového programu podľa počtu jadier. Obrázok inšpirovaný z [11].

Každý výrobca si rieši konkrétne veci po svojom. Vo všeobecnosti ale môžeme rozdeliť platformu na niekoľko hierarchických častí, ktoré sú zhodné pre všetky implementácie, čo je znázornené na obrázku 2.5.



Obrázok 2.5: Zobrazenie všeobecnej hardvérovej platformy. Obrázok prevzatý z [14].

Procesor (Processin Element)
Procesor na obrázku predstavuje jeden skalárny procesor[5]. Dokážu skutočne len jednoduché úlohy. Počet týchto procesorov v jednom multiprocesore sa líši nielen od výrobcu ale aj od modelového radu, či generácie. V dnešnej dobe sa tento počet zmenšuje, čo je výhodnejšie, pretože menej procesorov bude zdielať takú istú veľkú pamäť, počet registrov a taktiež ostatných jednotiek.

Výpočtová jednotka (Compute unit)
Výpočtová jednotka, niekedy nazývaný tiež Stream Multiprocessor. Počet týchto výpočtových jednotiek je na zariadení je viacero, na dnešných najmodernejších kartách za dokonca táto hranica približuje k tisíc. Každá takáto jednotka obsahuje niekoľko ďalších prvkov. Konkrétne sú to najdôležitejšie skalárne procesory (niekedy tiež Stream Processors alebo najčastejšie jadrá), ktorých množstvo závisí od konkrétneho zariadenia, preto ten názov multiprocesor. Ďalej je to zdielaná pamäť pre procesory v celej výpočtovej jednotke, registre a prípadné ďalšie špeciálne a pomocné jednotky, ako napríklad podporný procesor pre dátový typ *double.*

Zariadenie (Device)
Zariadenie predstavuje jedno fyzické zariadenie, ktoré môže byť či už GPU alebo aj CPU a pri vhodnom rozdelení úloh môžu pracovať spoločne. Nezahrňujú sa tam však žiadne jadrá alebo iné podmnožiny. Počet zariadení je obmedzený len na počet slotov na CPU a počet slotov na GPU.

Hostiteľská časť(Host)
Nad celým týmto je tzv. Hostiteľská časť, obvykle CPU, ktorá má na starosti celú obsluhu vykonávania programu medzi zariadeniami.

2.2.1 Pamäť a prístup k pamäti

Významným prvkom v architektúrach pre OpenCL je pamäť, na ktorej závisí rýchlosť a tým aj dostatočné využitie paralelizmu. Pamäť je hierarchicky rozdelená na viacero úrovní podľa rýchlosti a prístupu.

Registre
Najrýchlejšia a zároveň najmenšia pamäť sú takzvané registre, niekedy nazývaná aj privátna pamäť. Táto pamäť je obsadzovaná prekladačom a je umiestnená priamo na čipe, pričom programovo do nej nie je možné zapisovať ani čítať ju. Alokáciu a rozdelenie zabezpečuje kompilátor. Je určená pre interné účely procesora.

Zdielaná pamäť
Zdielaná pamäť je porovnateľne rýchla, je zdielaná medzi vláknami v jednom bloku a umiestnená na čipe. Zdielaná pamäť sa rozdeľuje v jednom bloku na viaceré časti tzv. banky, ku ktorým sa môže pristupovať súčasne. Ak viacero vlákien požaduje rovnakú banku, tak nastáva konflikt a takéto požiadavky sa musia rozdeliť, aby malo každé vlákno svoju banku. Týmto konfliktom je potrebné sa vyhnúť tým, že zistíme ako sa adresuje pamäť, pretože znižujú priepustnosť pamäte. Pokiaľ neexistuje konflikt, tak je dokonca zdielaná pamäť rovnako rýchla ako registre.

5 Skálarny procesor – procesor, ktorý vykonáva jednu operáciu s jedným číslom v tom istom čase

Lokálna pamäť

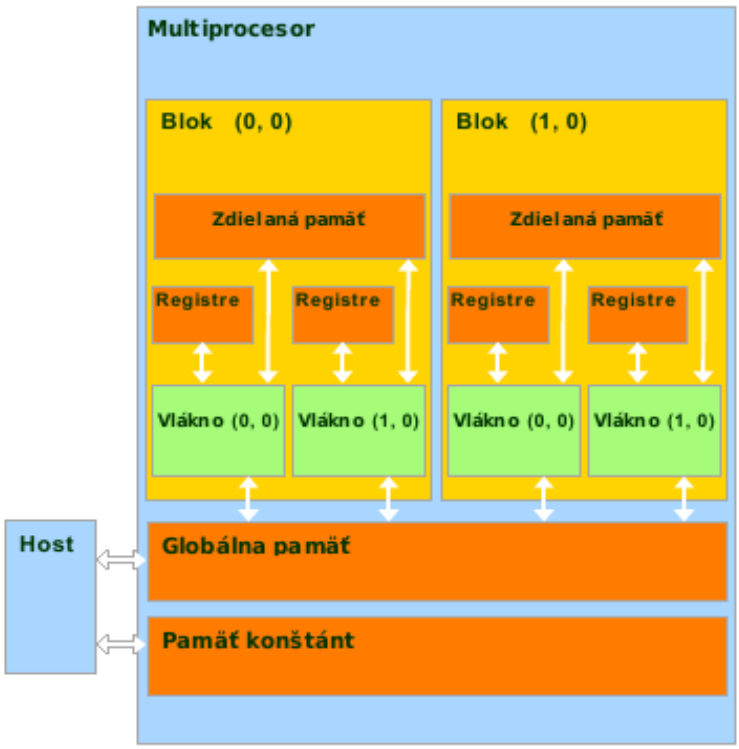
Je podobná globálnej, má malú šírku pásma, je však omnoho rýchlejšia, pretože sa nachádza priamo na čipe a niekedy už obsahujú aj cache. Ukladajú sa do nej veľké štruktúry alebo polia, ktoré sú príliš veľké pre veľkosť registra, ďalej polia, pri ktorých nemožno určiť, či sú indexované s konštantným množstvom prvkov alebo iné premenné ak sa používa viac registrov, ako je dostupných. Rieši prístup do nezarovnanej globálnej pamäti, čo je spomalujúci jav. Organizácia lokálnej pamäte je taká, aby boli po sebe idúce 32 bitové slová prístupné pre po sebe idúcich vlákien.

Globálna pamäť

Býva veľmi veľká, rádovo v stovkách MB až GB, nemá cache a je najpomalšia. Je viditeľná pre všetky vlákna v zariadení, ale obvykle je spracovanie vlákien v blokoch, ktoré potom pristupujú k tejto pamäti. Spracováva iba 32, 64 alebo 128 bytové inštrukcie, preto musia byť zarovnané na konkrétnu dĺžku. Globálna pamäť podporuje prácu s 1, 2, 4, 8 alebo 16 bytovými slovami a ich násobky [11]. Ak je však veľkosť iná, tak je zostavených viac inštrukcií s prístupovými vzormi, aby tieto inštrukcie nesplývali. Preto sa odporúča používať len typy odpovedajúce dĺžkou slov. Ďalej sa odporúča používať zarovnané štruktúry podľa 8 alebo 16 bytov. Pri nezarovnaní môže dôjsť k chybným výsledkom. Lepšia priepustnosť sa dá dosiahnuť aj dvojrozmerným poľom, kde každé vlákno so súradnicami (t_x, t_y) používa adresu pre prístup, ktorá je uložená v dvojrozmernom poli.

Pamäť konštánt

Nachádza sa v zariadení a nachádza sa v nej aj cache. Môže byť súčasť globálnej pamäte, ale môže to byť aj samostatná pamäť ako je to u firmy NVIDIA.



Obrázok 2.6: Hierarchia pamätí v multiprocesore. Obrázok inšpirovaný z [11].

Na obrázku 2.6 vidno hierarchické usporiadanie jednotlivých typov pamätí na jednom multiprocesore.

Pamäť	Umiestnenie	Cache	Prístup	Úroveň
Registre (privátna)	Na čipe	N/A	Čítanie/Zápis	Vlákno
Zdielaná	Na čipe	N/A	Čítanie/Zápis	Blok vlákien
Lokálna	Mimo čipu	Áno/Nie	Čítanie/Zápis	Vlákno
Konštánt	Mimo čipu/Na čipe	Áno	Čítanie	Multiprocesor
Globálna	Mimo čipu	Nie	Čítanie/Zápis	Multiprocesor
Textúry	Mimo čipu	Áno	Čítanie	Multiprocesor

Tabuľka 2.1: Prehľad jednotlivých pamätí [15].

2.3 OpenCL na softvérovej úrovni

V predchádzajúcej časti bola predstavená hardvérová časť v tejto bude naopak ľahký úvod do toho, ako sa pracuje s OpenCL zo softvérového pohľadu, čo sa na tvorbu môže a naopak musí použiť.
Aplikácia napísaná s frameworkom OpenCL sa rozdeľuje na dve hlavné časti a to na kód pre hostiteľskú časť a kód pre samotné zariadenie.

Hostiteľská časť, ktorá je obvykle spúšťaná na procesore ako Host z obrázka 2.5 môže byť napísaná v programovacích jazykoch ako sú C, C++, Java, Python a podobne. Pre všetky tieto a mnohé ďalšie programovacie jazyky existuje potrebné OpenCL API. Táto časť posiela príkazy z hostiteľa pre vymieňanie dát z a do zariadenia a taktiež spúšťa vykonávanie jadrového kódu.

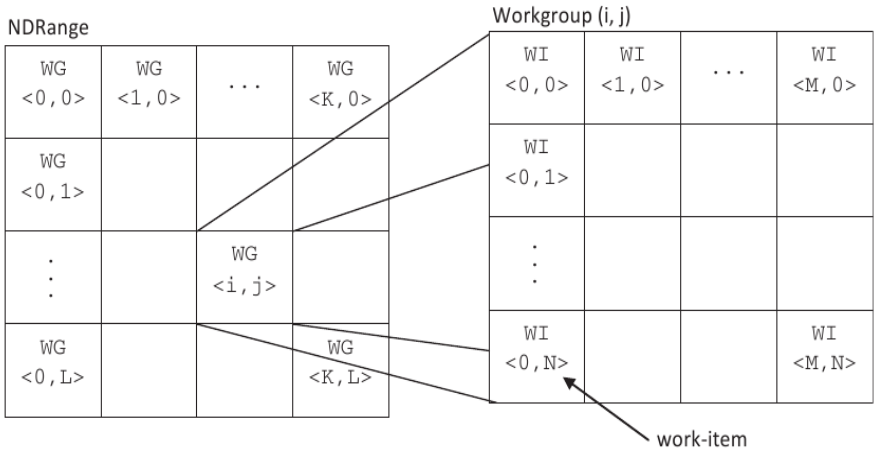
Kód, ktorý sa vykonáva na zariadení sa taktiež nazýva *jadro* alebo po anglicky *kernel* a musí byť napísaný v jazyku zvanom OpenCL C. Tento jazyk priamo vychádza zo štandardu jazyka C99, má však značné obmedzenia, na druhú stranu pridáva rôzne rozšírenia, napr. podporu vektorových dátových typov, špecifikovanie adresného priestoru (kvôli hierarchicky rozdelenej pamäti, viď vyššie), podpora pre paralelné vykonávanie, podpora pre prácu s obrázkami alebo zabudované matematické funkcie a operácie s nimi [16].

Tak ako pamäť aj spôsob vykonávania je hierarchický. Architektúra je navrhnutá ako SIMD[6] a VLIW[7]. V OpenCL je SIMD označované skôr ako SIMT (Single Instruction, Multiple Thread), čo znamená, že dáta sú spracovávané vo viacerých vláknach nezávisle na jadrách zariadenia. Vďaka architektúre VLIW je zas možné vykonávanie aj out-of-order, teda vlákna na seba nemusia čakať na dokončenie.
Jadro OpenCL tak musí spracovať veľké množstvo vlákien. Keď hosť odošle všetky vlákna, zariadenie si ich zarovná a rozdelí do mriežky zadanej veľkosti, tzv. pracovných skupín, pričom mriežka môže byť jedno, dvoj alebo trojrozmerná a jej veľkosť v podstate nie je obmedzená. Počet

6 SIMD (Simple instruction, multiple data) – hardvérová architektúra pre paralelné spracovanie, jedna inštrukcia spracováva mnoho dátových tokov

7 VLIW (Very long instruction word) – hardvérová architektúra umožňujúca spracovanie vlákna, aj keď sa ešte neskončilo vykonávanie predchádzajúceho

vlákien v pracovnej skupine závisí od konkrétneho zariadenia, môže byť taktiež jedno alebo dvoj rozmerná, veľkosť však musí byť mocninou dvojky. Ak sa veľkosť nezadá explicitne, tak jadro si ju samo určí a takisto aj dimenziu automaticky, čo ale môže mať negatívny vplyv na výkonnosť, pretože napr. pre dvojrozmernú mriežku je vhodnejšie aj pracovné skupiny rozdeliť na dvojrozmernú, inak sa stane, že aj mriežka bude nakoniec v skutočnosti len jednorozmerná. Medzi skupinami nie je možné zdielať pamäť a vymieňať si tak dáta, lebo model OpenCL je navrhnutý tak, aby boli pracovné skupiny nezávislé a mohli tak radšej riešiť úlohy na nižšej úrovni. Jednotlivé pracovné skupiny sú obvykle vykonávané na multiprocesore a interne sa rozdeľujú podľa výrobcu na tzv. *warps* (NVIDIA), ktoré obsahujú 32 vlákien [11] alebo *wavefront* (AMD) so 64 vláknami [6], a sú v takomto počte aj naraz vykonávané pomocou vyššie spomínanej architektúry SIMT. Z externého pohľadu však pracovné skupiny obsahujú už len jednotlivé vlákna alebo aj inak nazývané pracovné jednotky. Vlákna v skupine si môžu medzi sebou zdieľať pamäť a výpočty niektorých vlákien tak môžu v prípade potreby dostať aj dáta už predtým spracované iným vláknom, prípadne je dostupná aj funkcia bariéra, ktorá umožní počkať vo vykonávaní vlákna kým sa nedokončí iné vlákno. Ak by bolo potrebné, môžu vlákna využiť aj privátnu pamäť umiestnenú v multiprocesore. Samotné vlákna sa vykonávajú v skalárnom procesore. Graficky je celé rozdelnie znázornené na obrázku 2.7.



