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

AKCELERACE ALGORITMŮ NA ARCHITEKTUŘE LARRABEE

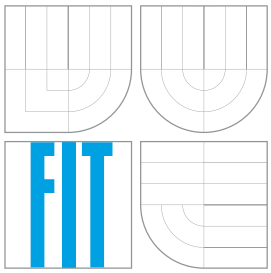
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. IVO VESELÝ

BRNO 2010

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

AKCELERACE ALGORITMŮ NA ARCHITEKTUŘE LARRABEE

ALGORITHM ACCELERATION ON LARRABEE PLATFORM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. IVO VESELÝ

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Dr. Ing. PAVEL ZEMČÍK

BRNO 2010

Abstrakt

Intel Larrabee je jednou z prvních plně programovatelných grafických architektur. Práce popisuje tuto více-jádrovou architekturu z pohledu hardwarové implementace i z pohledu programovacího modelu. Larrabee sází na mnoho úplných in-order jader vystavěných nad instrukční sadou x86. Jádra obsahují čtyři hardwarová vlákna, každé vybavené svou vlastní sadou registrů, a novou vektorovou jednotkou. Vektorová jednotka společně s rozšířením instrukční sady rapidně zvyšují výkonnost systému. Nové režimy cachování přispívají ke zvýšení propustnosti i v případě nespojitých datových struktur. Zaměření této architektury proto není jen počítačová grafika nebo zpracování obrazu ale všechny paralelní úkoly. Druhá část textu se zabývá syntézou hologramu. Konkrétně přináší dvě nové metody pro generování množiny bodových světelných zdrojů se zadanou vyzařovací charakteristikou.

Abstract

Intel Larrabee is one of the first of fully programmable graphical architectures. Thesis describes this many-core architecture by hardware implementation and programmer's model point of view. Larrabee bets on many complete in-order cores, built over x86 instruction set. Cores contains four hardware threads, each with it's own register file, and new vector processing unit. Vector processing unit together with instruction set extension rapidly increases system performance. New cache modes helps to increase throughput even when irregular data structures. This architecture is not focused only on computer graphics nor image processing, but all parallel tasks. Second part of this text deals with hologram synthesis. Specifically, it brings two new methods for patch of point light sources generation with concrete radiation.

Klíčová slova

Larrabee, počítačová grafika, zpracování obrazu, více-jádrové počítání, paralelní počítání, SIMD, GPGPU, syntéza hologramu, optické pole.

Keywords

Larrabee, computer graphics, image processing, many-core computing, parallel computing, SIMD, GPGPU, hologram synthesis, optical field.

Citace

Ivo Veselý: Akcelerace algoritmů
na architektuře Larrabee, diplomová práce, Brno, FIT VUT v Brně, 2010

Akcelerace algoritmů na architektuře Larrabee

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana doc. Dr. Ing. Pavla Zemčíka.

.....
Ivo Veselý
25. května 2010

Poděkování

Rád bych touto cestou poděkoval své mamince: „Vždy jsi ve mě věřila a stála za mnou, děkuji.“ Také bych rád poděkoval panu docentu Zemčíkovi za podporu při studiu a řešení tohoto projektu.

Obsah

Kapitola 1

Úvod

Firma Intel stále přichází na trh s novými procesory. Jedním z nejnovějších je Larrabee. Procesor určený především pro počítačovou grafiku. Ačkoliv Larrabee je spíše architektura, než konkrétní produkt. Tento dokument se zabývá možností urychlení některých výpočtů v této oblasti použitím architektury Intel Larrabee.

V počátcích osobních počítačů byla grafická pipeline (vykreslovací řetězec) implementovaná čistě softwarově za pomoci CPU. Jejich výkon ale nedostačoval nárokům kladeným ze strany spotřebitelů a proto se začala pomalu ale jistě vytvářet specializovaná řešení. Nejprve pouze jako grafické koprocesory – pomocné funkční bloky řešící nejvíce problémové části vykreslovacího řetězce. Později samostatné procesory usazované na sběrnici, řešící celý proces zobrazování. Až k celým grafickým kartám umístěným na dedikované sběrnici. Procesory na těchto kartách (GPU) dosáhli v dnešní době značného stupně vývoje. Také technologie kterými dnes disponujeme jsou na takové úrovni, že nám dovolují opustit hardwarové řešení některých částí vykreslovacího řetězce a implementovat je v software s výhodou větší flexibility. Při použití softwarového řešení již nejsme vázáni tím co dali vývojáři našemu GPU do vínku. Také pro trh jdoucí dnes převážně po nízké ceně a nízké spotřebě energie, je toto řešení přijatelnější. Je to dáno především tím, že takto stoupá využití plochy čipu v čase. Máme-li procesor, na jehož čipu jsou implementované jednotlivé části pipeline je v daném okamžiku využito pouze několik procent plochy čipu. Ostatní části čekají až na ně příjde řada, což je plýtvání.

Moderní grafické prcesory v reakci na tento trend zavádějí shaderovací jednotky[1] a tím programovatelnou pipeline. Pořád jsme ale vázáni logikou která obsluhuje linku. Jako například paměťový model, fixní bloky, atd. Zde navazuje architektura Intel Larrabee „návratem ke strým časům“. Je to vysoce paralelní architektura která dělá celý vykreslovací řetězec plně programovatelným. Můžeme tak využít celou plochu čipu k transformacím vrcholů a o chvíli později znovu celou plochu čipu k rasterizaci.

Některé úkoly v počítačové grafice a ve zpracování obrazu jsou stále velice závislé na surovém výpočetním výkonu. Náplní této práce je prostudovat architekturu Intel Larrabee a zhodnotit jakým způsobem by se dala využít pro akceleraci těchto výpočtů. Součástí práce je tak i návrh a implementace algoritmu z výše popsané třídy jako demonstrace jeho schopností. Je jedním z prvních mikroprocesorů které si kladou za cíl nahradit fixní vykreslovací řetězec a zároveň sloučit opět hlavní procesor s grafickým i když ne nutně[2].

[1] Malé výpočetní jednotky poskytující několik základních operací implementované ve velkém množství k dosažení velkého stupně datového paralelismu.

[2] Intel Larrabee může běžet jako GPU s vlastním operačním systémem a vykreslovacím řetězcem jen jako jednou z aplikací [1].

A tím se právě odlišuje od konkurenčních produktů, které mají sloužit jen jako grafické čipy bez logiky nutné pro běh operačního systému.

Architektura Intel Larrabee je relativně nová a tak je její popis náplní následující kapitoly. Jsou zde uvedeny důvody jejího vzniku, popis architektury jak z pohledu programovacího modelu, tak z pohledu implementačního. A především přehled rozšíření instrukční sady x86 o nové instrukce LRBni (Larrabee new instructions). Informace jsou podávány formou srovnání s instrukcemi již standardní technologie SSE. Tím jak by nová architektura mohla pomoci při řešení výpočetně náročných úkolů počítačové grafiky či zpracování obrazu a jaké prostředky nabízí je diskutováno ve třetí části tohoto dokumentu. Jsou zde rozebrány techniky masivní paralelizace a vektorizace výpočtů a tyto techniky jsou poté aplikovány při návrhu algoritmu demonstrujícího možnosti akcelerace. Čtvrtá kapitola se zabývá návrhem a implementací algoritmů syntézy hologramu. Konkrétně jde o metody výpočtu množiny bodových světelných zdrojů se zadanou vyzařovací charakteristikou. V další části jsou diskutovány výsledky práce vůči existující implementaci. Shrnuje vlastnosti navržených algoritmů a pomocí grafů demonstruje výsledky. Závěrečná kapitola je věnována kvalitativnímu shrnutí práce, diskuzi přínosu a možnostem pokračování.

Tato práce navazuje na semestrální projekt na Fakultě Informačních Technologií Vysokého Učení Technického v Brně. Od třetí kapitoly dále je na semestrální projekt plynule navázáno plněním dalších bodů zadání. Bohužel se po obhajobě semestrálního projektu ukázalo, že Intel architekturu Larrabee pravděpodobně neuvolní včas a tak třetí kapitola uvádí čtenáře do jakéhosi nového konceptu práce, který je však v souladu se zadáním. Jsou zde také rozvedeny podrobnější informace o neuvolnění cílové architektury.

Kapitola 2

Architektura Intel Larrabee

Tato kapitola seznamuje čtenáře s okolnostmi vedoucími ke vzniku architektury Larrabee z pohledu vývojářů. Dále představuje základní stavební bloky architektury a programovací model který podporuje obecné zaměření paralelní platformy jako zpracování obrazu, fyzikální simulace a medicínské a finanční analýzy. V závěru kapitoly je představeno rozšíření instrukční sady x86 LRBni formou srovnání s dnes již standardním rozšířením SSE. Kapitola pouze shrnuje nejdůležitější rysy nutné pro účely této práce, detailnější informace jsou k dispozici v citované literatuře.

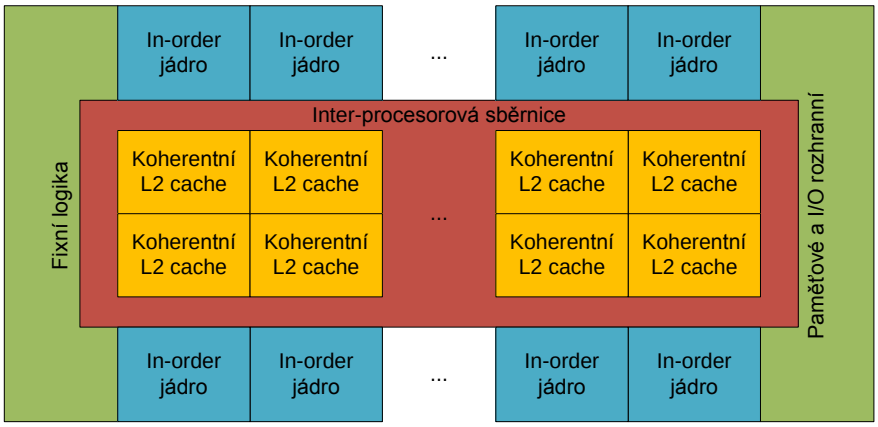
2.1 Proč Larrabee?