Obrázok 2.7: Graficky znázornené dvojrozmerné rozdelenie zariadenia pre vykonávania vlákien. Obrázok prevzatý z [14].

Ak chceme dosiahnuť s aplikáciou čo najlepšie výsledky, je potrebné dodržať niekoľko doporučení a podľa toho upraviť kód. Základné doporučenia sú tri.

Maximalizovanie paralelného vykonávania

Ako už bolo spomínané, OpenCL dokáže spracovávať a vykonávať operácie masívne paralelizovane. Pri inicializácií potrebnej pre spustenie OpenCL sa použije výpočtový výkon a nastáva menšie oneskorenie, kým sa začne vykonávanie. Následne je najvhodnejšie, ak zariadenie môže naplno využiť svoju kapacitu paralelným vykonávaním a sekvenčné úlohy, ktoré sa neopakujú veľakrát je väčšinou lepšie vykonať na hosťovi. Pri sekvenčných úlohách sa totiž jednoduché skalárne procesory ťažšie vyrovnávajú s náročnejšími úlohami a môže byť badateľné značné spomalenie aplikácie. Taktiež sa niekedy viac oplatí nezaťažovať všetky dostupné procesory a využitie nad 50% nemusí nutne viesť k lepším výsledkom, ba dokonca za určitých okolností je výhodnejšie ich menšie využitie a tým zvýšiť priepustnosť pamäte [17].

Maximalizovanie pamäťovej priepustnosti

Zrejme najväčším problémom pri OpenCL je operovanie s pamäťou. Odosielanie a prijímanie dát z hosťa do koncového zariadenia je vzhľadom na rýchlosť vykonávania veľmi pomalé. Naopak komunikácia medzi zariadeniami je veľmi rýchla. Napriek tomu sa dá táto nízka priepustnosť obísť pomocou rôznych techník.

V prvom rade je to minimalizovanie prenosu dát napr. používaním preprocesora pre konštanty. Výhodnejší je jeden veľký prenos dát ako viacero menších. Používať čo najmenej pomalú globálnu pamäť a použiť radšej omnoho rýchlejšiu lokálnu alebo pamäť konštánt ak je to možné. Globálna pamäť by sa mala využívať výhradne na nutné vstupy a výstupy, ale vo vnútri pracovnej skupiny pracovať hlavne s lokálnou, ktorá je určená pre skupinu. Prístup do, hlavne globálnej, pamäti by mal byť zarovnaný, inak sa zbytočne alokuje miesto. Nemenej dôležité je takisto obmedzenie prístupu viacerými vláknami do rovnakých baniek, čo už bolo spomínané.

Maximalizovanie inštrukčnej priepustnosti

Inštrukčná priepustnosť je definovaná ako počet inštrukcií za jeden takt. Pre lepšie využitie je samozrejme vhodné použiť menší počet taktov na vykonanie požadovanej úlohy. Medzi jednoduché matematické úkony patrí sčítanie, odčítanie, násobenie a iné, zložité sú zase delenie alebo zvyšok po delení. OpenCL poskytuje pre každý dátový typ určitú presnosť, pokiaľ však ide viac o rýchlosť ako o presnosť, tak OpenCL poskytuje okrem klasických matematických funkcií aj s prefixom *native_*, ktoré poskytujú menšiu presnosť závislú od zariadenia, ale obvykle sú podstatne rýchlejšie. Okrem toho OpenCL kompilátor poskytuje rôzne voľby optimalizácie, taktiež na úkor presnosti [18].

Keďže skupina Khronos stojí za viacerými otvorenými technológiami, medzi ktoré patrí taktiež nízkoúrovňové a obľúbené API pre grafické operácie OpenGL[8], tak prepojenie s ním je na veľmi dobrej úrovni. Je to dosiahnuté hlavne vďaka zdieľaniu výpočtových prostriedkov, zdieľaniu dát, čo znamená spoločnú pamäť jak pre OpenCL, tak aj OpenGL a dobrá synchronizácia na zariadení, prípadne medzi viacerými zariadeniami.

Ohľadom OpenCL nebolo popísaných ešte mnoho vecí, ale pre potreby tejto práce je predstavenie vyššie spomenutých technológií a techník dostačujúce, niektoré bližšie informácie budú viac spomenuté v kapitole 5 o implementácií. V nasledujúcej kapitole bude predstavený úvod do holografie a výpočet hologramu a s ním súvisiace optické pole s využitím OpenCL a GP-GPU.

8 OpenGL (Open Graphics Library) – štandart pre tvorbu aplikácií pomocou akcelerovaných grafických kariet

3 Holografia

V tejto kapitole budú popísané základné princípy a spôsoby, ktoré sa využívajú v holografii. Predovšetkým základy o podstate svetla, keďže od neho závisí celá podstata zobrazovania hologramu. Ďalej to bude interferencia a koherencia ako nutné požiadavky na vytváranie hologramu. Neskôr sa dostaneme k fyzikálnym javom ako napríklad difrakcia a na koniec rôzne zobrazenia hologramu. Bude sa však skutočne jednať len o základ bez hlbšieho popisu. Podrobnejšie informácie sa čitateľ môže dozvedieť napr. z [19] alebo [20].

3.1 Vlastnosti svetla

Kedysi sa vo fyzike viedol spor o tom, či má svetlo časticovú alebo vlnovú povahu. O tom, že sa jedná skôr o vlnenie svedčili aj fakty o interferencii a difrakcii. Ale konečné vyriešenie sporu priniesol až J. C. Maxwell so svojimi rovnicami, ktoré popisujú zákony v elektrickom a magnetickom poli. Pri predpoklade určitých charakteristických vlastností prostredia, potom z elektrického a magnetického poľa vyplýva skalárna rovnica

$$\nabla^2 u(\boldsymbol{p},t) - \frac{n^2}{c^2}\frac{\partial^2 u(\boldsymbol{p},t)}{\partial t^2} = 0 \tag{3.1}$$

kde $u(\boldsymbol{p},t)$ je rušenie na pozícii p v čase t v prostredí s refrakčným indexom n. Rušenie má strednú rýchlosť ako c/n, kde c je rýchlosť svetla a toto rušenie nazývame vlna. Ak vieme, že kruhová frekvencia je ω, tak potom vlnová dĺžka λ bude vyjadrená vzťahom

$$\lambda = \frac{c}{n\omega} \quad \text{prípadne} \quad \lambda = \frac{c}{\omega} \tag{3.2}$$

keď neberieme do úvahy prostredie.
Svetelný zdroj je zdroj elektromagnetického žiarenia. Ak je vlnová dĺžka svetelného zdroja medzi 360 – 800 nm, tak sa jedná o žiarenie, ktoré ľudské oko pozoruje ako viditeľné svetlo. Svetelný zdroj zapíšeme ako

$$A\cos(\omega t - \varphi) \tag{3.3}$$

kde A je amplitúda a φ je počiatočná fáza. Svetlo môžeme chápať aj ako vzájomné pôsobenie elektricky nabitých častíc, ktoré kmitajú. Ak máme v prostredí nejaké dva body vzdialené od seba dĺžkou r a jeden kmitá s určitou amplitúdou A a frekvenciou ω, tak druhý bod bude kmitať s amplitúdou menšou nepriamo úmerne so vzdialenosťou medzi nimi A/r a rozkmitá sa neskôr o čas, ktorý prejde, kým druhý bod zachytí informáciu od prvého. Frekvencia ω však zostáva. Vzťah medzi dvoma bodmi je nasledujúci

$$\frac{A}{r}\cos(\omega[t-\frac{r}{c}]-\varphi)=A'\cos(\omega t-\varphi') \tag{3.4}$$

Pri vytváraní fotografií sa z obrazu získava z každého bodu len jeho intenzita, ktorú vyjadrujeme ako amplitúdu umocnenú dvoma, čiže A^2 . Z intenzity sa však nedá získať informácia o vzdialenosti objektu, pretože sa neberie do úvahy fáza, ktorú svetlo vyžaruje.

3.2 Interferencia a koherencia

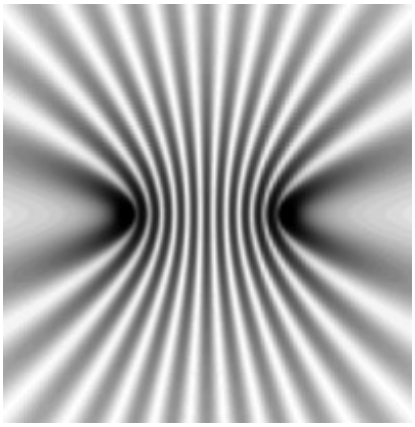
Interferenciou alebo skladaním nazývame vzájomné pôsobenie dvoch alebo viacerých koherentných vĺn z viacerých zdrojových svetiel. Intenzita interferencie je významná pre holografiu, pretože sa využíva počas zaznamenávania hologramu.
Opäť predpokladajme ľubovolné dva svetelné body s rovnicami

$$\frac{A}{r_1}\cos(\omega[t-\frac{r_1}{c}]) \quad \text{a} \quad \frac{A}{r_2}\cos(\omega[t-\frac{r_2}{c}]) \tag{3.5}$$

kde r_1 a r_2 je vzdialenosť dvoch bodov od nejakého stredu. Po úprave vyjmeme *c* ako rýchlosť svetla pomocou vlnového čísla. Vlnové číslo *k* je definované ako počet vlnových dĺžok pripadajúcich na jednotku vzdialenosti alebo ako $k=2\pi/\lambda$ známe pod pojmom uhlové vlnové číslo. Ak predpokladáme, že $r_1\approx r_2$ tak po sčítaní obidvoch bodov získame vzťah

$$2\frac{A}{r_1}\cos(\frac{2\omega t-k(r_1+r_2)}{2})\cos(\frac{k(r_1-r_2)}{2}) \tag{3.6}$$

frekvencia sa v oproti pôvodným hodnotám nezmenila. Hodnota amplitúdy je navyše ovplyvnená interferenčným členom a kvôli tomu sa amplitúda rovná nule ak sa vzdialenosti r_1 a r_2 líšia o $\lambda/2$ alebo $(2\lambda+1)/2$. Na obrázku 3.1 je zobrazený interferenčný jav. Čierna farba reprezentuje amplitúdu vyššiu od nuly a biela farba znázorňuje posun o $(2\lambda+1)/2$ teda tam, kde je amplitúda nulová.



Obrázok 3.1: Zobrazenie interferencie. Obrázok prevzatý z [21].

Koherencia je takisto dôležitá pre záznam hologramu, pretože od nej závisí viditeľnosť vzoru. Pojem koherencia značí schopnosť zväzku svetla interferovať po prejdení určitej vzdialenosti. Podmienkou je zachovanie rovnakej frekvencie tzv. fotochromatickosť, smeru kmitania a fázy alebo fázového rozdielu v danej vzdialenosti. Ak svetlo nie je koherentné, interferenčný obraz nie je dostatočne viditeľný. K ideálnemu fotochromatickému zdroju svetla sa približuje napr. laser alebo maser.

3.3 Vlnoplochy a difrakcia

Ako už bolo spomenuté vyššie, holografia zaznamenáva jak intenzitu, tak i fázu svetla. Fáza predstavuje vlnoplochu, čiže rozloženie zaznamenávaného vzoru v priestore. Z vlny sa preto zaznamenáva úplná komplexná amplitúda, ktorú pomocou Eulerovej identity a exponenciálu zapíšeme v tvare

$$\frac{A}{r}\exp(-j[kr+\varphi]) \tag{3.7}$$

kde amplitúda klesá nepriamo úmerne so vzdialenosťou od zdroja svetla a *kr* posúva fázu vzhľadom k vzdialenosti, kde sa bod nachádza. Klasické vlnoplochy sú dva typy a to jednoduchá rovinná a guľová. Guľovú vlnoplochu zapíšeme z komplexnej amplitúdy nasledovne

$$\frac{A}{r}\exp(j[\omega t-k.r-\varphi]) \tag{3.8}$$

Z výrazu je zrejmé, že amplitúda sa od zdroja zmenšuje, ale fáza zostáva stále rovnaká. Guľová vlnoplocha má totiž takú vlastnosť, že s pribúdajúcou vzdialenosťou viac pripomína rovinnú vlnoplochu. V rovinnej vlnoploche sú roviny konštantnej fázy kolmé na smer šírenia svetla. Tento vzťah vyjadríme ako

$$A\exp(-j[\boldsymbol{k}.\boldsymbol{x}+\varphi]) \tag{3.9}$$

kde $\boldsymbol{k}$ je vlnový vektor získaný z vlnového čísla k vynásobeného o $\boldsymbol{n}$, aby boli roviny medzi sebou rovnako vzdialené o vlnovú dĺžku, takže $\boldsymbol{k} = k.\boldsymbol{n}$ a $\boldsymbol{x}$ je bod v priestore so súradnicami $\boldsymbol{x} = (x, y, z)$. Takto vyjadrená rovinná vlnoplocha má aj pri zväčšujúcej sa vzdialenosti stále rovnakú amplitúdu a naopak fáza postupne rastie.

Difrakcia je ďalší významný fyzikálny jav, bez ktorého by sme sa v holografii neobišli. Pomocou difrakcie sa hologram vytvára a zároveň aj reprodukuje. Difrakcia alebo ohyb svetla je ohyb svetelných vĺn pri prechode malým otvorom alebo na okraji prekážky. Umožňuje určiť svetelné pole pri svojom šírení a vytvoriť optické pole, teda hologram. Pri difrakcii v holografii nás hlavne zaujíma, kedy, teda pod akým uhlom, bude viditeľná nenulová amplitúda. Ak bude počet otvorov v prekážke m, tak hovoríme m-tom difrakčnom maxime a vyjadruje sa ako

$$(\varphi_m - \varphi_{m-1} - \ldots - \varphi_2 - \varphi_1) + k.d.sin\varphi_p = m.2\pi \tag{3.10}$$

kde φ_m je m-tá veľkosť uhla dopadajúceho na prekážku, k je vlnové číslo, d je vzdialenosť medzi otvormi v prekážke a φ_p je uhol, pod ktorým sa pozeráme na prekážku. Táto rovnica sa dá zobecniť vzniknutý vzťah sa nazýva difrakčná rovnica, ktorá vyzerá nasledovne

$$sin\varphi_p = \frac{m.\lambda}{d} + sin\varphi_v \tag{3.11}$$

a hovorí nám, že ak svietime na otvory v prekážke, ktoré sú od vzdialenosti d pod uhlom φ_v tak m-té difrakčné maximum bude pozorovateľné pod nejakým výstupným uhlom φ_p .

3.4 Hologram

Holografia má za svoj cieľ vytvárať hologramy. Hologram je záznam optického poľa v danej rovine, ktorý môže byť reprodukovaný. Záznam hologramu sa dá popísať nasledovne. Máme svetelný zdroj, ktorý svieti objektovou vlnou na rovinu svetlocitlivého film pod určitým uhlom. Zároveň však svieti iné, referenčné svetlo na rovinu filmu. Tieto svetlá spolu interferujú a pri dopade na film zmenia na povrchu jeho štruktúru, na ktorej sa zachytí obraz. Po vyvolaní filmu a nasvietení referenčným svetlom pod rovnakým uhlom sa zrekonštruuje objektová vlna pod uhlom, ktorým dopadala na rovinu filmu. Za rovinou filmu sa teda budú zobrazovať rôzne kúsky vlnoplochy, ktoré budú vyzerať presne tak, ako keby pre rovinu svietilo objektové svetlo s pôvodným uhlom. Vznikne tak virtuálny obraz bodového zdroja. Po zmenení difrakčného maxima na zápornú hodnotu sa vytvorí obraz za rovinou filmu s opačnými uhlami. Takýto obraz voláme reálny obraz bodového zdroja. Zatiaľ čo virtuálny obraz vidíme ako normálny objekt, reálny obraz vidíme invertný a dá sa preto použiť ako zobrazenie na nejakú rovinu. Hologramy je možné zaznamenávať mnohými spôsobmi, avšak popísané budú len 3 základné. Viac informácií je uvedených v publikácii [22].

3.4.1 In-line hologram

Je to pôvodný a zároveň najľahší spôsob D. Gabora. Táto metóda je však použiteľná v podstate len s priehľadnými objektami. Funguje tak, že objekt nastavíme do cesty laserovému lúču, ktorý prejde cez objekt. Keďže je objekt rôzne tvarovaný, tak bude nastávať k difrakcii a po interferencii sa dopadajúce lúče zaznamenajú na film. Pri rekonštrukcii však dochádza k nepríjemnej vlastnosti a to k prekrytiu nultým difrakčným radom z priameho svetla laseru.

3.4.2 Off-axis hologram

Keďže in-line hologram nebol vyhovujúci pre bežné používanie kvôli nultému difrakčnému radu, bolo potrebné vymyslieť spôsob, aby sa obraz a nultý difrakčný rad neboli prekryté a oddelené od seba nejakým uhlom. Off-axis má túto vlastnosť a okrem toho pracuje aj s nepriehľadnými objektami. Postup záznamu je sa deje tak, že svetlo z laseru sa pustí na polopriepustné zrkadlo. Časť svetla dopadne na zrkadlo, z ktorého sa odráža lúč na rovinu filmu. Tento lúč je referenčná vlna. Druhá časť

svetla prejde ďalej, dopadá na objekt a odtiaľ sa náhodne odráža na rovinu filmu ako objektová vlna. Po interferencii a dopade sa zmení film. Po vyvolaní filmu a odobratí objektu zo scény dopadá na film iba referenčné, v tomto prípade zvané rekonštrukčné svetlo. Za filmom následne vznikne dokonalý obraz objektu. Nevýhodou ale je, že lúče dopadajú na film pod vysokým uhlom a tým pádom je výsledná frekvencia vysoká a vyžaduje veľmi jemnú technológiu na zápis. Ďalšia nevýhoda je, že off-axis hologram je náchylný k aberáciam, teda nepresnostiam. Dochádza k nej, ak na rekonštrukciu použijeme inú ako referenčnú vlnu, teda nebude sa rovnať referenčný uhol s rekonštrukčným alebo referenčná vlnová dĺžka s rekonštrukčnou. Dostaneme síce obraz, ale bude rôzne deformovaný, napr. uhlovo posunutý, roztiahnutý do šírky, do dĺžky, do hĺbky atď. Aberáciam sa však dá predísť a to viacerými spôsobmi.

Prvou metódou je tzv. hologram hologramu. Najskôr klasicky vytvoríme hologram, potom tento hologram zrekonštruujeme svetlom dopadajúcim pod opačným uhlom ako je referenčný. Za rovinu tohto hologramu dáme nový film, ktorý osvietime referenčnou vlnou. Na film dopadá objektová vlna z prvého hologramu, kde už je zaznamenaný vzorový objekt. Tým pádom vznikne po nasvietení pôvodnou referenčnou vlnou na novom filme obraz objektu. Tento objekt ale bude v strede hologramu.

Inou metódou je tzv. dúhová metóda. Prvé kroky sú rovnaké ako pri prvej metóde, ale keď začneme robiť hologram hologramu, tak za prvý hologram dáme prekážku s malou štrbinou. Tým vznikne na druhom holograme neúplná informácia. Ak sa pri rekonštrukcii druhého hologramu sa nepoužije rovnaké svetlo ako pri vytváraní, tak obraz bude uhlovo posunutý. Keď sa ale použije na rekonštrukciu svetlo s viacerými vlnovými dĺžkami, tak uvidíme síce jeden obraz, ale v každom uhle pohľadu bude mať obraz inú farbu.

3.4.3 Fourierov hologram

Posledným zo základných spôsobov zaznamenávania hologramu je Fourierov hologram. Na rozdiel od predchádzajúcich metód sa Fourierov hologram vytvára pomocou objektívu. Objekt musí byť ale priehľadný ako v prípade in-line hologramu. Kvôli tomu sa používa len na výpočtové simulácie, keď táto vlastnosť nevadí.

4 Holografia pomocou počítačov

Táto kapitola pojednáva, o základných metódach tvorenia hologramu na počítači, ďalej sa venuje urýchleniu týchto metód. Ako posledná vec je návrh na implementáciu vhodnej metódy zobrazovania, ktorú je možné použiť na akceleráciu pomocou OpenCL. Holografia, ktorá je spracovávaná pomocou počítačov sa taktiež nazýva digitálna holografia. Digitálna holografia je podobná optickej, ale pracuje sa s diskrétnymi číslami, používa však rovnaké princípy a metódy. Keďže sú počítače schopné simulovať číselné výpočty, sú schopné spolupracovať s optickou holografiou a to tak, že spracujú údaje z fyzikálnych experimentov. Z týchto údajov potom počítače majú na starosti tvorbu hologramu, rekonštrukciu hologramu, kompresiu hologramu, získanie potrebných informácii a reprodukcia hologramu. Najnáročnejší proces je pre počítače práve reprodukcia. V súčastnosti už existujú riešenia na zobrazenie hologramu. Tieto zariadenia sú však často pomalé, môžu zobraziť len malé obrazy objektov a okrem toho nie sú vylúčené chyby v zobrazení. Najväčším obmedzením je však zmienená veľkosť a detailnosť obrazov. Zo zväčšením hologramu alebo optického pola sa zvyšujú nároky jak na výpočtový výkon, tak aj na veľký objem spracovávaných dát. Preto sa hľadajú nové spôsoby ako urýchliť výpočet a znížiť objemovú náročnosť. Informácie v tejto kapitole budú prevažne z [23].

4.1 Tvorba hologramu

Tvorba hologramu nemá len jednu metódu, je ich viac a v nasledujúcich bodoch budú popísané niektoré z nich. Nejedná sa o zrýchlenie niektorej metódy, ale len základná tvorba.

4.1.1 Metóda založená na geometrii

Základná vlastnosť tejto metódy je použitie svetelného lúča na výpočet príspevku do optického pola. Počítač prechádza každý rez objektu podľa zvoleného kroku a zisťuje, kde sa nachádza svetelný bod. Po vypočítaní všetkých rezov sa medzivýsledky spočítajú a dostaneme výsledné údaje. Viditeľnosť je riešená pomocou metódy ray-casting. Aj keď ray-casting ignoruje difikraciu na prekážkach, napriek tomu vytvára hologram, ktorý je možno vidieť.

Metóda založená na geometrii používa jednoduché výpočty a vďaka tomu je možné ich urýchliť pomocou hardvéru a tak je možné dostať sa na lineárnu zložitosť. Táto metóda má ale jednu hlavnú nevýhodu, v podobe priveľkého množstva výpočtov, ktoré sa ani nedajú vhodne paralelizovať, pretože si to vyžaduje veľmi veľké pamäťové nároky. Ako už bolo spomenuté, prechádza sa každý rez a v pamäti zostáva až do konca, kým sa výsledky nesčítajú. Na bežných zariadeniach sa preto nedá dostatočne zrýchliť.

4.1.2 Metóda založená na vlnách

Táto metóda je založená na šírení vlny, konkrétnejšie na vytvorení uhlového spektra. Výhodou je dobrá rýchlosť spracovania. Značnou nevýhodou ale je, že šírenie uhlového spektra pracuje s frekvenčnou oblasťou, zatiaľ čo viditeľnosť pracuje v priestorovej oblasti. Tieto zmeny oblastí však zvyšujú zložitosť.

Na prevod optického pola do uhlového spektra sa používa FFT[9]. Pri FFT musíme vyriešiť ďalší problém a tým je periodicita. Aby sme sa jej vyhli, musíme optické pole dvojnásobne zväčšiť na každej strane, teda obsah bude štvornásobný. Potom ale vyberieme len časť pôvodného optického pola. S tým súvisí aj problém pri rotácii optického pola. Obraz spôsobuje nelineárnu deformáciu spektra, tým sa zhoršuje signál a dochádza k zvýšeniu šumu. Metóda založená na vlnách má zložitejšie výpočty, ale vďaka tomu, že uhlové spektrum sa dá veľmi dobre paralelizovať je pri vhodnej implementácii rýchla. Môžeme ju ale použiť iba za predpokladu, že nepotrebujeme riešiť viditeľnosť.

Medzi ďalšie metódy patrí tzv. pohľadová metóda, ktorá sa ale veľmi nepoužíva. Slúži na vytváranie Fourierovho hologramu a preto je špecifická.

4.2 Akcelerácia hologramu

V rámci digitálnych hologramov už boli spomenuté typy a spôsoby ich vytvárania. Nemenej dôležitá časť je taktiež akcelerácia, alebo zrýchlenie či už pri vytváraní alebo reprodukcii. Preto sú v nasledujúcej podkapitole popísané možnosti zrýchlenia.

4.2.1 Aproximácie

Aproximáciou, alebo inak povedané nahradením približného čísla, je možné skrátiť dobu, potrebnú na výpočet optického pola. Medzi časovo najzložitejšie operácie patrí odmocnina, pomocou ktorej získavame vzdialenosť. Odmocnina sa však dá aproximovať z prvých dvoch jej členov. Zrýchlenie teda spočíva v znížení počtu príspevkov do optického pola. Výsledný obraz ale nemusí byť dobre viditeľný zo všetkých uhlov. Okrem toho je síce digitálna rekonštrukcia jednoduchá, optická rekonštrukcia je zložitejšia a vyžaduje špeciálne zariadenia.

4.2.2 Zjednodušenie scény

Medzi ďalšie možnosti akcelerácie patrí zjednodušenie scény. Táto technika sa ešte rozdeľuje na ďalšie podskupiny. V každej ale ide o princíp rozdelenia objektu na menšie časti, ktoré by boli výpočtovo jednoduchšie. Bližší popis je nasledovný.

Zhlukovanie svetelných bodových zdrojov.
S narastajúcim počtom svetelných bodových zdrojov rastie aj zložitosť výpočtu. Napriek tomu sa dá táto vlastnosť využiť na zrýchlenie. A to z dôvodu, že optické pole vytvorené z mnohých svetelných bodových zdrojov je jednoduchšie. Možností na akceleráciu je viacero, napr. sa používajú tabuľky s predpočítanými hodnotami alebo pomocou diskrétnej Rayleigh-Sommerfeld-ovej rovnice.

9 FFT (Fast Fourier Transform) – algoritmus pre vypočítanie diskrétnej Fourierovej transformácie slúžiaci na digitálne spracovanie signálov, riešenie parciálnych diferenciálnych rovníc a iné výpočty.

Zhlukovanie čiar.
Akcelerácia v tomto prípade spolieha na výpočet viacerých svetelných bodových zdrojov naraz, ktoré sú v jednej línii. Jeden riadok teda nahradí hneď niekoľko svetelných bodových zdrojov a optické pole sa dá potom aproximovať. Táto technika je oproti Zhlukovaniu svetelných bodových zdrojov rýchlejšia, keďže sa nahradí veľké množstvo bodov jedným riadkom. Taktiež ale nastáva problém s viditeľnosťou.

Zhlukovanie trojuholníkov. Podobne ako v predchádzajúcej technike, aj v tejto sa počíta s viacerými svetelnými bodmi, ale v trojuholníkoch. Celý model sa teda rozdelí na trojuholníky, s ktorými sa ďalej pracuje. Výpočet však nie je vyjadrený žiadnou rovnicou, ale počíta sa z predpočítaných tabuliek optického pola. Tento spôsob je najzložitejší a nie je ani veľmi efektívny, pretože počet a umiestnenie je limitujúce.