Pochopení toho proč existuje Larrabee může pomoci pochopit co je to Larrabee. Intel vyrábí dekádu od dekády rychlejší jedno-jádrové procesory zvyšováním frekvence hodin, zvyšováním velikosti cache pamětí a přidáváním tranzistorů navíc pro zvýšení počtu úkolů dokončených v každém jednom taktu hodin. Tento trend se určitě nezastaví a bude pokračovat jako vývoj hlavních procesorů i v budoucnu, ačkoliv to jde hůře a hůře. To je částečně způsobeno tím, že „většina nízko vysícího ovoce již byla sklizena“ a částečně tím, že procesory začínají dosahovat maximální hranice příkonu a oba přístupy, zpracování instrukcí mimo pořadí (out-of-order) i zvyšování frekvence hodin jsou výkonově náročné.

V poslední době však Intel přidává tranzistory jiným způsobem – přidáváním dalších jader. Tento přístup má velkou výhodu, že daný software může své úkoly provádět paralelně. Výkonnost tak může růst téměř lineárně s počtem jader integrovaných na čipu.

Larrabee dovádí tento přístup do jeho logického závěru. S mnoha jádry pracujícími s instrukcemi v pořadí daném překladačem (in-order) taktovanými na frekvenci zajišťující rovnováhu mezi výkonem a příkonem. Navíc, tato jádra jsou optimalizována spíše pro vícevláknové vektorové aplikace, než pro jedno-vláknové skalární aplikace. Hardwarová vlákna a vektorové jednotky dále rozšiřují výhody paralelizace. Toto všechno dovoluje Larrabee získat maximum práce z každého watu a každého čtverečního milimetru s výbornou rozšiřitelností do budoucna [1].

Larrabee je mnohem flexibilnější než současné GPU. Je to architektura podobná CPU založená na instrukční sadě x86 podporující podprogramy a výpadky stránek v paměti. Některé operace které GPU provádějí tradičně v blocích fixní logiky, jako je rasterizace a skládání (blending) výsledků pixel shader jednotek, jsou na Larrabee prováděny čistě softwarově. Stejně jako GPU používá Larrabee fixní logiku pro filtrování textur, ačkoliv jádra s touto logikou spolupracují a podporují ji například v případě výpadku stránky [7].



Obrázek 2.1: Blokové schéma architektury Larrabee. Počet a typ jednotlivých funkčních bloků je implementačně závislý.

[2] uvádí implementaci softwarového rasterizeru.

2.2 Hardwarová architektura

Larrabee je architektura se třemi rozdílnými aspekty – mnoho jader, mnoho hardwarových vláken a nové vektorové instrukce. Primárně je tato architektura určena pro GPU, ale může být použita také jako CPU [1].

Architektura Larrabee je založena na mnoha in-order CPU jádrech, která jsou doplněna novou vektorovou jednotkou (VPU). Jádra komunikují skrze inter-procesorovou sběrnici (inter-processor ring network) s vysokou propustností. Na sběrnici je připojena i fixní logika, jako jsou paměťová rozhraní, texturovací jednotky a další nutná logika závislá na aplikaci. Obrázek 2.1 ukazuje blokové schéma architektury. Každé jádro disponuje jeho vlastní částí L2 cache. To poskytuje vysokou propustnost přístupu k L2 každému z jader a zjednodušuje sdílení dat a synchronizaci.

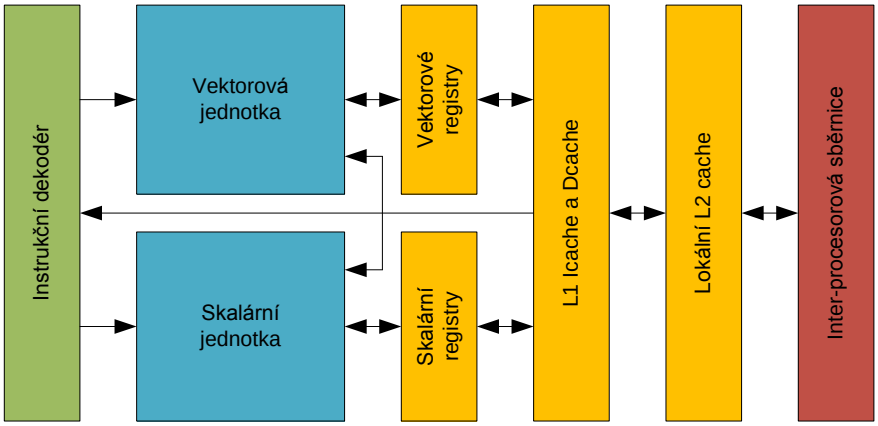
Data v tabulce 2.1 byla motivací k použití in-order jader s širokými vektorovými jednotkami. Prostřední sloupec ukazuje špičkový výkon moderních out-of-order CPU (konrétně Intel Core 2 Duo). Pravý sloupec ukazuje testovací CPU založené na procesoru Intel Pentium uvedeném v roce 1992. Jádro procesoru Pentium bylo modifikováno k podpoře čtyř hardwarových vláken a 512-bitové VPU [7].

Jádra a cache paměti Obrázek 2.2 ukazuje schématicky jedno jádro Larrabee plus jeho napojení na inter-procesorovou sběrnici a jeho L2 cache. Instrukční dekodér podporuje standardní instrukční sadu procesoru Pentium a další instrukce popsané v sekci LRBni.

Pro zjednodušení návrhu používají skalární a vektorová jednotka oddělené soubory registrů. Data přenášená mezi nimi musí být zapsána do paměti L1 cache a poté přečtena zpět. L1 cache se vyznačuje nízkou latencí přístupu. Je to v podstatě rozšířený soubor registrů. To značně vylepšuje výkonnost mnoha algoritmů. Larrabee disponuje 32kB instrukční cahce (Icache) a 32kB datové cache (Dcache) pro zajištění plynulého přístupu k paměti ve čtyřech hardwarových vláknech. Společná L2 cache je rozdělena na několik lokálních bloků, jeden pro každé jádro. Každé jádro disponuje rychlým přístupem k jeho vlastnímu bloku

Počet jader	2 out-of-order	10 in-order
Dokončovaných instrukcí	4 za takt	2 za takt
Šířka VPU	4 (SSE)	16
Velikost L2 cache	4MB	4MB
Skalární propustnost	4 za takt	2 za takt
Vektorová propustnost	8 za takt	160 za takt

Tabulka 2.1: Porovnání out-of-order a in-order jader. Navrhované jádro dosahuje pouze poloviční skalární propustnosti, ale 20x vyšší vektorové propustnosti s přibližně stejnou plochou na čipu a stejným příkonem. Tento rozdíl znamená 40x více FLOPS. Navíc VPU podporuje tzv. fused vynásob-a-sečti narozdíl od SSE (více v kapitole 2.4). Larrabee neobsahuje testované jádro, ale jemu podobné.



Obrázek 2.2: Jádro procesoru Larrabee a jeho napojení na inter-procesorovou sběrnici.

L2 cache. Inter-procesorová sběrnice zajišťuje koherentní přístup ke sdíleným datům. Data zapsaná jedním jádrem do L2 cache mohou být jiným jádrem zapsána do paměti. Každý blok L2 cache má 256kB. Larrabee také přidává nové instrukce a instrukční režimy pro explicitní načítání dat do cache pamětí. Například streamovaná data typicky přepisují data v paměti cache. Larrabee umožňuje označit každý řádek vyrovnávací paměti jako potenciální oběť při dalším přepisu poté co jsou data přečtena. Je to instrukční režim ve kterém se označování provádí automaticky v jednom cyklu se čtením. Instrukce pro explicitní načítání dat do paměti rovněž dovolují použití L2 cache jako tzv. scratchpad paměti [7].

Skalární jednotka Skalární pipeline je odvozena od pipeline procesoru Pentium umožňující dokončovat až dvě instrukce v jednom taktu. Pentium používá krátkou a „levnou“ pipeline. Larrabee přidává moderní techniky jako více vláken, 64-bitové rozšíření a sofistikovaný předvýběr. Jádra podporují plně instrukční sadu x86, proto mohou hostit existující programy včetně jader operačních systémů. LRBni přidává nové skalární instrukce jako bit-count a bit-scan.

Jádra disponující schopností dokončovat několik instrukcí v jednom taktu často ztrácí výkonnost kvůli hledání instrukcí, které mohou být spuštěny současně. Párovací pravidla pro první a druhou instrukční pipeline jsou u in-order jader deterministická, což dovoluje překladači provádět analýzu v širokém rozsahu během kompilace, to u out-of-order jader nelze. Protože druhá pipeline je relativně krátká a „levná“, jsou plocha a příkon promarněný neúspěšným provedením v každém cyklu malé.

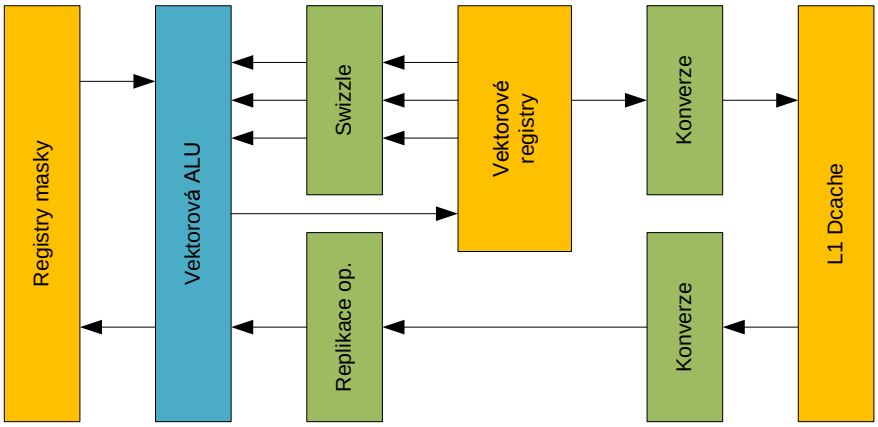
Larrabee obsahuje čtyři hardwarová vlákna s oddělenými registrovými soubory. Přepínání vláken je tak rychlé a může překlenout výpadek, kde není překladač schopen naplánovat vykonávání bez prostojů. Přepnutí vlákna může také překlenout výpadek v L1 cache [7].

Vektorová jednotka Vysokého výpočetního výkonu je dosaženo díky 512-bitové vektorové jednotce. Ta provádí operace nad 16-ti 32-bitovými celočíselnými operandy (int32), nad 16-ti 32-bitovými operandy v plovoucí řádové čárce s jednoduchou přesností (float), nad 8-mi 64-bitovými celočíselnými operandy (int64) a nad 8-mi 64-bitovými operandy v plovoucí řádové čárce s dvojitou přesností (double). Obrázek 2.3 ukazuje blokové schéma VPU s L1 cache.

VPU podporuje širokou škálu instrukcí pro práci jak s celočíselnými operandy tak s operandy v plovoucí řádové čárce. Instrukční sada poskytuje standardní aritmetické instrukce včetně instrukce MAC (multiply and accumulate) a standardních logických operací včetně extrakce bytově nezarovnaného pole. Toto všechno jsou operace typu load. Čtou operandy z registru nebo paměti a zapisují výsledek do vektorového registru. Další sada load-store instrukcí podporuje velké množství konverzí mezi plovoucí řádovou čárkou a datovými typy dostupnými na klasických GPU. Používání oddělených instrukcí pro každý datový typ šetří značnou část plochy čipu a příkonu za cenu malé ztráty výkonu.

Instrukční sada VPU obsahuje také instrukce typu gather a scatter. Jsou to load a store podporující čtení/zápis operandů z/do nespojitých bloků paměti. Místo načítání 16-ti bloků paměti z jedné adresy umožňuje načítat bloky až ze 16-ti různých adres specifikovaných v jiném vektorovém registru. Rychlost těchto operací je limitována rychlostí cache, která je schopná přistoupit v jednom cyklu pouze k jednomu řádku.

Tok programu může být ve VPU predikován pomocí maskovacích registrů (mask register), které mají vyhrazen jeden bit pro každý element vektorového registru. Maska říká která část vektorového registru nebo paměti bude zpracována a která zůstane nedotčena. Například skalární if-then-else struktura může být namapována do VPU nastavením masky



Obrázek 2.3: Vektorová jednotka. Podporuje tzv. swizzle registrových operandů, numerické konverze a replikace elementů načítaných z paměti. Registry masky dovolují predikovat zápis výsledků vektorových operací.

v závislosti na výsledku instrukce porovnání a provedením obou větví na základě masky. Větev může být vynechána úplně, pokud maskovací registr obsahuje samé nuly či samé jedničky. To snižuje penalizaci chybné predikce skoku a dává větší volnost překladači při plánování instrukcí [7].

Inter-procesorová sběrnice Larrabee používá pro komunikaci jader, L2 cache a dalších logických bloků obousměrnou kruhovou síť. Fyzická implementace sestává z několika propojených kruhů, proto kruhová síť.

Inter-procesorová sběrnice také poskytuje cestu pro přístup L2 cache do paměti. Typická implementace obsahuje několik rozhraní pro přístup do paměti rozmístěných na sběrnici z důvodu snížení pravděpodobnosti zahlcení [7].

Fixní logika Na inter-procesorové sběrnici jsou usazeny i další bloky fixní logiky. Po vzoru paměťových řadičů jsou rozmístěny na různých místech sběrnice.

Larrabee je vybaveno texturovacími jednotkami, protože tato operace nemůže být efektivně prováděna v jádrech. Vývojáři provedená analýza ukázala, že softwarové texturování na jádrech použitých v Larrabee by zabralo 12x až 40x déle než fixní logice (závisí na potřebě dekomprese textur). Tyto texturovací jednotky jsou interně srovnatelné s texturovacími jednotkami typických GPU. Obsahují 32kB cache na texturu pro každé jádro a podporují všechny běžné techniky jako jsou komprimované formáty DirectX 10, mipmapping, anizotropické filtrování, atd. Texturovací jednotky provádějí překlad virtuálních stránek na fyzické a výpadky hlásí CPU jádru, které dá texturovací jednotce pokyn k pokračování ihned po načtení stránky do cache paměti [7].

2.3 Programovací model

Programovací model nese označení Larrabee Native a vychází z dobře známého programovacího modelu více-jádrových architektur založených na x86. Stěžejním pro programování je kompletní C/C++ překladač, který staticky překládá programy do instrukční sady

Larrabee. Mnoho C/C++ aplikací může být úspěšně přeloženo a spuštěno na Larrabee bez jakýchkoliv modifikací. Existují však dvě omezení. Zaprvé, nejsou portována systémová volání aplikace. A zadruhé, aktuální architektura ovladačů vyžaduje znovu-přeložení aplikací. Dále jsou rozvedeny tři důležité aspekty programování aplikací pro Larrabee Native [7]:

Využívání softwarových vláken Larrabee Native prezentuje flexibilní způsob programování softwarových vláken. Schopnosti aplikačního rozhraní (API) na architektonické úrovni jsou srovnatelné se známým POSIX Threads API (P-threads) aplikačním rozhraním. Larrabee Native bylo rozšířeno o možnost definovat vývojáři přiřazení softwarového vlákna některému hardwarovému vláknu nebo jádru (affinity). Ačkoliv je P-threads mocný nástroj pro porgramování více-vláknových aplikací, může být cena vytváření a přepínání vláken pro některé aplikace příliš vysoká. Larrabee Native nabízí pro snížení této režie plánovač založený na algoritmu Distributed task stealing. Navíc je aplikační rozhraní vybaveno podporou OpenMP [7].

SIMD vektorizace Všechny SIMD (single instruction – multiple data) vektorové jednotky jsou plně programovatelné pomocí Larrabee Native. Překladač je vybaven vlastní verzí technologie auto-vectorization od Intelu. Vývojáři kteří potřebují programovat vektorové jednotky přímo, mohou využít sadu vestavěných nebo inline funkcí [7].

Komunikace mezi CPU a Larrabee V systému kde je Larrabee přítomno jako samostatná funkční jednotka, například ve formě přídavné karty, je nutné zajistit komunikaci mezi hostitelským procesorem (CPU) a Larrabee. Práce Larrabee je poté koordinována ovladačem operačního systému (OS) běžícího na hlavním procesoru. Knihovny Larrabee poskytují pro tuto formu spolupráce rychlý protokol pro zasílání zpráv nebo dat (message/data passing protocol). Protokol kontroluje veškerou komunikaci a přenosy dat mezi Larrabee a hlavním procesorem. Aplikační rozhraní podporuje jak synchronní tak asynchronní přenosy dat. Navíc, vykonání některých standardních knihovních funkcí jazyka C/C++ musí být v režii hostitelského porcesoru. Patří sem například vstupně-výstupní funkce read(), write(), open(), close(), atd. [7]

Podpora nepravidelných datových struktur Larrabee poskytuje podporu pro aplikace používající nepravidelné (irregular) datové struktury jako jsou komplexní stromové struktury, prostorové datové struktury, nebo velké rozptýlené n-dimenzionální matice. Podpora je zabudována v programovacím modelu, paměťové hierarchii a v instrukcích pro VPU. Narozdíl od stream-procesorů Larrabee dovoluje, ale nevyžaduje, přímé softwarové řízení načítání dat do paměti na různých úrovních hierarchie [8]. Software jednoduše čte nebo zapisuje adresy a hardware transparentně načítá data. Složitost se značně zjednodušuje a jádro může pracovat s těžce predikovatelnými nestrukturovanými přístupy do paměti. Důležitým aspektem práce nad nepravidelnými datovými strukturami je efektivní podpora operací typu gather-scatter. Tak může SIMD VPU pracovat s nespojitými bloky dat. Larrabee implementuje ve VPU gather-scatter instrukce, které načítají data do registru až z šestnácti různých adres v paměti cache bez penalizace přístupu. Sdílení paměti mezi jádry a funkčními jednotkami vyžaduje ochrané mechanizmy. Těmi jsou konvenční softwarové zámky, semafory a kritické sekce [7].

2.4 LRBni

LRBni přináší stejně jako rozšíření SSE (streaming SIMD extension) nové registry, nové instrukce pracující nad těmito registry, nové instrukce pro řízení cache paměti a předvýběr dat. LRBni navíc přidává několik nových instrukcí pro skalární jednotku. Narozdíl od SSE nabízí Larrabee snadnější práci s vektorovou jednotkou. Protože má každé hardwarové vlákno vlastní sadu registrů včetně vektorových, není nutné se zabývat ukládáním stavu jednotky při přepnutí kontextu. Také zaměření paměťové hierarchie spíše na spojité datové toky přináší značné urychlení, ačkoliv SSE jednotka má jisté možnosti ovlivnění obsahu cache paměti. Pro účely seznámení s instrukční sadou LRBni je možné rozdělit instrukce do čtyř skupin [1]:

- aritmetické a logické operace a posuvy,
- generování vektorové masky,
- vektorové load-store operace,
- ostatní instrukce.