4.2.3 Zjednodušenie optického pola

Na akceleráciu pomocou zjednodušenia optického pola sa používajú pospájané klasické funkcie. Ako vstup je zhluk svetelných bodových zdrojov, z ktorých dostaneme hologram pozostávajúci z nezávislých difrakčných prvkov. Tieto prvky nazývame hogely, čiže jednorozmerné hologramy.

5 Návrh a optimalizácia

Piata kapitola sa najskôr venuje návrhu algoritmu pre výpočet optického pola vo všeobecnosti, a ktorý je implementovaný v nasledujúcej kapitole. Ďalej bude nasledovať optimalizácia popísaného algoritmu, aby sa tak čo najviac využíval výkon a samotný výpočet tak skrátil dobu vykonávania. Hlavne budú predstavené jednoduché úpravy, ktoré k skráteniu času však výrazne pomôžu. Určite by sa dali aplikovať aj ďalšie, menej jednoduché optimalizácie, táto práca sa im ale nebude venovať. K čomu optické pole slúži už bolo spomenuté avšak bez bližších podrobností a názorných ukážok. Nasledujúce podkapitoly a kapitoly budú obsahovať informácie, ktoré boli predstavené v predchádzajúcich kapitolách, preto sa nebudú podrobne vysvetľovať.

5.1 Návrh algoritmu

Správny návrh algoritmu je dôležitý pre ďalší vývoj aplikácie a ušetrí mnoho času pri programovaní, keďže sú známe detaily, na ktoré si treba dať pozor a na čo sa nesmie zabudnúť. Preto teraz bude predstavené, ako budem postupovať ďalej.

Pre naše účely môžeme predpokladať nasledujúcu scénu. Objekt v priestore, ktorý chceme zrekonštruovať ako hologram je napríklad kocka. Na kocku svieti koherentný svetelný zdroj so známou presnou polohou v priestore, vlnovou dĺžkou, amplitúdou a fázou. Toto svetlo sa rozdelí na dve časti pomocou polopriepustného zrkadla tak, že jedna časť smeruje na kocku a druhá časť je referenčné svetlo smerujúce na optické pole, na ktoré dopadá pod určitým uhlom. Po dopade svetla na kocku sa časť svetla dopadne priamo na optické pole, prípadne z odrazu, časť sa odrazí mimo do priestoru, na hranách kocky však nastáva difrakcia, čiže lom svetla. Po odraze alebo difrakcii svetla nastáva zmena amplitúdy a zmení sa taktiež fáza dopadajúceho svetla. Následne pred dopadom na optické pole nastáva interferencia referenčného svetla a svetla od kocky.
Takto vzniká optické pole z optickej holografie. Digitálny spôsob je tomuto veľmi podobný. Optické pole je v tomto prípade dvojrozmerné pole rozdelené na pixely, ktoré majú určitú veľkosť a rovnakú veľkosť majú aj pixely, z ktorých je zostavený objekt, teda kocka. Každý pixel kocky je potom svetelným bodom, ktorý vyžaruje svetlo s určitou amplitúdou a fázou.

Medzi najjednoduchšie postupy ako získať optické pole z bodového zdroja je vypočítanie každého jedného svetelného bodu kocky umiestnenej v priestore a jeho následné pripočítanie do celého pola. Príspevok jedného bodu do pola sa vypočíta pomocou niekoľkých krokov. Prvým je zistenie vzdialenosti bodu od pola. Vzdialenosť l zistíme zo vzťahu

$$l=\sqrt{((x_A-x_p)^2+(y_A-y_p)^2+(z_A-z_p)^2)} \tag{5.1}$$

kde x_A, y_A, z_A sú súradnice bodu v priestore a x_p, y_p, z_p sú súradnice jedného pixelu v optickom poli. Vlnovú dĺžku λ, fázu φ a amplitúdu A svetelného zdroja poznáme, takže vzorec môžeme zapísať ako

$$A\exp(j(\frac{2\Pi}{\lambda}l)) \tag{5.2}$$

Ako vidno, vo vzorci vystupuje Eulerovo číslo a teda sa jedná o komplexné číslo, čo sa veľmi nehodí. Obvykle je nutné v programovacích jazykoch použiť dodatočné knižnice, ktoré vedia operovať s komplexnými číslami, inak je obtiažne s nimi riešiť aj základné matematické operácie ako sčítanie, odčítanie, násobenie alebo delenie. Pre jednoduchšie zaobchádzanie s komplexnými číslami môžeme použiť Eulerov vzorec, ktorý má tvar

$$\exp(jx)=\cos(x)+j\sin(x) \tag{5.3}$$

Podľa Eulerovho vzorca používa ako reálnu časť $\cos(x)$ a imaginárnu ako $j\sin(x)$. Vzorec 5.2 tak môžeme zapísať nasledovne

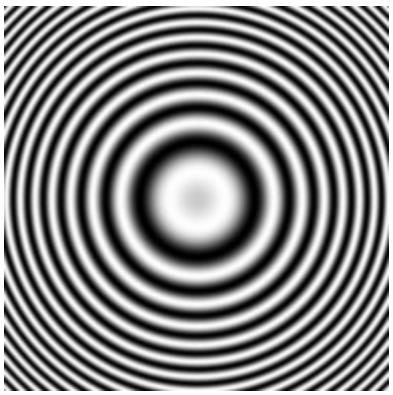
$$A\cos(\frac{2\Pi}{\lambda}l)+A\sin(\frac{2\Pi}{\lambda}l) \tag{5.4}$$

Po tejto úprave sa dá s komplexnými číslami ľahšie operovať, pričom nie sú potrebné doplňujúce knižnice, ktoré by mohli mať negatívny vplyv na rýchlosť aplikácie, ale stačí dátový typ podobný štruktúre.
Vzorec 5.4 je potrebné použiť pre každý bod pola a pre každý bod kocky. Z toho vyplýva, že výpočet sa bude opakovať X * Y * P, kde X je jedna dimenzia pola, Y je druhá dimenzia a P je počet pixelov kocky, resp. objektu. Keď zoberieme do úvahy, že pre dobre rozoznateľný objekt musí byť veľkosť pixela rádovo v miliónтinách (10^{-6}) metra, prípadne menšie, tak aj pri veľmi malých a jednoduchých, milimetrových objektoch sa dostaneme k číslam z opačnej strany škály a to rádovo k miliardám (10^{9}) až biliónom (10^{12}) výpočtov.

Príklad 5.1: veľkosť jedného pixelu je 2.10^{-6} a pole má na X-ovej aj Y-ovej osi 8 192 pixelov, čo predstavuje 16,384 mm. Vo výške 12 mm nad plochou sa nachádza objekt, ktorý pozostáva z hrán kocky. Hrana má 2 000 pixelov, teda 4 mm. Po vynásobení rozmerov plochy a počet bodov objektu, vyjde nám počet (8192 * 8192 * 24 000) = 1 610 612 736 000 nie najjednoduchších výpočtov.

Príklad 5.1 jasne ukazuje, že aj objekty, ktoré sú pre ľudské oko len ťažko viditeľné kvôli veľkosti vyžadujú veľký výpočtový výkon. Zároveň je zrejmé, že uvedený postup nie je vôbec optimálny a aj pri použití paralelného spracovania nedosiahneme uspokojivé výsledky v rýchlosti výpočtu.

Každý bod objektu vyžaruje svetlo, ktoré sa šíri všetkými smermi a má vlnový priebeh. Svetlo z bodu teda dopadá na celé pole a to pod určitým uhlom, ktorý je malý, rádovo v jednotkách stupňov. V mieste dopadu je svetlo v aktuálnej časti periódy a podľa toho, akú má v tomto mieste daná funkcia (sínus alebo kosínus) veľkosť sa pripočíta príspevok do optického pola. Keďže svetlo sa šíri guľovito, na rovinnej ploche sú rovnaké hodnoty umiestnené do kruhu. Názorná ukážka je na obrázku 5.1.

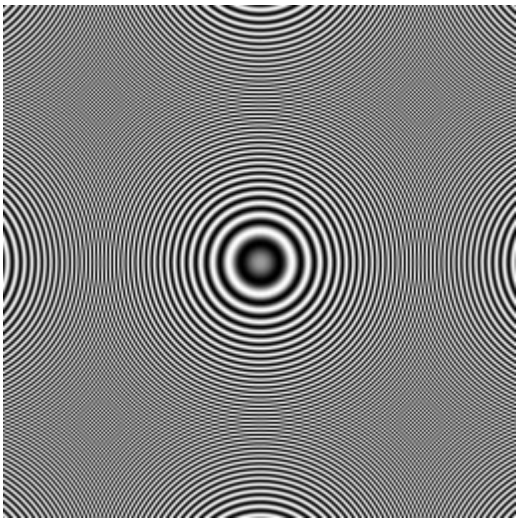


Obrázok 5.1: Reálna zložka jedného bodu vzdialeného 8 mm od optického pola. Veľkosti bodu 2μm.

V strede obrázka sú kružnice hrubšie, ale s väčšou vzdialenosťou sa zužujú až dosiahnu bod, keď už sa nebudú zužovať, ale začnú sa kruhy zrkadlovo opakovať. Tento jav vzniká, keď sa prekročí hraničný uhol, pri ktorom sa začne hodnota periódy svetla akoby vracať. Z tenších kruhov tak začnú byť hrubšie až sa opäť začnú zužovať a takto periodicky ďalej. Ak by sa použilo optické pole, kde sa nachádzajú tieto periodické kruhy, tak vo výslednom holograme by bolo vidno vzorové objekty viackrát posunuté v osových súradniciach. Preto je veľmi vhodné vyvarovať sa tomuto neduhu. Uhol, ktorý je ešte prípustný a pri ktorom nebude vidno opakovanie sa vypočíta približne podľa vzťahu 5.5.

$$\alpha = \arcsin\left(\frac{\lambda}{p}\right) \tag{5.5}$$

Pričom λ je vlnová dĺžka referenčného svetla a p je veľkosť bodu. Veľkosť uhla α je celková, nie od kolmice plochy na bod, čiže polovičná.
Ako vyzerá periodicita kruhov je na obrázku 5.2.



Obrázok 5.2: Opakovanie reálnej zložky z obrázka 5.1 pri vzdialenosti 2 mm od optického pola. Veľkosti bodu 2μm.

Medzi uhlom a vzdialenosťou je, dá sa povedať, nepriama úmera, čím je bod od plochy viac vzdialený, tým menšiu časť ako keby vidno. Dokumentujú to aj obrázky 5.1 a 5.2, kde sú tie isté pohľady, akurát vzdialenosť na obrázku 5.1 je štvornásobne väčšia. Opakovanie kružníc teda nastáva, ak je bod vzhľadom k ploche blízko. Odstrániť tento nežiadúci efekt nie je veľmi zložité, stačí vypočítať uhol podľa vzorca 5.5 a pomocou tohto uhla a ďalej rátať len s kruhom, ktorý sa vytvoril vďaka otočeniu uhla na kolmicu smerujúcu od rovinnej plochy. V tejto práci sa však s týmto riešením nebude počítať a budú brané do úvahy aj opakujúce sa kruhy.

5.2 Optimalizácia algoritmu

Keď vieme, ako sa premietne jeden svetelný bod v priestore, tak môžeme využiť princíp optického pola. Každý jeden svetelný bod pridá do optického pola svoj prírastok a teda to, čo vznikne na obrázku 5.1 alebo 5.2. Tento prírastok bude pre však buď posunutý o súradnice, kde sa nachádza alebo ešte aj rozptyl bude rôzny vzhľadom na inú výšku. Toto sa dá využiť pre zmenšenie množstva výpočtov a to takým spôsobom, že sa vypočíta len jedno vzorové pole pre každú výšku objektu. Vo vzorovom poli bude premietnutý bod umiestnený v strede pola a následne sa posúvať do súradníc, kde sa nachádzajú svetelné body objektu v tej danej výške. Posun je oveľa rýchlejšia operácia, ako keby sa mal každý bod zvlášť vypočítať. Výkon sa ušetrí aj vďaka tomu, že sa nemusí „presúvať“ celé pole, pretože zrejme len málo bodov objektu bude presne v strede pola a väčšina bude rozmiestnená mimo stredu. Pri najlepšej možnej situácií, keď sa bod nachádza na hrane pola, stačí pričítať štvrtinu pola a ušetrí sa tak až 75% posunu. Nemožno zabudnúť ani na to, že pri odstránení periodicity sa zmenší veľkosť pola, takže zníženie výkonovej náročnosti bude poznateľné.

Aj napriek všetkým výhodám má toto riešenie menšiu nevýhodu. Ak je v súbore s objektom také poradie, že Z-ová súradnica sa stále alebo často mení, tak sa musí najskôr vypočítať vzorové pole, ďalej ak riešime periodicitu, tak aj orezať nevhodnú časť a nakoniec sa ešte musí posunúť pole podľa súradníc svetelného bodu. Oproti tomu keby sa rátalo priamo, stačí vypočítať jedno pole bez ďalších náročných operácií. Najideálnejší stav by teda bol, keby boli jednotlivé súradnice svetelných bodov usporiadané podľa Z-ovej súradnice. V prípade vlastného usporiadania ešte pred vykonávaním výpočtu optického pola je však opäť nutné použiť výkon, ktorý mohol byť použitý na počítanie pola. V tejto práci sa však bude používať navrhované riešenie, keďže je šanca na ušetrenie výkonu väčšia ako pri priamom počítaní pola.

Pri pohľade na výrazy 5.1 a 5.3, ktorými sa počíta optické pole je možno všimnúť si niekoľko, pre procesor, náročných operácií. Konkrétne sa jedná o mocninu, odmocninu a goniometrické funkcie sínus a kosínus. Vo výraze 5.1 je mocnina len druhého stupňa, čo sa dá jednoducho obísť pomocou vynásobenia premennej s tou istou premennou, čo je jednoduchá operácia. Na bežných procesoroch to ale nie je nutné, pretože obvykle obsahujú registre, ktoré mocninu dokážu veľmi rýchlo vypočítať. Podobné je to aj s odmocninou, súčasné procesory zvládajú túto úlohu veľmi rýchlo. Nemusí tomu tak byť pri použití jednoduchších, napr. skalárnych procesoroch, ktoré sú základnými jednotkami na grafických kartách. Pre prípad, že zabudovaná funkcia odmocniny je pomalá, sa dá použiť menej presná Newtonová iteračná metóda [24]. Pomocou tejto metódy je možné aproximovať odmocninu nielen druhého rádu, ale aj vyšších rádov. Ako už názov napovedá, ide o opakovanie, resp. rekurzia jednoduchého vzorca, ktorý má zápis

$$x_{k+1} = \frac{1}{2}(x_k + \frac{a}{x_k}) \tag{5.6}$$

kde a je číslo, ktoré chceme odmocniť a x_k je predchádzajúca vypočítaná hodnota, pričom x_0 môže byť ľubovolné. Obvykle sa zadáva x_0 ako 1, ale pri veľkých číslach je vhodnejšie zadať väčšie číslo a zmenší sa tak počet iterácií, ktoré budú viesť k správnejšiemu výsledku. Celkový počet iterácií sa musí odhadnúť a zavisí od veľkosti čísla a požadovanej presnosti.
Poslednými náročnými funkciami je sínus a kosínus. Goniometrické funkcie sú periodické a tak dávajú priestor pre jednoduchú, ale účinnú optimalizáciu.

Jak sínus, tak kosínus má jednu periódu o vzdialenosti 2Π, ktorá sa neustále opakuje a tým pádom sa opakuje aj jej hodnota. Teda keď máme číslo v radiánoch, ktoré je väčšie ako jedna perióda, tak môžeme toto číslo vydeliť hodnotou 2Π a dostaneme sa tak do rozsahu $\langle 0, 2\Pi \rangle$. To už je dostatočne malý rozsah na to, aby ho bolo možné dobre využiť. Uvedený rozsah sa rozdelí na určitý počet častí, ktorým sa priradí konkrétna hodnota podľa funkcie. Počet, na ktorý sa funkcia rozdelí závisí od požadovanej presnosti a samozrejme platí, že čím väčší počet rozdelených úsekov, tým väčšia presnosť. Pre ďalšie použitie je však vhodné, aby bol počet úsekov $2^n - 1$, kde *n >= 1*, pretože hodnoty každého jedného úseku sa zapíšu do jednorozmerného pola a pristupuje sa k nemu tak, že veľké číslo sa „oreže“ pomocou funkcie bitového súčinu a získa sa tak číslo z rozsahu. Napríklad pri rozdelení na 1 024 častí bude toľko isto aj v poli a číslo, ktoré je mimo tento rozsah sa získa binárnym súčinom s číslom 1 023. Pri desatinných číslach sa úplne zanedbá časť za čiarkou, čo pri goniometrických funkciách zmení hodnotu, ale pre výpočet optického pola to nemá až taký vplyv na výsledok. Pre ešte väčšiu optimalizáciu je najvhodnejšie dopredu pripraviť tabuľku s hodnotami a negenerovať ju až za behu programu, prístup tak bude o niečo rýchlejší.

Optimalizáciou pre goniometrické funkcie končí 4. kapitola. Poznatky, ktoré boli predstavené vyššie budú použité pri tvorbe aplikácie a venuje sa jej 5. kapitola. Ako bolo spomenuté v úvode, určite existujú ďalšie možnosti pre zrýchlenie, ale táto práca sa im ďalej nevenuje.

6 Implementácia

V tejto kapitole budú aplikované informácie získané z predchádzajúcich kapitol. Konkrétne 2., ktorá pojednáva o frameworku OpenCL, v 3. časti je teória ohľadne holografie a 5. kde je predstavený návrh s optimalizáciou.

Aplikácia je implementovaná v jazyku C, konkrétne štandard C99 spoločne s frameworkom OpenCL. Štandard C99 bol vybratý hlavne z dvoch dôvodov. Prvý je ten, že sa jedná o nízkoúrovňový jazyk, ktorý je vhodný na aplikácie, kde je potrebná veľká rýchlosť a druhý dôvod bol ten, že programovací jazyk pre jadro OpenCL vychádza zo štandardu C99, ktorého názov je OpenCL C. Tvorba aplikácie v OpenCL je vo verzii špecifikácie 1.0. Primárny vývoj bol na linuxovej distribúcií Debian s kompilátor GCC. Vďaka použitiu štandardu C99 by však mala byť aplikácia bez problémov skompilovateľná aj s inými kompilátormi a na iných operačných systémoch. Aplikácia bola kompilovaná aj na operačnom systéme Microsoft Windows XP s prekladačmi v Microsoft Visual Studio C++ 2008 Professional a taktiež prekladačom v MinGW, ktorý vychádza z GCC. Na odladovanie celej aplikácie bol v Debian-e použitý nástroj GDB a na odladovanie OpenCL kódu boli použité nástroje NVIDIA OpenCL Visual Profiler a gDEBugger. Vo Windows XP bol použitý debugger z Visual Studio.
Aplikácia je konzolová bez doplňujúceho grafického rozhrania.

6.1 Vstup aplikácie

Vstupom aplikácie pre spracovanie a vytvorenie optického pola je súbor s trojrozmerným objektom a konfiguračnými možnosťami. Formát súboru nie je prevzatý z nijakých už existujúcich, ale vytvoril som svoj podľa vlastný potrieb. Na začiatku súboru sa musia nachádzať konfiguračné údaje, ktoré majú formát:

- KĽÚČ: HODNOTA

kde kľúče majú nasledovné názvy:

- X – veľkosť X-ovej strany optického pola, hodnota musí byť deliteľná 8
- Y – veľkosť Y-ovej strany optického pola, hodnota musí byť deliteľná 8
- Lambda – vlnová dĺžka zdrojového svetla
- Pixel – fyzická veľkosť jedného svetelného bodu v priestore udaná v metroch
- SourceX – súradnica X-ovej osi zdrojového svetla
- SourceY – súradnica Y-ovej osi zdrojového svetla
- SourceZ – súradnica Z-ovej osi zdrojového svetla
- SourcePhase – fáza zdrojového svetla

Každému kľúčovému názvu musí byť priradená odpovedajúca hodnota, pričom kľúčový názov môže byť napísaný jak malými, tak veľkými písmenami. Oddeľujúci znak je buď medzera, prípadne viac

medzier alebo znak „:“, za ktorým môže tiež nasledovať medzera/medzery. Názov s hodnotou musia byť na jednom riadku spolu a nemôže byť týchto dvojíc viac na riadku.
Pre každý bod modelu je takýto formát:

- X; Y; Z; AMPLITÚDA; FÁZA

A má nasledovný význam:

- X – súradnica X-ovej osi bodu modelu
- Y – súradnica X-ovej osi bodu modelu
- Z – súradnica X-ovej osi bodu modelu
- AMPLITÚDA – amplitúda jedného svetelného bodu modelu
- FÁZA – fáza jedného svetelného bodu modelu

Medzi jednotlivými hodnotami je možné mať viac medzier, ale musí tam byť aj oddeľujúci znak „;“. V súbore môžu byť aj prázdne riadky, ale v konfiguračnej časti ich môže byť len 7, inak aplikáciu skončí chybou.
Pre lepšiu predstavu sa nachádza v prílohe B krátky úsek možného vstupného súboru.

6.2 Inicializácia OpenCL

Pod hostiteľskou časťou aplikácie sa v tomto prípade rozumie kód napísaný v jazyku C, ktorý posiela dáta do zariadenia a takisto ich spracované aj prijíma. V tejto podkapitole budú prezentované funkcie pre inicializáciu výpočtov pomocou OpenCL, aj keď hlavný návrh už bol predstavený v kapitole 5.1.
Aplikácia hneď po spustení kontroluje správnosť konfiguračných informácií a tie sa zapíšu do premenných, ktoré neskôr budú využité na výpočet. V prípade zlého formátu sa ukončí. Užívateľ si môže pri spustení vybrať, či má aplikácia počítať optické pole pomocou OpenCL alebo len klasicky. Ak je vybraté OpenCL, tak sa začne inicializácia.
Najskôr sa pomocou príkazu `clGetPlatformIDs()` zistí počet dostupných platforiem, alokuje sa príslušná pamäť a potom sa vytvorí tým istým príkazom, len s inými parametrami zoznam platforiem. Podobne sa vytvorí aj počet a zoznam zariadení, akurát s volaním `clGetDeviceIDs()`. Informácie poskytujú názov, výrobcu a iné. Pri tomto príkaze je dostupná voľba zariadenia. Na výber sú možnosti vybrať len z CPU, GPU, dedikovaný akcelerátor, implicitné zariadenie alebo zo všetkých dostupných zariadení. Táto aplikácia používa na výber všetky dostupné volaním

```
status = clGetDeviceIDs (platform[0],CL_DEVICE_TYPE_ALL,num_devices,device,NULL);
```

Pre správu pamäťových objektov, komunikáciu hosťa so zariadením a udržovanie informácií programu a jadra vytvorené pre každé zariadenie existuje abstraktný kontajner, ktorý sa nachádza na hosťovi a nazýva sa kontext. Jeho vytvorenie prebieha volaním `clCreateContext()`, konkrétne:

```
context = clCreateContext (NULL, num_devices, device, NULL, NULL, &status);
```

kde prvý parameter je výber platformy, ten sa ale vybral v volaním `clGetPlatformIDs()`, `num_devices` je počet zariadení a `device` je ich zoznam, `&status` potom volá chybovú alebo úspešnú hlášku.
Keď chce hosť niečo vykonať na zariadení, tak do neho potrebuje posielať príkazy. Príkazy na vykonanie sa posielajú vytvorením fronty spôsobom:

```
queue = clCreateCommandQueue (context, device[],
                              CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE, &status);
```

`context` je kontext vytvorený vyššie, `device[0]` označuje prvé zariadenie v zozname. `CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE` umožňuje vykonávanie príkazov vo fronte aj keď ešte ostatné neskončili. Zariadenie tak môže lepšie využiť dostupné prostriedky (pamäť a multiprocesory) a celkovo sa tým môže urýchliť výpočet.
Po nastavení kontextu a fronty sa definujú pamäťové objekty. OpenCL podporuje dva typy objektov a to *buffery* alebo *obrázky*. S obrázkami sa v aplikácií nepracuje, preto je použitý typ buffer, čo je súvislý úsek pamäte podobný poliam. V tejto aplikácií sa používa iba jedno výstupné (optické) pole, ktoré je v globálnej premennej, aby k nemu mohli pristupovať všetky pracovné skupiny a multiprocesory. Buffer je vytvorený tak, aby sa do neho dalo iba zapisovať a to sa dosiahlo použitím `CL_MEM_WRITE_ONLY`. Celá definícia vytvorenia bufferu:

```
out_buffer = clCreateBuffer (context, CL_MEM_WRITE_ONLY,
                             sizeof (Complex_n) * DIMENSION,  NULL, &status);
```

Jadro programu OpenCL, ktoré je napísané v jazyku OpenCL C, je uložený ako reťazec znakov uložený buď priamo v hostiteľskom programe alebo zvlášť v súbore. Ak je jadro v súbore, tak musí byť najskôr uložené v pamäti pri behu aplikácie. API funkcia `clCreateProgramWithSource()` si následne načíta jadro do programu k ďalšiemu spracovaniu. Celé volanie funkcie je:

```
program = clCreateProgramWithSource (context, 1, (const char**) &programSource,
                                     NULL, &status);
```

kde pomocou `(const char **) &programSource` sa volá reťazec so zdrojovým kódom jadra.
Ďalším krokom je skompilovanie programu, ktorý je vykonávateľný na zariadení a to volaním príkazu `clBuildProgram()`. Tento príkaz umožňuje pridať niekoľko možností, s ktorými sa program skompiluje. Napr. je možné zadať preprocesoru konštanty alebo načítať zvlášť knižnice. Ďalej sú to voľby s pohyblivou desatinnou čiarkou, výpisy upozornení a chýb. Medzi voliteľné možnosti však patria aj optimalizačné voľby. Niektoré sú určené pre obrázky, napr. aliasingu a ostatné sa týkajú matematických výpočtov. Optimalizácie sa týkajú napr. nepoužívanie znamienok pre číslo 0, používanie len konečných čísel a iné. Všetky tieto možnosti sú zahrnuté jednou voľbou a to `-cl-fast-relaxed-math`. Ako už bolo spomenuté v kapitole 5, pre optické pole nie je potrebná vysoká presnosť výpočtov a tak je táto voľba zahrnutá v aplikácii. Okrem spomenutých volieb, ktoré sa nachádzajú v špecifikácií OpenCL existujú ešte aj iné, ktoré závisia od výrobcu zariadenia a použitého kompilátora. Tieto vlastné možnosti väčšinou nepridávajú žiadnu funkčnosť navyše, ale umožňujú rôzne výpisy pre ladenie chýb a lepšiu optimalizáciu zariadenia. Program sa potom skompiluje príkazom:

```
status = clBuildProgram (program, num_devices, device, "-cl-fast-relaxed-math",
                         NULL, NULL);
```

Poslednou vecou pre nastavenie náležitostí pred samotným vykonávaním na zariadení je definovanie, ktoré jadro sa má vykonať. Názov jadra sa vkladá do API funkcie `clCreateKernel ()` ako druhý argument.

```
kernel_model = clCreateKernel (program, "opt_field_model", &status);
```