LRBni přidává do architektury x86 dvě skupiny registrů. První skupinou je 32 512-bitových vektorových registrů, v0 až v31. Druhou skupinou je osm 16-bitových registrů masek, k0 až k7 [1].

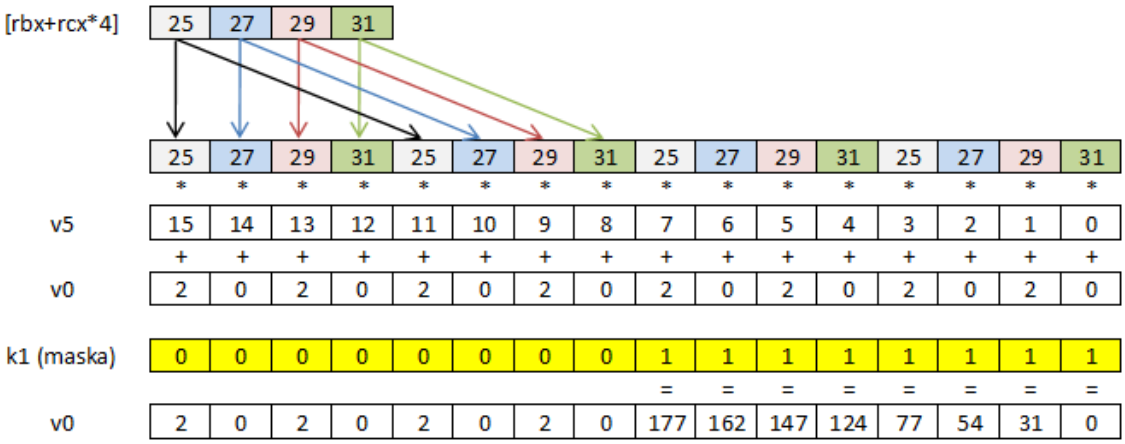
Na úvod pár konvencí. Vektorové instrukce začínají písmenem v, instrukce měnící registry masky začínají písmenem k. Přípony určují datový typ operandů stejně jako u rozšíření SSE. px říká, že se jedná o vektor 8-mi nebo 16-ti elementů, kde x určuje typ elementu. s pro float, i pro 32-bitový znaménkový int, u pro 32-bitový neznaménkový int, q pro 64-bitový znaménkový int a d pro double. Operace typu load-store používají pouze rozlišení délky elementu následovně, d pro 32-bitové elementy a q pro 64-bitové elementy. Typicky mají instrukce tři operandy, z nichž dva jsou vstupní a jeden výstupní. To umožňuje překladači lépe plánovat instrukce.

Například instrukce

$$vmadd231ps\ v0\ \{k1\}, v5, [rbx + rcx * 4]\ \{4to16\}$$

provede rozšíření čtyř floatů v paměti na adrese dané registry rbx a rcx na 16-ti prvkový vektor. Tento vektor vynásobí s vektorovým registrem v5, přičte v0 a na místech kde jsou v registru masky jedničky zapíše zpět do v0. A to vše pouze za cenu operace vynásob-a-sečti. Operaci popisuje názorně Obrázek 2.4. Notace 231 v mnemonickém kódu udává umístění registrů v operaci vynásob-a-sečti. Proto $madd231$ znamená: vynásob druhý operand se třetím a přičti první. [10] popisuje knihovnu zapsanou v jazyce C++, která implementuje instrukce rozšíření LRBni. Obsahuje navíc jejich detailní popis.

Aritmetické a logické operace a posuvy Aritmetické a logické operace a posuvy obsahují všechny očekávané instrukce: sčítání, odčítání, sčítání s přetečením, odčítání s výpujčkou, násobení, zaokrouhlení, zaokrouhlení se saturací, maximum, minimum, logický součet, logické násobení, logický exkluzivní součet, logický posuv a aritmetický posuv o různý počet bitů pro každý element a konverze mezi datovými typy včetně složených. Nabízí také množství operací typu MAC, jako vynásob-a-sečti nebo vynásob-a-odečti, které jsou stejně rychlé jako základní aritmetické operace. LRBni také poskytuje hardwarovou podporu pro vyšší



Obrázek 2.4: vmadd231ps v0 {k1}, v5, [rbx+rcx*4] {4to16}.

matematické výpočty a funkce. Všechny tyto instrukce pracují paralelně nad 8-mi nebo 16-ti elementy, ačkoliv ne všechny instrukce podporují všechny kombinace. Například většina vynásob-a-sečti operací nepodporuje int32 operandy. Důvod je však logický, operace typu MAC jsou implemntovány jako tzv. fused. To znamená že žádný bit operací v plovoucí řádové čárce není během výpočtu zaokrouhlen, proto jsou tyto operace někdy přesnější než ekvivalentně zapsaná posloupnost běžných aritmetických instrukcí. Navíc jsou provedeny rychleji.

Oba, cílový i první zdrojový operand musí být typicky vektorové registry (pro některé instrukce musí být první operand registr masky). Ale poslední operand může být paměťovým operandem. Tato možnost razantně snižuje počet instrukcí typu load a pozitivně ovlivňuje výkon. Instrukce vynásob-a-sečti a vynásob-a-odečti mají tři vektorové operandy, stejně jako většina aritmetických vektorových instrukcí, ale první z nich musí sloužit součastně jako zdrojový a cílový operand.

Pokud je zdrojovým operandem paměť, může být operand rozšířen (broadcast) z jednoho nebo čtyř elementů v paměti až na 16 elementů vektoru. Je-li tímto operandem vektorový registr, mohou být jeho elementy manipulovány (swizzle) stejně jako to umožňuje jazyk HLSL (High level shading language). Dále může každá vektorová instrukce provádět predikaci. Všechny registry masky obsahují 16 bitů, každý náležící jednomu elementu vektorového registru. Vektorová instrukce může dostat jako operand registr masky (writemask operand). Ten bude poté sloužit jako maska pro zápis, kde elementy mající v masce nulu zůstanou nezměněny. Všechny tyto operace jsou díky hardwarové podpoře zcela zdarma [1].

Generování vektorové masky Vektorová maska je generována především vektorovým porovnáním nebo kopií registru pro obecné použití rax, rbx, rcx, atp. Může být ale vytvořena instrukcemi přičti-a-generuj-přenos a odečti-a-generuj-výpujčku. Nebo z několika málo instrukcí navržených pro podporu rasterizace, přičti-a-nastav-masku-podle-znaménka. Registry vektorové masky mohou být také změněny sadou aritmeticko-logických instrukcí (například *kand*).

V některých případech je nutné ovlivnit běh programu pomocí příznaků příznakového registru procesoru (general processor flags). K tomu slouží instrukce *kortest*. Je to jediná vektorová instrukce měnící registr příznaků. Pokud jsou výsledkem logického součtu dvou registrů vektorových masek samé nuly je nastaven ZF (zero flag), pokud jsou výsledkem

samé jedničky je nastaven CF (carry flag) [1].

Vektorové load-store operace Larrabee umožňuje zarovnaný i nezarovnaný přístup do paměti. Stejně jako všechny aritmetické instrukce i operace load podporuje broadcast. Narozdíl od ostatních vektorových instrukcí podporuje množství typových konverzí známých z DirectX nebo OpenGL. Vektorový store zapisuje všech 16 elementů najednou, jen spodní čtveřici, nebo jen jeden nejspodnější element vektoru. Umí provádět typové konverze komplementární k těm poskytovaným operací load s několika málo grafickými rozšířeními, například pro sRGB (standardní RGB), vyžadující vlastní instrukci. K dispozici jsou i masky pro zápis a čtení pracující srovnatelně jako u aritmetických instrukcí.

Důležitou částí této podkapitoly jsou instrukce typu gather-scatter a způsob jakým adresují paměť. Adresy pro tyto operace jsou získány jako součet bázového registru a elementu vektorového registru znaménkově rozšířeného na 64 bitů. Spodní část adresy ve vektorovém registru může být volitelně násobena 2x, 4x, nebo 8x. Bez tohoto násobku by nebylo možné adresovat více než 4GB paměti. Operace gather a scatter podporují všechny typové konverze jako operace load a store. Stejně tak masky pro zápis a čtení. Naopak neumožňují broadcast, neboť je to pro tyto instrukce zbytečná operace.

Nakonec zmínka o instrukcích *vcompress* a *vexpand*. Slouží k efektivnímu ukládání elementů vektoru, které mají nastaven bit v registru masky, sekvenčně do paměti. Nemají-li tyto instrukce specifikovanou masku, chovají se stejně jako *vload* a *vstore* s nezarovnaným přístupem [1].

Ostatní instrukce Do této podkapitoly spadají všechny ostatní instrukce které přineslo rozšíření LRBni. Jmenovitě instrukce pro předvýběr dat a cachování popsané v kapitole 2.2, instrukce pro vkládání a prokládání dat na bitové úrovni. Dále byla rozšířena instrukce bsf respektive bsr (bit-scan). Instrukce totiž začala vždy vyhledávat od nultého bitu. Byla proto přidána instrukce $bsfi$ respektive $bsri$ začínající vyhledávání od bitu následujícího za specifikovaným cílovým operandem, což dovoluje pokračovat ve vyhledávání bez nutnosti přepisování operandu ve kterém se hledá.