Po tomto poslednom inicializačnom kroku sa začnú čítať súradnice modelu zo vstupného súboru. Po rozparsovaní na jednotlivé hodnoty sa kontroluje Z-ová súradnica. Ak sa súčasná hodnota nezhoduje s predošlou, tak sa vypočíta vzorové optické pole. Pre výpočet pomocou OpenCL sa musia nastaviť argumenty, ktoré vstupujú do jadra. Nastavenie argumentu môže vyzerať napr. takto:

```
status  = clSetKernelArg (kernel_model, 0, sizeof (cl_mem), &out_buffer);
```

pričom argumenty musia byť označené takým istým číslom, v akom poradí sú aj argumentmi v jadre. V aplikácií je počet argumentov 7 resp. 8, ale len jednému sa vytvoril samostatný buffer. Je to z toho dôvodu, že len ten jeden argument je výstupný a ostatné vstupné sú len čísla, ktoré je jednoduchšie a rýchlejšie len vložiť.
Po všetkých predošlých procedúrach už stačí na spustenie vykonávania jadra len jedna API funkcia `clEnqueueNDRangeKernel ()`, ktorá zabezpečuje posledné nastavenia pre spustenie jadra. Okrem iného sa nastavuje rozdelenie zariadenia na zadaný počet dimenzií, teda dá sa rozdeliť na jedno, dvoj alebo trojrozmerný priestor. Ďalej posun, o ktorý sa má zažať indexovanie, nezačne sa teda od 0, ale od zadaného posunu. Argument `global_work_size` určuje počet pracovných jednotiek v jednotlivých dimenziách. Nasleduje nastavenie `local_work_size`, čo je počet pracovných jednotiek v pracovnej skupine, taktiež tento argument môže byť jedno, dvoj a trojrozmerný. Veľkosť dimenzie by sa však mala rovnať s `global_work_size`. Po spustení aplikácie a zvolení počítania cez OpenCL je možné zvoliť rozdelenie 8 x 8 a 16 x 16. Rozdiel je potom v obsadení procesorov. Väčšie obsadenie však nezaručuje zrýchlenie aplikácie ako bolo písané v kapitole 2.3. Pri voľbe 8 x 8 je menšie a pri 16 x 16 zas väčšie obsadenie. Volanie funkcie je nasledovné:

```
status = clEnqueueNDRangeKernel (queue, kernel_model, 2, NULL, global_work_size,
                                 local_work_size, 0, NULL, NULL);
```

kde 2 predstavuje dvojrozmerné pole, posun nie je žiadny. `global_work_size` je definovaná ako pole s dvoma prvkami, ktoré sú X a Y, teda rozmery optického pola. X a Y však musia byť deliteľné 8, inak OpenCL skončí chybou.

```
size_t global_work_size[2] = {X, Y};
```

`local_work_size` je taktiež definovaná ako pole s dvoma prvkami, ale veľkosť zadáva užívateľ. Pri dvojrozmernom priestore musia byť hodnoty každého prvku mocniny čísla dva. Rozdielne nastavenie spôsobí väčšie využitie procesorov. Maximálna veľkosť je v súčasnosti na kartách NVIDIA 512

inštrukcií. Po vypočítaní na zariadení sa výsledky zapisujú do hosťa pomocou funkcie clEnqueueReadBuffer ():

```
status = clEnqueueReadBuffer (queue, out_buffer, CL_TRUE, 0,
                              sizeof (Complex_n) * DIMENSION, temp_array, 0, NULL,
                              NULL);
```

`out_buffer` je buffer, ktorý bol inicializovaný vyššie a bol určený len na zápis, myslí sa tým ale zápis len na zariadení, v tomto prípade sa výsledky z neho prepisujú do pola `temp_array`, ktoré bolo definované v hostiteľskej časti a môže sa teda používať bežným spôsobom.

6.3 Jadro OpenCL

V jadre OpenCL sa nachádza približne ten istý algoritmus ako, v prípade klasického výpočtu. Rozdiely sú v použití niektorých funkcií. Ako bolo spomínané, niektoré zložitejšie matematické funkcie ako je mocnina alebo odmocnina sú na klasických procesoroch veľmi rýchle, porovnateľne s rôznymi optimalizáciami vďaka špeciálnym registrom. Skalárne procesory v zariadeniach s podporou OpenCL však nič také nemajú. Preto sú tam takéto operácie pomalšie. Okrem mocniny a odmocniny tam patrí aj delenie. Okrem bežných dostupných matematických funkcií existujú aj tie isté, len s prefixom *native_,* ktoré sú síce menej presné, ale podľa testovania je citeľné zrýchlenie. Jadro OpenCL nepodporuje dvojrozmerné polia ako napríklad v jazyku C. Preto sú všetky výpočty umiestnené v jednorozmernom poli s veľkosťou X * Y z konfiguračného súboru a celá aplikácia taktiež pracuje s jednorozmernými poliami. Indexy dvoch rozmerov sa získavajú príkazmi `get_global_id (0)` a `get_global_id (1)`, ktoré pracujú rovnako ako inkrementovanie čísel v cykle for a vdaka tomu v jadre OpenCL nie je žiadny cyklus, ktorý nie je veľmi vhodný pre skalárne procesory.

6.4 Výpočet optického pola

Po získaní vzorového pola nastáva jeho posun, kde stred vzorového pola bude v súradniciach aktuálneho svetelného bodu. Do optického pola sa následne priráta viditeľná časť. Posun sa v aplikácií uskutočňuje pomocou `for` cyklov, ktorým predchádzajú podmienky o pozícií svetelného bodu a podľa toho sa určia hraničné body pre veľkosť pola, ktoré sa pripočíta k tomu optickému. Ak sa zmení Z-ová súradnica svetelného bodu, vypočíta sa nové vzorové optické pole, pričom aplikácia si sama neusporiadáva poradie pre rýchlejšie vykonávanie.

Po pripočítaní všetkých bodov tak vznikne optické pole, ktoré ale nie je kompletné. Musí sa ešte pripočítať prírastok zdrojového svetla. Všetky informácie o ňom musí obsahovať vstupný súbor okrem amplitúdy. Tá sa získa z optického pola, ktoré sa vypočítalo a to prevedením komplexných čísiel na reálne výrazom 6.1

$$\sqrt{a^2 + bi^2} \tag{6.1}$$

kde a je reálna zložka a bi je imaginárna zložka komplexného čísla a potom je amplitúda medián z celého pola. V aplikácii však používam len aritmetický priemer, ktorý je na základné zobrazenie postačujúci. Po zistení amplitúdy a pripočítaní vznikne kompletné optické pole, ktoré sa vypíše do zvoleného súboru. Od veľkosti amplitúdy závisí, aký svetlý a kontrastný bude výsledný objekt. Čím väčšia amplitúda, tým svetlejší objekt bude a naopak menej kontrastný. Takýmto spôsobom je potom možné upravovať scénu a dostať tak požadujúcu.
Po aplikovaní výrazu 6.1 na takto upravené optické pole, dostaneme hologram.

V tejto kapitole bola bližšie popísaná aplikácia z viac programátorského pohľadu, nastavenie niektorých parametrov vstavaných funkcií a konkrétnejší popis algoritmov. Pre detailnejšie informácie je potrebné nahliadnuť do zdrojových kódov, ktoré sú na priloženom CD.

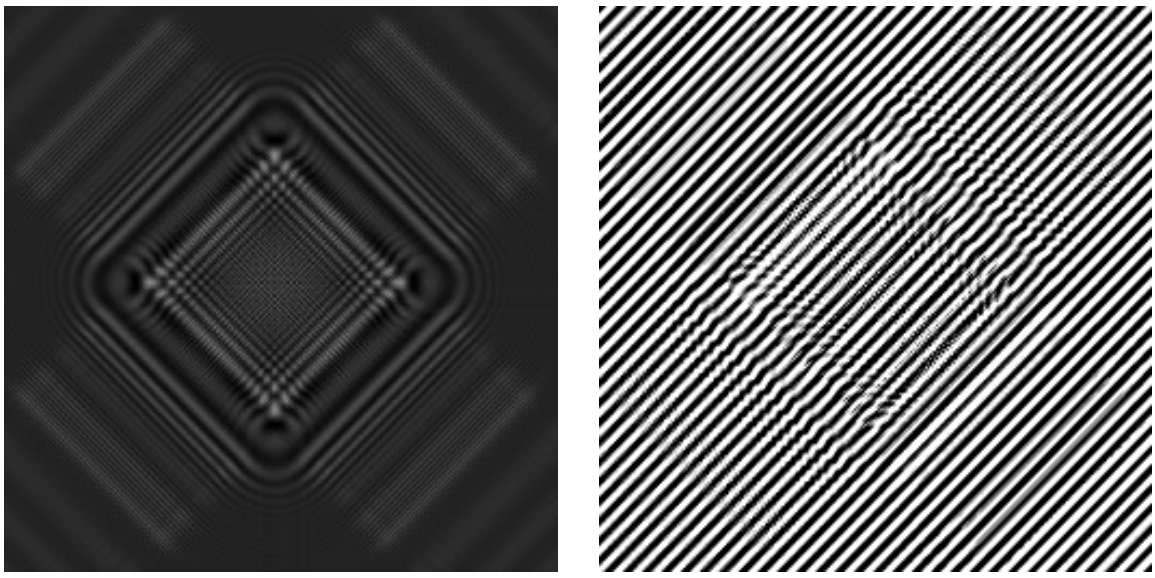
7 Výsledky a vyhodnotenie

Siedma kapitola predstaví výstupy a dosiahnuté výkonnostné výsledky a porovnania. Doteraz bolo optické pole len popísané bez toho, aby bola zobrazená lepšie viditeľná podoba. V nasledujúcich podkapitolách však budú zobrazené aj grafické podoby optického pola a aj hologramu. Na zobrazenie sú použité aplikácie z Holography Toolkit [25], ktoré vytvárajú hodnoverné výstupy.

7.1 Výstup aplikácie

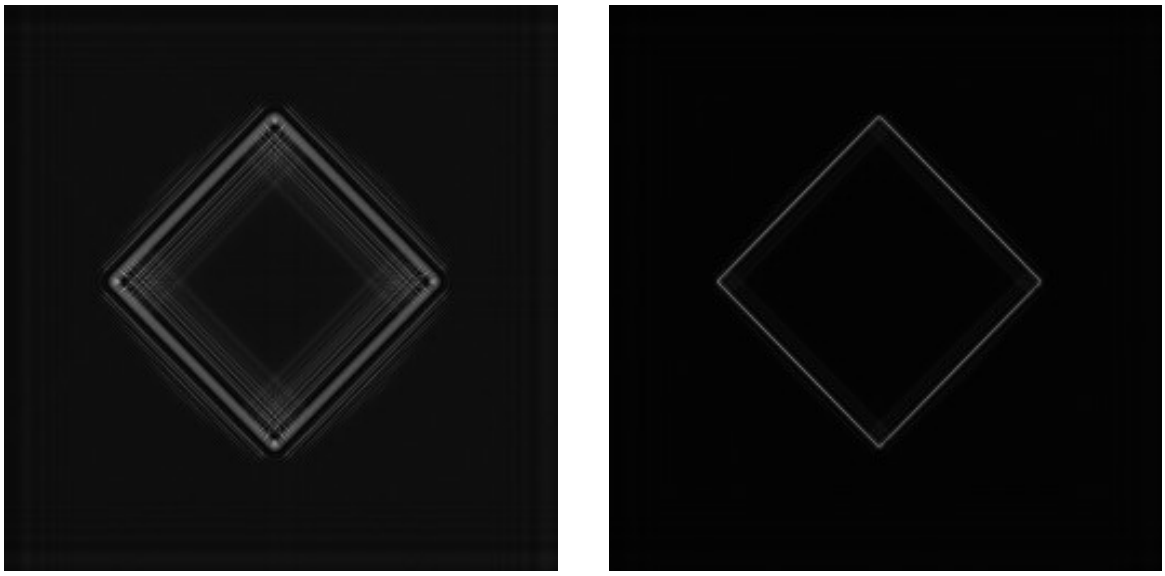
Vstup aplikácie, ktorým je model v priestore, bol predstavený v kapitole 6.1. Po spracovaní vzniklo optické pole, ktoré je uložené v súbore obsahujúci komplexné čísla. Aby sa mohol použiť Holography Toolkit musí byť súbor v presnom formáte a to takom, že reálnu zložku nasleduje imaginárna oddelená znamienkom. Imaginárna časť musí obsahovať na konci i a čísla musia obsahovať medzi sebou čiarku okrem konca riadku. Dôležité je aj usporiadanie a to do štvorcovej matice. Pre spracovanie a zobrazenie používa toolkit vlastný binárny formát s príponou .df. V toolkite sa ale tiež nachádza skript pre program Matlab, ktorý skonvertuje súbor ak je v horeuvedenom formáte. Vzniknutý binárny súbor už len stačí predať aplikácií. Pre zobrazenie boli použité dve aplikácie z toolkitu a to HoloPorpagLarge a DftoHologram. HoloPropagLarge slúži na propagáciu optického pola medzi dvoma paralelnými plochami. To znamená, že výstupom bude zobrazenie pola v nastavenej výške a nie len v nulovej. DftoHologram zas slúži na výpočet hologramu z optického pola.

Ako pokusný objekt slúžili hrany kocky s veľkosťou 300 px, čo pri veľkosti vodu 2μm predstavuje 0,6 mm a bola osovo otočená o 45°, no s optickým polom bola rovnobežná. Plocha mala rozlíšenie 1 024 x 1 024 px a zdrojové svetlo malo vlnovú dĺžku 635 nm. Na obrázku 6.1 je zobrazená propagácia v nulovej výške. Spodná hrana kocky začína vo vzdialenosti 0,11 mm nad optickým polom.