Kapitola 3

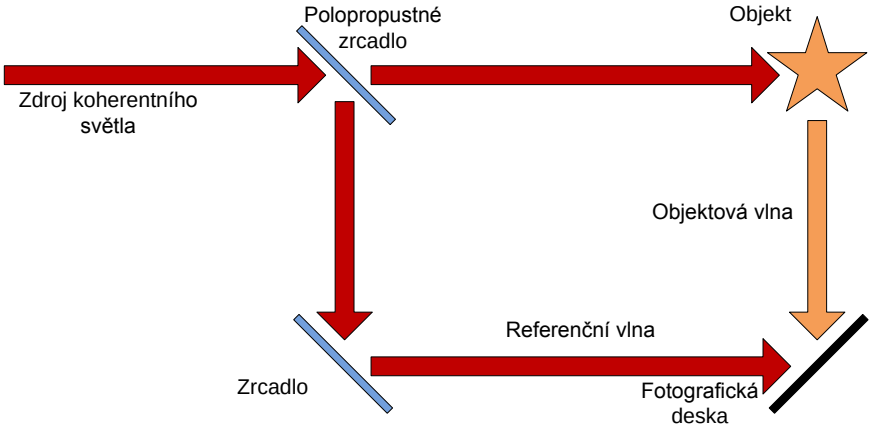
Návrh algoritmu

Shaderovací jazyky jako HLSL mají jistá omezení, která musí vývojáři dodržovat. Zato GPU zaručuje paralelní zpracování několika pixelů. Vektorizace je však mnohem obecnější technologií. To neplatí o MMX, SSE, nebo Altivec. Podporují vektorovou aritmetiku s tím rozdílem, že vektorová jednotka Larrabee poskytuje plnohodnotné vektorové funkční jednotky (ALU). Ostatní využívají pouze skalárních jednotek CPU a jejich zřetězení. Tyto technologie jsou navíc schopny efektivně pracovat pouze se spojitými oblastmi dat. Proto mohou být užitečné pouze nad přirozeně vektorovými daty jako RGBA barvy, XYZW souřadnice atp. Díky své architektuře a překladači s technologií auto-vectorization ja Larrabee vhodné pro mnohem širší použití [1][9].

3.1 Syntéza hologramu

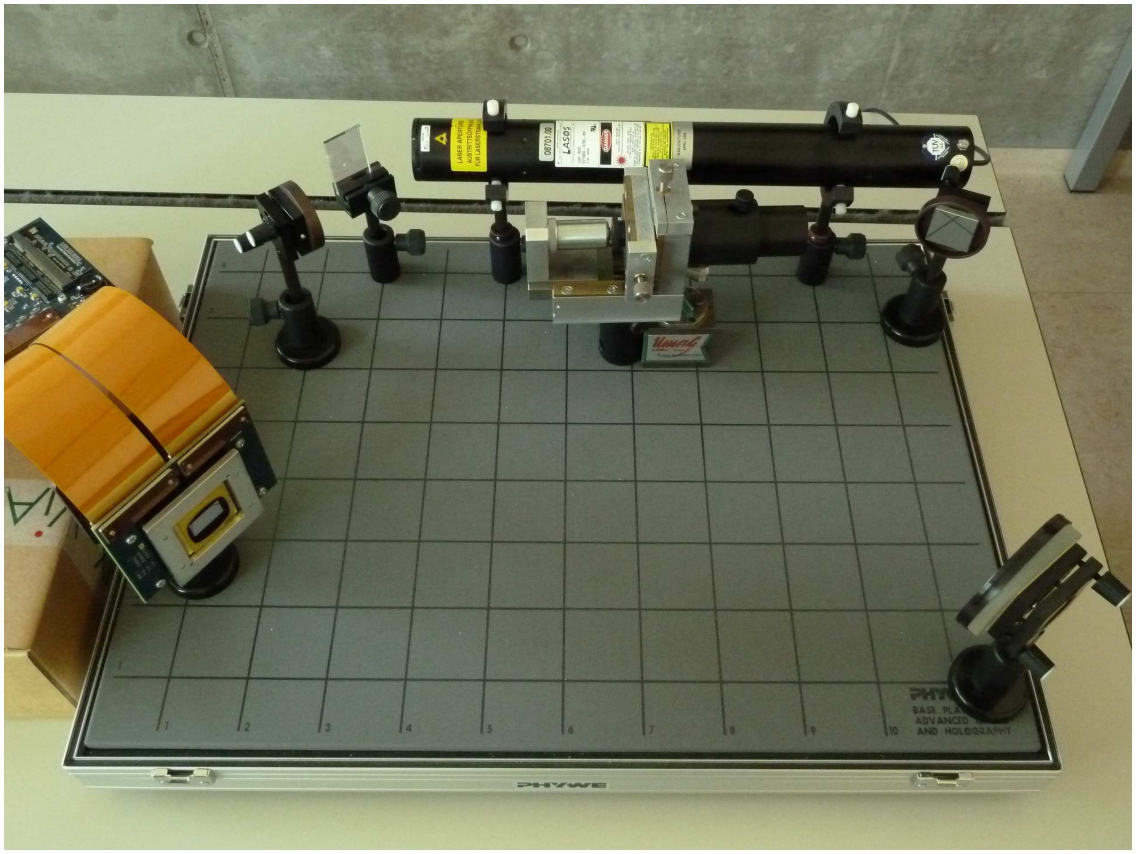
Holografie je docela stará vědní disciplína. Za zakladatele holografie může být považován prof. Dennis Gabor. Navrhl holografické zobrazení během práce na zvýšení rozlišení v elektronové mikroskopii [5]. Disciplína je založena na fyzikálních fenoménech difrakce a interference. Hologram je záznam optického pole generovaného scénou. Záznam má formu obrazu intenzity a interference mezi vlnami přicházejícími ze scény a referenční vlnou, která může být jednoduše reprodukována. Když referenční vlna ozáří hologram, zaznamenaná vlna je znovu vytvořena (rekonstruována). Hologram tak poskytuje pohled na scénu stejně jako skutečný svět a proto může být využit k zobrazení virtuální scény. Důležitou vlastností takového 3D pohledu je, že není omezen počtem pozorovatelů ani jejich polohou. To dělá tuto metodu ideální pro foto-realistické zobrazení. Díky matematickému popisu zúčastněných fyzikálních jevů může být tato metoda simulována numericky. Avšak, simulace je značně výpočetně náročná [3].

V jednoduchosti lze chápat hologram jako optické pole které obsahuje příspěvky objektových vln jdoucích ze scény. V současné době neexistuje metoda jak spočítat optické pole a simulovat tak fyzikální jev jako při záznamu hologramu v dostatečné kvalitě dostatečně rychle [6]. Mějme objekt který chceme zobrazit ve formě hologramu. Tento objekt musí být reprezentován miliony bodových světelných zdrojů (point light source)(tzv. point-cloud). Každý bodový světelný zdroj vyzařuje kulovou vlnu interferující se všemi ostatními. Cílem syntézy optického pole je akumulovat příspěvky všech světelných zdrojů v daném místě výsledného rastru. Nutno zmínit, že všechna čísla jsou komplexní protože je nutné uchovávat amplitudu i fázi. Parametry mřížky jsou závislé na použité metodě rekonstrukce hologramu. Práce předpokládá použití holografického displeje (obrázek 3.3) založeného na



Obrázek 3.1: Princip záznamu hologramu reálného objektu. Koherentní světelná vlna je rozdělena na objektovou vlnu, ozařující zaznamenávaný objekt, a referenční vlnu. Záznam je proveden na destičku z fotocitlivého materiálu.

Obrázek 3.2: Jednou by mohly být hologramy používány jako komunikační prostředek. Od toho nás však dělí mnoho technických otázek. Obrázek převzat od LucasArts ltd. Druhý obrázek ukazuje současné využití hologramů jako bezpečnostních prvků. Známka pravosti na bateriích mobilních telefonů značky Nokia. Převzato od Nokia Corp.

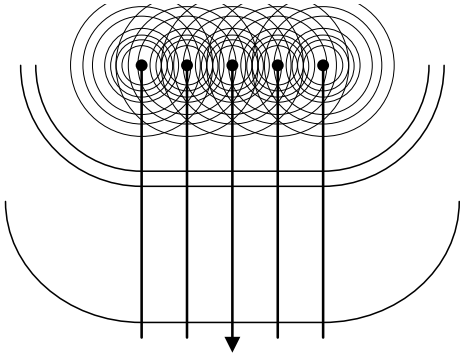


Obrázek 3.3: Holografický display vyvinutý na Fakultě Informačních Technologií Vysokého Učení Technického v Brně. Pro zobrazení je použit čip vyráběný společností Texas Instruments pro projektory. Čip je založen na technologii používající mikroskopická zrcátka o rozměru 10x10 mikrometrů v pravidelné matici o rozměrech 1920x1080 bodů. Obrázek převzat od [12].

mikroskopických zrcátcích vyvinutého na Fakultě Informačních Technologií Vysokého Učení Technického v Brně. Displej používá matici 1920x1080 optických elementů o velikosti 10x10 mikrometrů. Zrcátka po osvícení laserovým světlem vykreslují hologram [12].

Naivní přístup k syntéze optického pole je sečíst příspěvky všech bodových zdrojů pro každý bod rastru. Nicméně je zde několik dalších věcí ke zvážení. Shannon-Niquistův vzorkovací teorém je první. Krok mřížky není nekonečně malý, proto je k získání správných výsledků nutné dodržet vzorkovací teorém. Tam, kde je lokální frekvence větší než maximální reprezentovatelná, je nutné doplnit nuly. Druhou věcí je složitost zobrazovaného objektu. Ne všechny bodové zdroje musí být viditelné z aktuálně počítaného bodu. Takže je nutné pro každý příspěvek vyřešit viditelnost bodového zdroje. Z uvedeného je vidět, že složitost výpočtu se pohybuje v řádu $O(N^4)$.

Interference zmíněná výše přináší ještě jeden problém. Pokud se v blízkosti bodového zdroje vyskytne několik dalších v přibližně stejné vzdálenosti od optického pole se stejnou amplitudou a fází, vyzářená vlna bude plošná. Tuto situaci demonstruje obrázek 3.4 Pozorovatel proto uvidí otvor v objektu s bodovým světelným zdrojem v nekonečnu. K zabránění tomuto problému je nutné nahradit každý bodový zdroj množinou bodových zdrojů (tzv. patch) s definovaným výkonovým spektrem porušujícím podmínku vzniku. Cílem zbylé



Obrázek 3.4: Interference kulových ploch se stejnou amplitudou a fází.

části práce proto bude navrhnout a implementovat metodu výpočtu parametrů „patche“ tak, aby vyhovovaly danému spektru. To s sebou nese ještě jeden velký přínos. Aktuálně je možné zobrazit hologram s využitím pouze difuzního osvětlení. Tato metoda umožní využití i jiného osvětlovacího modelu než například Lambertova.

Definujme optické pole jako matici komplexních čísel $\hat{U}_{m,n}$, kde $m \in \langle -M, M-1 \rangle$ a $n \in \langle -N, N-1 \rangle$ leží v rovině $\kappa : z = 0$. $\hat{U}_{m,n}$ je bod rastru, jehož poloha v prostoru je $\vec{x}_{m,n} = (m\Delta_x, n\Delta_y, 0)$, kde Δ_x, Δ_y je krok mřížky rastru. Podle [11] může být příspěvek bodového světelného zdroje $\vec{x}_s = (x_s, x_y, z_s)$ do vzorku optického pole vyčíslen pomocí Rayleigh-Sommerfeldovi rovnice.

$$\hat{U}_{m,n} = \sum_{s \in S} \frac{\sqrt{I_s}}{r} \exp\left(\jmath \frac{2\pi}{\lambda} r\right) \cos\theta \tag{3.1}$$

$$r = \sqrt{(m\Delta_x - x_s)^2 + (n\Delta_y - y_s)^2 + z_s^2} \tag{3.2}$$

V této rovnici s zastupuje bodový světelný zdroj. Bodový zdroj má amplitudu a fázi reprezentovanou komplexním číslem $\sqrt{I_s}\exp(\jmath\varphi_s)$, kde I_s je intenzita a φ_s je počáteční fáze. θ je odchylka spojnice mezi bodovým zdrojem s a bodem rastru daným souřadnicemi $\vec{x}_{m,n}$ od normály roviny κ. λ značí vlnovou délku světla.

3.2 Definice problému

Abychom byli schopni vypočítat parametry – amplitudu a fázi – jednotlivých bodových zdrojů v „patchi“, je nutné si uvědomit netrivialitu této úlohy. Řešení vede na přeurčenou soustavu rovnic s komplexními čísly. V současné době není známo analytické řešení úlohy. Dokonce i přibližné řešení je považováno za úspěšné. V úvahu je nutné vzít i množství bodových zdrojů jejichž parametry hledáme. Zbylá část této kapitoly se tedy zabývá definicí problému.

Předpokládejme funkci $\hat{F}_{s,k}$ osvětlení bodu optického pole bodovým světelným zdrojem $s \in S$, kde $k \in K$ je linearizovaný index bodů optického pole (výše označeno m, n). Dále mějme parametry bodového zdroje $\hat{A}$ vyjádřené jako komplexní číslo

$$\hat{A}_s = \sqrt{I_s}\exp(\jmath\varphi_s). \tag{3.3}$$

Definujme osvětlení bodu k optického pole bodovým zdrojem s s amplitudou $\hat{A}$ jako

$$\hat{O}_{s,k} = \hat{A}_s \cdot \hat{F}_{s,k}. \tag{3.4}$$

Potom pro všechny bodové zdroje v „patchi“ platí

$$\hat{O}_k = \sum_{s \in S} \hat{O}_{s,k} = \sum_{s \in S} \hat{A}_s \cdot \hat{F}_{s,k}. \tag{3.5}$$

Nyní si definujme požadované osvětlení optického pole L. To jsou hodnoty kterých bychom v hologramu rádi dosáhli a mohou být dány například použitým osvětlovacím modelem. Odtud

$$L_k \approx |\hat{O}_k|. \tag{3.6}$$

L je v tomto případě reálné číslo, neboť na fotografické desce je také fyzicky uložena pouze amplituda světelné vlny. Fáze je v ní zakódována díky interferenci. Navíc pro rekonstrukci hologramu je to v součastnosti jediná možná reprezentace.

Předpokládáme-li platnost podmínky $|K| \gg |S|$, dostaneme předefinovanou soustavu rovnic kterou můžeme řešit pomocí minimalizace chyby e. Můžeme psát

$$L_k = |\hat{O}_k| + e_k, \tag{3.7}$$

kde

$$\sum_{k \in K} {e_k}^2 = min. \tag{3.8}$$

Můžeme odstranit nepříjemnou absolutní hodnotu a přepsat rovnici do tvaru

$$\hat{L}_k = \hat{O}_k \cdot \hat{P}_k + \hat{e}_k, \tag{3.9}$$

kde fáze $\hat{L}$ je nulová nebo zadaná. Objevila se zde však další proměnná $\hat{P}$ kompenzující odstraněnou absolutní hodnotu. Ta by měla být čistě imaginární ($\hat{P} = \exp(\jmath P)$) s amplitudou rovnou jedné. Podmínka minimalizace by pak měla tvar

$$\sum_{k \in K} \hat{e}_k \cdot \hat{e}_k^* = min. \tag{3.10}$$

V tomto bodě můžeme položit několik otázek, jejichž zodpovězení je nutné k nalezení řešení. Na tyto otázky se pokusí odpovědět kapitola zabývající se implementací.

- Která forma zápisu bude vhodnější pro nalezení řešení?
- Nebylo by možné zanedbat některé méně významné vlastnosti systému?
- Jaká jsou kritéria minimalizace chyby? Co znamená že chyba je minimální?
- Má tento problém analytické řešení?

3.3 Řešení na Larrabee

Z definice problému vyplývá, že výsledkem bude řešení přeurčené soustavy rovnic. Tato soustava čítá tisíce a tisíce rovnic a musí být řešena během výpočtu optického pole, tj. nelze ji předpočítat. Základním matematickým aparátem pro řešení přeurčené soustavy rovnic jsou matice a operace nad nimi. Nejčastěji používanými operacemi jsou násobení matic, transpozice a výpočet inverzní matice. Výpočet inverzní matice je na počítači nejčastěji řešen pomocí Gauss-Jordanovy eliminace s centrováním pivotu. Jedním z cílů akcelerace tedy

bude co možná nejrychlejší výpočet maticových operací. Zde by se dalo využít hned několika přístupů. Zaprvé, distribuovat části matic na jednotlivá jádra a počítat je paralelně. Protože pracujeme s komplexními čísly, daly by se použít pro výpočet matice reálných čísel takové, že matici koeficientů naplníme vlevo reálnými složkami komplexních čísel a vpravo imaginárními složkami. Stejně tak vektor výsledků a vektor pravých stran by měl dvojnásobný počet řádků. Tyto matice poté rozdělit na poloviny a počítat je paralelně na dvou jádrech. Tento postup používá například Matlab pro výpočet rovnic s komplexními čísly. Dalším želízkem v ohni jsou gather a scatter instrukce, které Larrabee nabízí. Ty velice pomohou při operacích které přistupují k prvkům matic z pohledu paměti takřka náhodně. Avšak je nutné dát pozor na přílišnou distribuci výpočtu jedné entity napříč čipem. Mohlo by se stát že komunikace nutná pro zajištění správných výsledků by byla tak velká, že by zcela překryla veškeré urychlení dosažené paralelizací. Samozřejmostí je využití vektorové jednotky při operacích jako je násobení řádkového vektoru skalárem a podobně.

Další úroveň paralelizace nabízí vysoké počty „patchů“ ve scéně. Těch je řádově v jednotkách až desítkách milionů. Nabízí se proto možnost počítat několik „patchů“ naráz. Výhodou v této situaci je to že jsou na sobě navzájem nezávislé. A proto jediným problémem je konečné složení optického pole z těchto dílčích částí. Kombinací obou přístupů je možné dospět až k variantě kdy je geometricky děleno přímo optické pole. K tomu je možné přistoupit z mnoha úhlů. Jedním je rozdělení optického pole do regulární mřížky. To s sebou však může přinést nepříjemnosti při opětovném skládání. Další variantou je rozdělit optické pole podle viditelnosti, zde by při dobré dekompozici mohlo zpětné skládání, dle mého úsudku, úplně odpadnout. Třetí možností je dělení prostoru podle hustoty bodových zdrojů, popřípadě „patchů“. Zde uvedené varianty nejsou dozajista jediné možné, ale jsou to základní osvědčené přístupy používané v jiných oblastech počítačové grafiky a zpracování obrazu.

Při vývoji bude k nezaplacení podpora standardu OpenMP. Ten dělá vývoj paralelních aplikací až neuvěřitelně jednoduchým. Nedokáže však všechno, bude tedy i tak nutné napsat některé základní operace pomocí instrukcí LRBni.

Bohužel až do této chvíle nebyla architektura Larrabee firmou Intel uvolněna a tak bude zbytek práce pokračovat s běžnou architekturou x86. Firma Intel nevydala k tomuto kroku žádné tiskové prohlášení. Jisté je tedy pouze to, že architektura byla vrácena zpět do vývoje – nebyla tedy prozatím zrušena.

Dnes se běžně při výpočtu optického pole používá patch s konstantní amplitudou a náhodně generovanou fází. Normální rozložení generátoru náhodných čísel zajistí přibližně difuzní vyzařovací charakteristiku. Cílem následující kapitoly tedy bude experimentovat s možnými vylepšeními této charakteristiky. V ideálním případě se mi povede najít metodu, která stanoví parametry „patche“ takové, že vyzařovací charakteristika bude mít definovaný tvar. Implementace se nebude soustředit na rychlost řešení, ale na jeho kvalitu a výpočetní složitost. Očekávám, že se podaří nalézt řešení zjednodušením rovnice 3.7, případně iteračním výpočtem. Toto řešení bude poté v páté kapitole konfrontováno s aktuálně používaným.