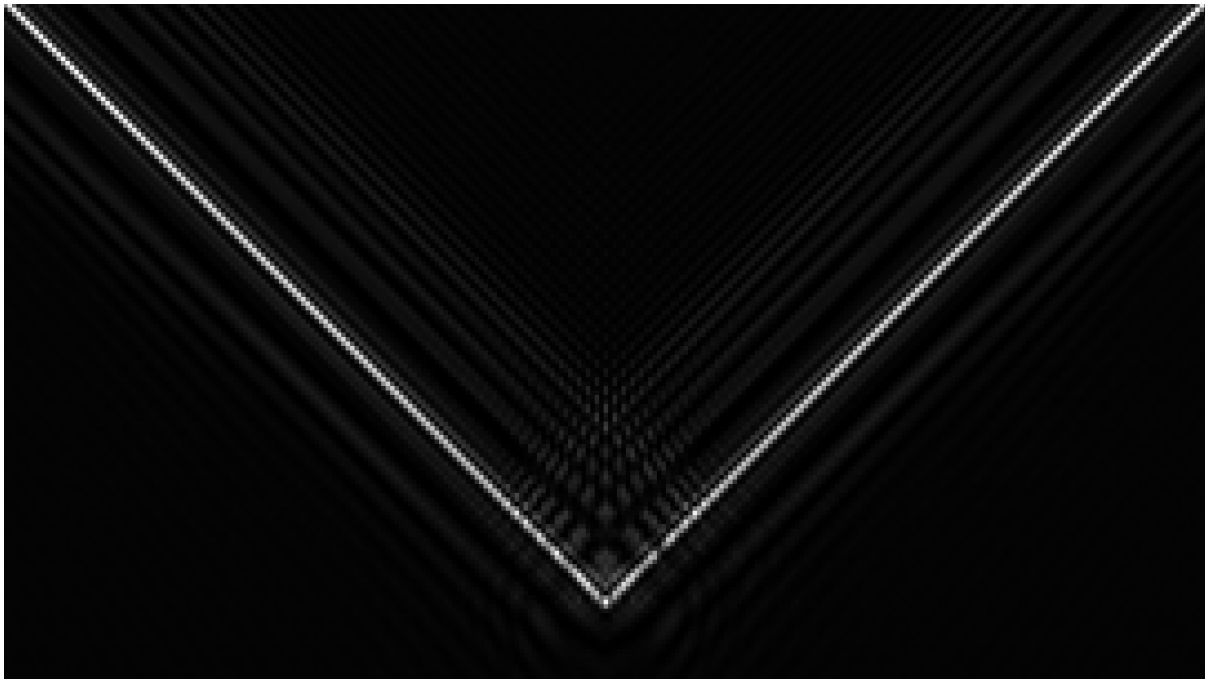


Obrázok 6.1: Vľavo zobrazenie propagácie pootočenej kocky, rovnobežnej s optickým polom. Vpravo zobrazenie hologramu.

Z propagácie na obrázku 6.1 vidno len veľmi hrubý obrys hrán kocky. Keďže každý bod je zároveň svetelným bodom, tak vyzerá zobrazenie akoby bola kocka vyprsknutá na plochu. Pri posunutí zobrazovanej plochy vyššie obraz dostáva jasnejšie kontúry a pri umiestnení do výšky spodnej hrany kocky je obraz už veľmi ostrý. Hologramy sú veľmi podobné bez okom badateľných rozdielov, preto nie sú zobrazované.

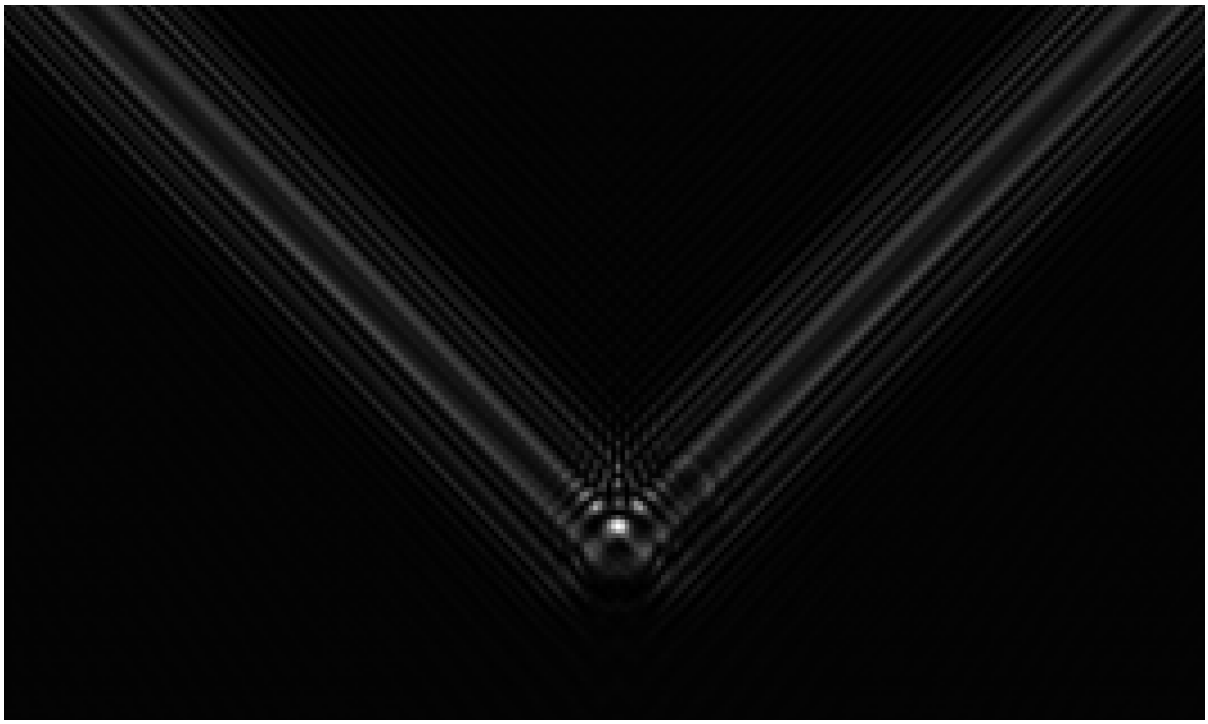


Obrázok 6.2: Vľavo je umiestnenie zobrazovania 1 mm pod spodnou hranou kocky.
Vpravo je zobrazený rez v spodnej hranekocky.



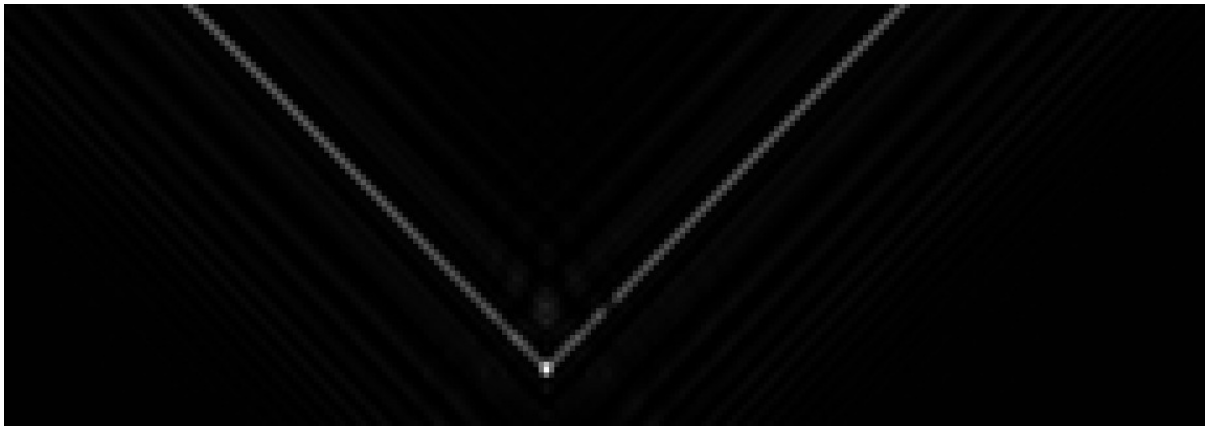
Obrázok 6.3: Detail rožnej časti kocky v reze cez spodný okraj.

Keď zobrazovacia rovina prechádza tesne hranou, tak by mal byť obraz zrejme ostrý. Nie je tomu však tak, pretože body už majú na spodnej strane od ostatných veľkú intenzitu a hlavne v rohoch je viditeľné „presvitanie“. Vďaka tomu sú však body také svetlé a dobre kontrastujú s okolím. Keďže svetlo z bodov vyžaruje všetkými smermi, je v strede kocky najväčší rozptyl do strán.



Obrázok 6.4: Detail rožnej časti kocky v reze cez stred.

Na obrázku 6.4 je detailný pohľad na rez kocky jej stredom vo výške 11,3 mm. Nastáva tu efekt, pri ktorom sa dva svetelné zdroje pri interferencii akoby rozdelia a najsvetlejší bod sa tak stáva ten kolmý na plochu, ktorý ma veľkú intenzitu od všetkých bodov. Na obrázku 6.5 je naopak vidno najostrejší obraz, ale zároveň málo jasný a kontrastný, pretože smerom nahor nie je taká svetelná intenzita. Body kolmé na rovinu sú opäť jasnejšie.



Obrázok 6.5: Detail rožnej časti kocky v reze cez vrchnú časť.

7.2 Testovanie

Cieľom diplomovej práce nebolo len vypočítanie optického pola a jeho zobrazenie. Hlavným cieľom bola akcelerácia týchto výpočtov na čo možno najkratší čas a dosiahnuté výsledky sa nachádzajú v tejto podkapitole.

Testovanie prebehlo vždy s dvoma vzorkami. Jedna testovacia vzorka bola rovnaká ako v kapitole 7.1, teda rozmery 1 024 x 1 024 px a 300 px na hranu kocky, čiže objekt s počtom bodov 3600. Druhá vzorka bola v podstate rovnaká, len sa zmenila veľkosť pola na 2 048 x 2 048 px. Väčšie rozlíšenie pola bolo nutné kvôli niektorým grafickým kartám, ktoré mali aj napriek svojej papierovo rozdielnym výkonom takmer totožné výsledky. Pri väčšom rozlíšení sa však už prejavili rozdiely.

Aplikácia sa spúšťa buď z konzoly alebo priamo dvojklikom. Pri spúštaní z konzole je nutné napísať argumenty v tomto poradí: *vstupny_subor vystupny_subor metoda_pocitania rozdelenie_skupiny*

Vstupným súborom sa myslí model vo formáte uvedenom v kapitole 6.1, výstupný súbor môže mať ľubovolný názov ale bez medzier tak ako aj vstupný. Metóda počítania je buď *1* pre počítanie cez OpenCL alebo *2* pre klasické počítanie. Ak je zvolená metóda cez OpenCL, tak sa musí ešte zadať počet pracovných jednotiek do pracovnej skupiny. Možnosti sú *1* pre 8 x 8 a voľba *2* pre 16 x 16. Pri spustení aplikácie v nekonzolovom režime, tak sa tieto voľby zadávajú postupne, ako si ich aplikácia pýta. Na výstup aplikácia vypíše celkový čas behu aplikácie a čas algoritmu. Myslí sa tým čas, ktorý bol potrebný na výpočet vzorových polí, bez času stráveného posúvaním pola. Tento čas by mal byť z hľadiska zvýšenej efektivity najviac výrazný.

V nasledujúcich riadkoch budú testovacie zostavy a výsledné časy.

Konfigurácia č. 1.

CPU (počet jadier)	Intel Core 2 Duo P8400 @ 2.26 Ghz (2)
Veľkosť pamäte	2 GB
GPU (počet jadier)	NVIDIA Quadro NVS 150M (8)
Veľkosť pamäte	512 MB
Verzia ovládačov GNU/Linux	195.36
Verzia ovládačov Windows	197.16

Tabuľka 7.1: Hardvérová konfigurácia.

	1 024 x 1 024		**2 048 x 2 048**	
	Celkový čas	**Čas algoritmu**	**Celkový čas**	**Čas algoritmu**
GPU, 8 x 8	0m 43s	0m 24s	2m 38s	1m 40s
GPU, 16 x 16	0m 43s	0m 24s	2m 38s	1m 39s
CPU	1m 20s	1m 1s	5m 6s	4m 6s
Zrýchlenie %	46	60	48	60

Tabuľka 7.2: OS Windows XP, kompilátor z Visual Studio 2008 Professional.

	1 024 x 1 024		2 048 x 2 048	
	Celkový čas	**Čas algoritmu**	**Celkový čas**	**Čas algoritmu**
GPU, 8 x 8	1m 28s	0m 25s	5m 2s	1m 39s
GPU, 16 x 16	1m 28s	0m 25s	5m 1s	1m 39s
CPU	4m 18s	3m 16s	16m 31s	13m 8s
Zrýchlenie %	66	87	70	87

Tabuľka 7.3: OS Windows XP, kompilátor GCC, verzia 4.6.1.

	1 024 x 1 024		2 048 x 2 048	
	Celkový čas	**Čas algoritmu**	**Celkový čas**	**Čas algoritmu**
GPU, 8 x 8	1m 16s	0m 24s	4m 20s	1m 38s
GPU, 16 x 16	1m 16s	0m 24s	4m 20s	1m 38s
CPU	2m 16s	1m 25s	8m 36s	5m 54s
Zrýchlenie %	44	72	50	72

Tabuľka 7.4: OS Debian 6, kompilátor GCC, verzia 4.4.5.

Na výsledkoch prvej konfigurácie je vidieť veľký rozdiel medzi kompilátormi aj na tom istom operačnom systéme. Čo však stojí za pozornosť je fakt, že výpočet na grafickej karte pomocou OpenCL je na každej aplikácií rovnako rýchly a vôbec tak nezávisí od toho, ako sa preloží hosťovský kód a takisto to nijak rapídne nezávisí od ovládačov. To môže byť veľmi výhodné pre multiplatformné vyvíjanie aplikácií. Výsledky sú celkom prijateľné aj napriek tomu, že grafická karta je relatívne stará a vyšla v začiatkoch pôsobenia OpenCL.

Konfigurácia č. 2.

CPU (počet jadier)	Intel Core i7 Q720 @ 1.6 Ghz (4)
Veľkosť pamäte	4 GB
GPU (počet jadier)	NVIDIA GeForce GT 330M (64)
Veľkosť pamäte	1 GB
Verzia ovládačov Windows	295.73

Tabuľka 7.5: Hardvérová konfigurácia.

	1 024 x 1 024		2 048 x 2 048	
	Celkový čas	**Čas algoritmu**	**Celkový čas**	**Čas algoritmu**
GPU, 8 x 8	0m 16s	0m 5s	0m 54s	0m 20s
GPU, 16 x 16	0m 16s	0m 5s	0m 54s	0m 21s
CPU	1m 11s	1m 0s	4m 42s	4m 8s
Zrýchlenie %	77	99	80	99

Tabuľka 7.6: OS Windows 7 64bit.