Kapitola 4

Implementace

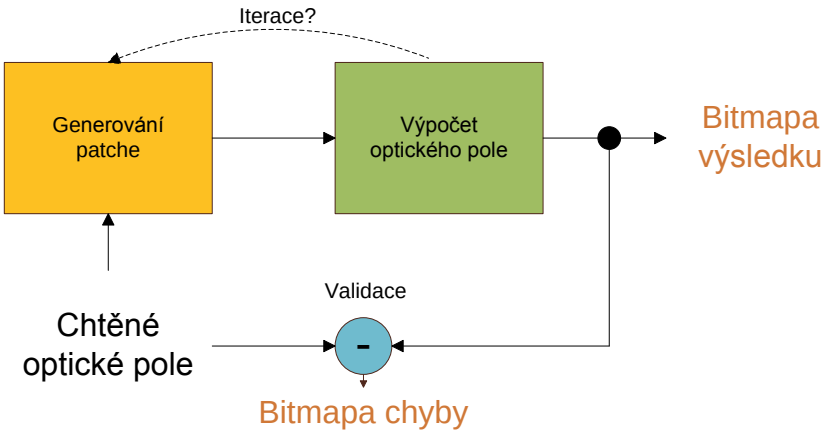
Předchozí kapitoly tvoří dostatečnou bázi znalostí aby bylo možné přejít k praktickému řešení problému. Kapitola se zabývá návrhem několika možných řešení. Odvozuje potřebné matematické vztahy a popisuje jejich implementaci. Je nutné tuto kapitolu chápat jako experimentální. Navržená řešení jsou nápady, které mohou nebo nemusejí vést ke korektním výsledkům. Pomocí experimentování s jednotlivými řešeními budou stanoveny jejich vlastnosti. V první části kapitola popisuje obecnoou architekturu demonstrační aplikace. Druhá podkapitola je věnována návrhu možných řešení spolu s jejich opodstatněním a očekávanými vlastnostmi. Zbytek této kapitoly je pak věnován jednotlivým řešením a experimentování s nimi.

4.1 Obecný náhled architektury aplikace

Demonstrační aplikace byla napsána v jazyce C++ s využitím objektového návrhu. Diagram tříd je uveden na obrázku A.1 v dodatku A. Aplikace se ovládá pomocí parametrů příkazového řádku. Ty jsou vypsány po spuštění s parametrem —*help*. Zdrojový kód obsahuje komentáře formátované pro generování programové dokumentace systémem Doxygen. Aplikace je nezávislá na cílové platformě. Součástí zdrojových kódů jsou proto Makefile a projekt pro VS2008.

Vzhledem k povaze problému je nutné pracovat s maticemi a maticovými operacemi. Tuto část práce obstarává knihovna pro práci s maticemi, vektory a quaterniony CwMtx. Knihovna je distribuována pod licencí Lesser GPL (LGPL). Zdrojové kódy knihovny jsou přiloženy k aplikaci včetně všech předepsaných dokumentů. Je napsána v jazyce C++ ve formě šablon. Většina operací je implementována pomocí přetížených operátorů. Výsledný kód je pak přirozeně čitelný a přehledný. CwMtx existuje ve dvou variantách. S dynamickými a statickými datovými strukturami. Autor na svých stránkách uvádí, že statická verze je třikrát až pětkrát rychlejší než verze s dynamickými datovými strukturami. Bohužel maximální velikost matice je pro tento případ příliš malá. Avšak předpokládám, že při velikostech matic nad deset tisíc prvků se režie dynamického přidělování paměti skryje za dobu výpočtu. Veškeré nadbytečné alokace a přesuny dat jsou v programu eliminovány.

Obrázek 4.1 ukazuje architekturu demonstrační aplikace. Základní filozofie zní následovně. Generátor (třída *Patch*) vygeneruje patch na základě vstupních parametrů. Těmito vstupními parametry jsou body optického pole ke kterým by se měl výsledek blížit. Vygenerovaný patch předá algoritmu pro výpočet optického pole (třída *Screen*), ta provede simulaci a výsledné optické pole může být konfrontováno s referenčním. Případně může být



Obrázek 4.1: Architektura demonstrační aplikace.

patch vrácen v případě iterativního algoritmu zpět k přegenerování. Výsledné optické pole je možné si uložit prostřednictvím třídy *Bitmap* do dvou obrázků, kde první reprezentuje reálnou složku a druhý imaginární složku optického pole. Také matici chyb je možné uložit.

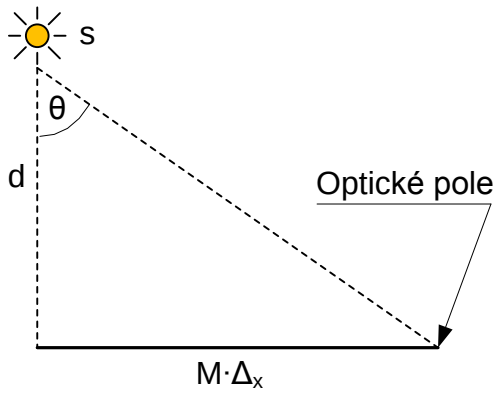
Pro začátek obsahuje třída *Patch* metodu generující „patche“ s konstantní amplitudou i fází a metodu, která vytvoří patch s pseudo-difuzní vyzařovací charakteristikou. Pseudo-difuzní protože fáze je v tomto případě generována náhodně generátorem s normálním rozložením pravděpodobnosti. Patch je v tomto případě reprezentován kolekcí bodových světelných zdrojů (Pls) s definovanou polohou, amplitudou a fází.

Jádrem třídy *Screen* je implementace rovnice 3.1 v metodě $Compute(patch)$. Ta spočítá odezvu daného patche na optickém poli. Tuto rovnici lze za určitých předpokladů zjednodušit. Zjednodušení neprodukuje žádnou pozorovatelnou chybu. Spočívá v položení základního předpokladu, že vzdálenost bodového zdroje od optického pole je velká v poměru k rozměrům optického pole. Pak úhel maximální odchylky θ je velice malý a kosinus tohoto úhlu se proto blíží k jedné. Hodnotu tohoto kosinu lze tedy zanedbat. Situaci ukazuje obrázek 4.2. Součástí této třídy jsou metody pro generování optického pole s požadovaným osvětlením. Je tu metoda umožňující nagenerovat konstantní optické pole – nic zajímavého. Druhou metodou je Phong-Blinnova osvětlovací funkce (obrázek 4.3). Ta vychází ze zjednodušeného Phongova osvětlovacího modelu, kde je spekulární složka vypočtena z odrazového vektoru (v obrázku označen R). Zatímco v Blinnově úpravě je spekulární složka počítána z takzvaného „half“ vektoru. To je vektor poloviční úhlu mezi vektorem směřujícím k bodovému zdroji a vektorem směřujícím k pozorovateli. Úprava tedy spočívá v upřednostnění výkonnosti před správností. Tento osvětlovací model dovoluje nasimulovat jak difuzní tak spekulární vyzařovací charakteristiku.

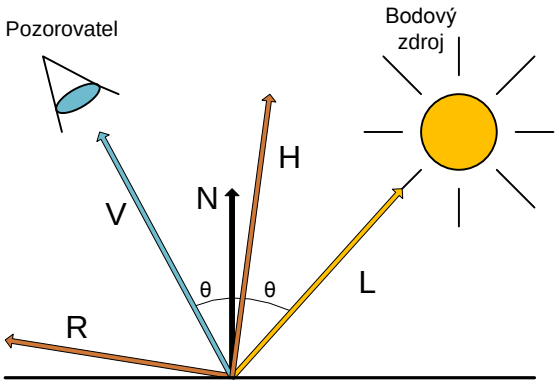
To jak se výsledné optické pole blíží k referenčnímu lze ověřit metodou $ErrorToBitmap()$. Metoda počítá chybu jako rozdíl čtverců amplitud jenotlivých bodů rastru

$$e_k = \hat{L}_k \cdot \hat{L}_k^* - \hat{O}_k \cdot \hat{O}_k^* = |\hat{L}_k|^2 - |\hat{O}_k|^2 \tag{4.1}$$

a výslednou chybu uloží do obrázku. Aby byl ve výsledných obrázcích vidět celý rozsah hodnot, jsou při ukládání všechny hodnoty normovány do intervalu $\langle 0; 1)$. Tím se mohou některé velmi malé informace (vzhledem k okolí) ztratit, proto je nutné používat obrázky pouze pro ilustraci a výpočty provádět vždy nad uvedenými datovými strukturami.



Obrázek 4.2: Při vzdálenosti bodového zdroje s od optického pole mnohem větší než je rozměr optického pole lze zanedbat působení kosinu v rovnici 3.1. Pro hodnoty $d = 0.5m$, $M = 1000$ a $\Delta_x = 10\mu m$ je tedy maximální úhel θ roven $0.02rad$, proto $\cos(\theta) \doteq 0.99998$.



Obrázek 4.3: Phong-Blinnův osvětlovací model.

Metoda	Klady	Zápory
Analytická (komplexní)	rychlá, přesná	neznámé řešení,
Best-fit	rychlá, jednoduchá implementace,	ne příliš přesná
Iterační	jednoduchá implementace, předem definovaná přesnost	rychlost závisí na počtu iterací a konvergenci, problém zajistit konvergenci

Tabulka 4.1: Vlastnosti navržených metod.

4.2 Implementovaná řešení

Při syntéze hologramu je dnes běžné používat „patche“ s konstantní amplitudou a fází generovanou náhodně. Při tomto postupu je spoléháno na normální rozložení generované posloupnosti a to, že výsledek bude mít přibližně difuzní vyzařovací charakteristiku. Logickým postupem by tedy bylo položit si otázku zda není možné fázi spočítat. A proč nakonec jen fázi? Nedával by patch lepší výsledky pokud by byla spočítána amplituda i fáze? Ve skutečnosti je výpočet amplitudy i fáze jako celku mnohem jednodušší než výpočet fáze při zadané amplitudě, neboť pak je nutné vyjádřit fázi komplexního čísla v závislosti na amplitudě a s touto závislostí počítat.