Na druhej konfigurácií dopadli výsledky veľmi dobre a oproti prvej je nárast rýchlosti viditeľný. Pri menšom poli sú výsledky algoritmu veľmi malé a je možné, že najviac času zabrala len konfigurácia grafickej karty. Opäť sa však neprejavila zmena počtu výpočtových jednotiek v pracovnej skupine.

Konfigurácia č. 3.

CPU (počet jadier)	Intel Core i5 2400S @ 3.1 Ghz (4)
Veľkosť pamäte	4 GB
GPU (počet jadier)	AMD Radeon HD7850 (1024)
Veľkosť pamäte	2 GB
Verzia ovládačov Windows	12.4

Tabuľka 7.7: Hardvérová konfigurácia.

	1 024 x 1 024		2 048 x 2 048	
	Celkový čas	**Čas algoritmu**	**Celkový čas**	**Čas algoritmu**
GPU, 8 x 8	0m 10s	0m 3s	0m 34s	0m 9s
GPU, 16 x 16	0m 10s	0m 3s	0m 34s	0m 9s
CPU	0m 50s	0m 43s	3m 11s	2m 49s
Zrýchlenie %	80	99	82	99

Tabuľka 7.5: OS Windows 7 64bit.

Nakoniec zostala najvýkonnejšia hardvérová zostava. Výsledky sa opäť ešte viac zlepšili v prospech OpenCL. Pre jednoduché výpočty a vo veľkom množstve je zdá sa OpenCL najlepšia voľba, za podmienky, že sa dá využiť paralelizmus. Opäť ale zväčšenie obsadenosti nepomohlo, ale ani nezhoršilo výsledky.

V tejto kapitole boli prezentované dosiahnuté výsledky a tie sa celkom podarili. Zníženie času je o niekoľko desiatok percent percent a aj pri najslabšej konfigurácií sa zrýchlenie pohybovalo nad 50%. Ďalšie testovacie vzorky sa nachádzajú v prílohe C.

8 Záver

Ako názov práce napovedá, cieľom bola akcelerácia výpočtu optického pola. Voľbou pre implementáciu bol framwork OpenCL, ktorý je zameraný na vysokoparalelné výpočty. Frameworku a jeho architektúre sa venovala druhá kapitola, v ktorej boli spomenuté hlavné informácie. Ďalšia kapitola sa venovala optickej holografii, ale aj základným princípom na ktorých je holografia postavená. Digitálna holografia bola hlavnou súčasťou nasledujúcej kapitoly. Až po týchto nadobudnutých vedomostiach som mohol začať navrhovať aplikáciu, pretože bez potrebných vedomostí by zrejme návrh nebol úplne výhodný a efektívny. Ešte počas navrhovania sa musel brať ohľad aj na optimalizáciu, aby sa dala využiť najväčšia výhoda OpenCL a tou je paralelizmus. Po návrhu som mohol začať s implementáciou, ktorá najskôr prebiehala bez použitia frameworku OpenCL. Až po overení funkčnosti som aplikáciu prepísal, aby využívala OpenCL.

Jednou z najťažších vecí bolo odladiť OpenCL tak, aby som sa dostal k požadovanej akcelerácií. Každý výrobca má totiž vlastné nástroje a spôsoby, ktoré nie sú kompatibilné s ostatnými. Po odladení a testovaní na rôznych zariadeniach sa výsledky oproti konvenčnému riešeniu výrazne líšili a to v prospech OpenCL. Zrýchlenie celej aplikácie je v niektorých prípadoch niekoľkonásobné. Dokonca aj na starom hardvéri, na ktorom OpenCL ešte len začínal, ktorý nepodporuje mnoho súčasných funkcií sú výsledky viac, než uspokojivé, keďže zrýchlenie bolo nad 50 %. Nové zariadenia prekonali hranicu 80-ich percent, čo považujem za úspech. Výhoda je aj v jednoduchom vývoji aplikácií a taktiež široká podpora platforiem od klasických CPU, cez GPU až po špecializované hardvérové akcelerátory.

Aj napriek úspechu v podobe zrýchlenia je momentálne holografia nie veľmi použiteľná a tak zrejme žiaľ nebude. Moja skepsa pramení z toho, že aj na výpočet statického drobného objektu, ledva viditeľný okom, je potrebný vysoký výkon a okrem toho aj pamäť, aj keď tento problém sa dá vyriešiť napr. použitím FFT. Pri videách musí byť tento výkon a pamäť ešte znásobená. Samozrejme je tu priestor na zlepšenie a v prípade väčšieho povedomia ľudí o tejto technológií sa môže situácia zlepšiť aj v relatívne krátkom čase.

Prácu je možné rozširovať o viaceré vylepšenia a prispieť tak k ešte väčšiemu zrýchleniu. Jednou z možností, ktorá už bola spomenutá je odstrániť periodicitu a pracovať tak s menším polom. Ďalšou možnosťou je preniesť posun pola na zariadenie s OpenCL a vykonávať tak paralelne ešte viac úkonov, keďže ako vidno z testov v kapitole 7.2 najviac brzdí výkon práve pomalé sekvenčné CPU. Pre veľké hologramy to zas môže byť zameranie na spomínanú Rýchlu Fourierovú transformáciu alebo veľa iných možností.

Literatúra

[1] Khronos Group [online]. 2010 [cit. 2012-05-23]. Dostupné z WWW: <http://www.khronos.org/>.

[2] Nobel Media. Nobelprize [online]. 2010. Dostupné z WWW: <http://nobelprize.org/nobel_prizes/physics/laureates/1971/>.

[3] KIRCOS, Bill. Larrabee [online]. 2010-05-25. Dostupné z WWW: <http://blogs.intel.com/technology/2010/05/an_update_on_our_graphics-rela.php>

[4] NVIDIA Corporation. CUDA [online]. 2012. Dostupné z WWW: <http://developer.nvidia.com/what-cuda>.

[5] NVIDIA Corporation. DirectCompute [online]. 2012. Dostupné z WWW: <http://developer.nvidia.com/directcompute>.

[6] Advanced Micro Devices, Inc.: *Stream Computing OpenCL Programming Guide* [online]. Advanced Micro Devices, Inc., revision 1.03, 2010 [cit. 2012-05-23]. Dostupné z WWW: <http://developer.amd.com/gpu_assets/ATI_Stream_SDK_OpenCL_Programming_Guide.pdf>

[7] Advanced Micro Devices, Inc.: *AMD Accelerated Parallel Processing OpenCL* [online]. Advanced Micro Devices, Inc., revision 1.3f, 2011 [cit. 2012-05-23]. Dostupné z WWW: <http://developer.amd.com/sdks/amdappsdk/assets/amd_accelerated_parallel_processing_opencl_programming_guide.pdf>

[8] Stanford University Graphics Lab. BrookGPU [online]. 2010. Dostupné z WWW: <http://graphics.stanford.edu/projects/brookgpu/>

[9] Intel Corporation. Sh [online]. 2010. Dostupné z WWW: <http://www.libsh.org/>.

[10] The OpenMP API specification for parallel programming [online]. 2011. Dostupné z WWW: <http://openmp.org/wp/>

[11] NVIDIA Corporation: *OpenCL Programming Guide for the CUDA Architecture* [online]. NVIDIA Corporation, version 4.2, 2012-09-03 [cit. 2012-05-23]. <http://developer.download.nvidia.com/compute/DevZone/docs/html/OpenCL/doc/OpenCL_Programming_Guide.pdf>.

[12] NVIDIA Corporation. GeForce FX 5500 [online]. 2010 [cit. 2012-05-23]. Dostupné z WWW: <http://www.nvidia.co.uk/page/fx_5200.html>.

[13] NVIDIA Corporation. GeForce GTX 480 [online]. 2010 [cit. 2012-05-23]. Dostupné z WWW: <http://www.nvidia.co.uk/object/product_geforce_gtx_480_uk.html>.

[14] GASTER Benedict R..; HOVES Lee; KAELI david; MISTRY Perhaad; SCHAA Dana: *Heterogeneous Computing with OpenCL*. Morgan Kaufmann, 2011-08-31 [cit. 2012-05-23]. ISBN-13: 978-0123877666; ISBN-10: 0123877660.

[15] NVIDIA Corporation: *NVIDIA OpenCL Best Practices Guide* [online]. NVIDIA Corporation, version 1.0, 2009-08-10 [cit. 2012-05-23]. Dostupné z WWW: <http://nvidia.com/content/cudazone/CUDABrowser/downloads/papers/NVIDIA_ OpenCL_BestPracticesGuide.pdf>.

[16] MUNSHI Aaftab, GASTER Benedict, MATTSON Timothy G., FUNG James, GINSBURG Dan: *OpenCL Programming Guide*. Addison-Wesley Professional, 2011-07-23 [cit. 2012-05-23]. ISBN-13: 978-0321749642; ISBN-10: 0321749642.

[17] VOLKOV V.: *Better performance at lower occupancy* [online]. GPU Technology Conference 2010, 2010-09-22 [cit. 2012-05-23]. Dostupné z WWW: <www.cs.berkeley.edu/~volkov/volkov10-GTC.pdf>.

[18] Khronos OpenCL Working Group: *The OpenCL Specification* [online]. Khronos Group, version 1.0, revision 48, 2009 [cit. 2012-05-23]. Dostupné z WWW: <www.khronos.org/registry/cl/specs/opencl-1.0.pdf>.

[19] GOODMAN Joseph W.: *Introduction to Fourier Optics*. Roberts & Company Publishers, 3rd edition, 2004-12-10 [cit. 2012-05-23]. ISBN-13: 978-0070242548; ISBN-10: 0070242542.

[20] BORN M.; WOLF E.: *Principles of Optics*. Cambridge University Press, 7th edition, 1999-10-13 [cit. 2012-05-23]. ISBN-13: 9780521642224; ISBN-10: 0521642221.

[21] RP Photonics. Encyclopedia of Laser Physics and Technology. 2008-10-01. Dostupné z WWW: <http://www.rp-photonics.com/interference.html>

[22] BENTON Stephen A.; BOVE V. Michael Jr.: *Holographic Imaging*. Wiley-Interscience, 2008-04-14 [cit. 2012-05-23]. ISBN-13: 978-0470068069; ISBN-10: 047006806X.

[23] HANÁK, Ing. Ivo. *Urychlení výpočtu číslicového hologramu*. Plzeň, 2010. 142 s. Disertační práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky. Dostupné z WWW: <http://graphics.zcu.cz/Download-document/158-Accelerating-digital-hologram-generation>.

[24] DALÍK J.; PEŠL J.: *Iterační metody pro řešení algebraických rovnic.* Fakulta stavební Vysokého učení technického v Brně; Fakulta informatiky Masarykovy univerzity v Brně, 2001. Dostupné z WWW: <http://math.fce.vutbr.cz/vyuka/matematika/numericke_metody/>

[25] Holography Toolkit: *Nástrojová sada pro digitální holografii.* Fakulta aplikovaných věd Západočeské univerzity v Plzni, 2008. Dostupné z WWW: <http://www.kiv.zcu.cz/en/research/downloads/product-detail-en.html?produkt_id=31>

Zoznam príloh

Príloha A

Obsah CD

Priložené CD obsahuje:

- zdroj technickej správy
- technickú správu vo formáte PDF
- zdrojové kódy aplikácie
- vstupný súbor s modelom hrán kocky
- spustiteľný súbor pre Microsoft Windows XP/Vista/7, 32/64 bit
- spustiteľný súbor pre GNU/Linux 64bit

Príloha B

Ukážka vstupného súboru

X: 1024

Y: 1024

Lambda: 0.000000635

Pixel :0.000002

sourceX 490

sourceY 490

sourceZ 60000000

sourcephase: 0

500;200;5500;1.000000;0.000000

501;201;5500;1.000000;0.000000

502;202;5500;1.000000;0.000000

503;203;5500;1.000000;0.000000

504;204;5500;1.000000;0.000000

505;205;5500;1.000000;0.000000

506;206;5500;1.000000;0.000000

507;207;5500;1.000000;0.000000

508;208;5500;1.000000;0.000000

509;209;5500;1.000000;0.000000

510;210;5500;1.000000;0.000000

511;211;5500;1.000000;0.000000

512;212;5500;1.000000;0.000000

513;213;5500;1.000000;0.000000

514;214;5500;1.000000;0.000000

515;215;5500;1.000000;0.000000

Príloha C

Testovacie vzorky

Konfigurácia č. 4.

CPU (počet jadier)	Intel Xeon X5675 @ 3.06 Ghz (6)
Veľkosť pamäte	8 GB
GPU (počet jadier)	NVIDIA Quadro 4000 (256)
Veľkosť pamäte	2 GB
Verzia ovládačov Windows	276.28

	1 024 x 1 024		**2 048 x 2 048**	
	Celkový čas	**Čas algoritmu**	**Celkový čas**	**Čas algoritmu**
GPU, 8 x 8	0m 11s	0m 3s	0m 42s	0m 14s
GPU, 16 x 16	0m 11s	0m 3s	0m 42s	0m 14s
CPU	0m 49s	0m 42s	3m 16s	2m 49s
Zrýchlenie %	77	99	78	99

Konfigurácia č. 5.

CPU (počet jadier)	Intel Core i7 930 @ 2.8 Ghz (4)
Veľkosť pamäte	4 GB
GPU (počet jadier)	NVIDIA GeForce GTX 560 TI (384)
Veľkosť pamäte	1 GB
Verzia ovládačov Windows	295.73

	1 024 x 1 024		**2 048 x 2 048**	
	Celkový čas	**Čas algoritmu**	**Celkový čas**	**Čas algoritmu**
GPU, 8 x 8	0m 13s	0m 3s	0m 40s	0m 12s
GPU, 16 x 16	0m 11s	0m 3s	0m 40s	0m 12s
CPU	0m 55s	0m 46s	3m 35s	3m 7s
Zrýchlenie %	80	99	81	99