Analytické řešení definovaného problému není v dnešní době známo. Pokoušel jsem se o jeho nalezení, ale matematika a výpočetní metody v oblasti řešení soustav komplexních rovnic jsou netriviální a vyžadují značné matematické znalosti a především zkušenosti. Zkušenosti jsou však to co mi chybí. Podobné problémy se vyskytují v klimatologii, meteorologii, tektonice[1] a mnoha dalších geofyzikálních oborech. Článek [4] popisuje metodu CPCA (Complex Partial Component Analysis) používanou k identifikaci průchodu stojaté vlny geofyzikálními daty. Metoda je však značně náročná na porozumění a aplikaci.

Pokud připustím jistou chybu, lze analytické řešení zjednodušit. Výpočet může být proveden minimalizací chyby metodou nejmenších čtverců. Aby byl vektor chyby minimální, musí být kolmý na všechna $F_{s,k}$.

$$\forall s \quad \sum_{k \in K} (F_{s,k} \cdot e_k) = 0 \tag{4.2}$$

Položíme-li předpoklad, že $\hat{O}_k$ je čistě reálné, pak $|\hat{O}_k| = \Re(\hat{O}_k)$. Tímto krokem je odstraněna ona nepříjemná absolutní hodnota v rovnici 3.7. Stanovili jsme tak podmínku, že osvětlení je čistě reálné avšak fáze bodového zdroje může stále být nenulová. Při vývoji aplikace jsem postupoval tak, že jsem vyvinul program pro výpočet odezvy „patche“ na optickém poli. A tento patch jsem vypočítal v Matlabu. Později jsem řešení implementoval v metodě $BestFit()$.

Další možností je řešení iterační. Iterační řešení nevyžaduje detailní popis chování systému. Je však časově náročnější. Zpravidla záleží na počtu iterací které je nutné provést k dosažení výsledku. To je značně závislé na rychlosti konvergence metody. Zajištění konvergence je také nejtěžším úkolem při definici metody. Tabulka 4.1 shrnuje vlastnosti navržených metod. Následuje detailní popis implementovaných řešení.

[1] Vědní disciplína zabývající se pohybem zemské kůry.

4.3 Optimální výpočet amplitudy a fáze

Řešení vychází z předpokladu, že $\hat{O}_k$ je čistě reálné. Potom platí

$$|\hat{O}_k| = \Re(\hat{O}_k). \tag{4.3}$$

Analytické řešení se tím zjednoduší tak, že jej lze spočítat minimalizací chyby metodou nejmenších čtverců.

Mějme funkci

$$\hat{U}_{m,n} = \sum_{s \in S} \frac{\sqrt{I_s}}{r} \exp\left(\jmath \frac{2\pi}{\lambda} r + \varphi_s\right). \tag{4.4}$$

Ta je odvozena od rovnice 3.1 zanedbáním úhlu z důvodu definovaného v kapitole o obecné architektuře. Navíc se tu objevuje proměnná φ_s zastupující počáteční fázi bodového zdroje. Při bližším pohledu je zřejmé, že jde o funkci osvětlení bodu optického pole množinou S bodových světelných zdrojů, označenou $\hat{O}_k$. Je zde zastoupena osvětlovací funkce $\hat{F}_{s,k}$ a hledaná amplituda i fáze bodového zdroje. Nahradíme-li (m, n) linearizovaným indexem k, můžeme psát

$$\hat{O}_k = \hat{U}_k = \sum_{s \in S} \hat{F}_{s,k} \cdot \hat{A}_s. \tag{4.5}$$

Na tuto rovnici aplikujeme dříve zmíněný předpoklad a zapíšeme ji ve tvaru

$$\hat{O}_k = \Re(\hat{U}_k), \tag{4.6}$$

dosadíme rovnici 4.4, čímž získáme

$$\hat{O}_k = \Re\left(\sum_{s \in S} \frac{\sqrt{I_s}}{r} \exp\left(\jmath \frac{2\pi}{\lambda} r + \varphi_s\right)\right). \tag{4.7}$$

Vztah dosadíme do rovnice 3.7

$$L_k = \Re\left(\sum_{s \in S} \frac{\sqrt{I_s}}{r} \exp\left(\jmath \frac{2\pi}{\lambda} r + \varphi_s\right)\right) + e_k \tag{4.8}$$

a vidíme, že jsme se zbavili nepříjemné absolutní hodnoty o které jsem hovořil při definici problému. Vytknutím odstraníme sumu z vnitřku rovnice a použitím vzorce pro převod polární reprezentace komplexního čísla na obdelníkovou osamostatníme pouze reálnou složku. Tím získáme následující vztah

$$L_k = \sum_{s \in S} \frac{\sqrt{I_s}}{r} \cos\left(\frac{2\pi}{\lambda} r + \varphi_s\right) + e_k, \tag{4.9}$$

ze kterého vyjádříme matice koeficientů pro minimalizaci.

Pro minimalizaci použijeme lineární metodu nejmenších čtverců (označovaná LLS nebo někdy OLS[2]). Odvozovat ji zde nemá cenu. Je to známá metoda a dobře dokumentovaná například v nápovědě k Matlabu. Mnohem podstatnější je totiž způsob jakým metodu aplikujeme. Uveďme tedy pouze finální tvar řešení pro rovnici $Ax = b$, kde A je matice koeficientů, x je sloupcový vektor výsledků a b je sloupcový vektor pravých stran.

$$(A^T A)^{-1} A^T b = x \tag{4.10}$$

[2]Odrinary least square - značí že se jedná o neváhovanou variantu metody.

V našem případě bude řešení vypadat následovně:

$$\begin{bmatrix} \hat{f}_{1,1} & \hat{f}_{1,2} & \cdots & \hat{f}_{1,s} \\ \hat{f}_{2,1} & \hat{f}_{2,2} & \cdots & \hat{f}_{2,s} \\ \vdots & \vdots & \ddots & \vdots \\ \hat{f}_{k,1} & \hat{f}_{k,2} & \cdots & \hat{f}_{k,s} \end{bmatrix} \cdot \begin{bmatrix} \hat{a}_1 \\ \hat{a}_2 \\ \vdots \\ \hat{a}_s \end{bmatrix} = \begin{bmatrix} l_1 \\ l_2 \\ \vdots \\ l_k \end{bmatrix}$$

Prvky matice koeficientů dostaneme z rovnice 4.9, obsah ostatních dvou vektorů je zřejmý.

Navržené řešení jsem otestoval pomocí nástroje Matlab. Postupoval jsem tak že jsem si pomocí skriptu v Matlabu vygeneroval matice a spočítal parametry bodových zdrojů. Tyto parametry jsem uložil do textového souboru a přenesl je tak do progarmu napsaného v C++, pomocí něhož jesem spočítal a vykreslil odezvu na optickém poli. Později jsem implementoval metodu $BestFit()$, která řeší celý úkol. Tato metoda kopíruje následující osnovu:

1. vytvoř bodové zdroje a přiřaď každému pozici,
2. naplň matice,
3. proveď minimalizaci a
4. přepiš výsledky do bodových zdrojů.

Stanovení pozice bodového zdroje je nutné zajistit ještě před plněním matic neboť rovnice 3.2 ji používá k vyhodnocení vzdálenosti. Během výpočtů je nutné si dát pozor především na reprezentaci komplexních čísel. Pro některé části výpočtu je vhodnější reprezentace polární (Rayleigh-Sommerfeldova rovnice). Pro jiné je vhodnější reprezentace obdelníková (minimalizace).

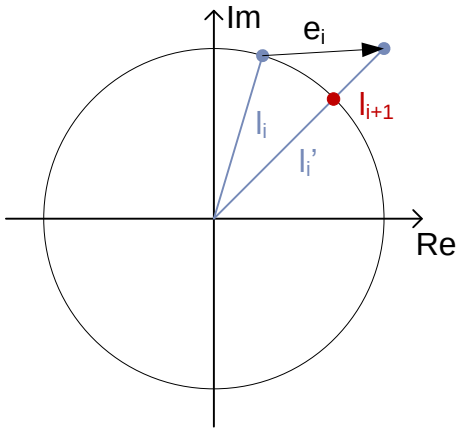
Experimentálně jsem implementoval metodu vzorkující optické pole s dvojnásobným krokem. Tento postup má výhodu v řešení poloviního počtu rovnic a vzhledem k poměru počtu bodových světelných zdrojů a počtu rovnic je korektním. Očekávám tedy stejně kvalitní řešení s nižší časovou náročností metody.

4.4 Výpočet amplitudy a fáze iterační metodou

Iterační výpočet je přesným opakem k analytickému. Není třeba rovnici dokonale popisující systém, stačí mít jednoduchou rovnici aproximující chování systému. Jedinou podmínkou je aby metoda konvergovala (tzn. našla uspokojivé řešení v konečném čase). Výhodou iteračního řešení je většinou jednodušší rovnice. Nevýhodou je délka výpočtu a nebezpečí že metoda nebude konvergovat nebo nebude konvergovat na dané datové sadě. Případně bude konvergovat pomalu.

Následujících několik kroků popisuje algoritmus implementovaného řešení.

1. Nageneruj cílové optické pole,
2. spočítej best-fit,
3. spočítej chybu; pokud je menší než zadaná maximální odchylka, přejdi na bod 6,
4. uprav rovnici,



Obrázek 4.4: Situace po výpočtu kroku iterační metody. Obrázek popisuje graficky vztah referenčního a vypočteného optického pole. $\hat{l}_i$ zde představuje referenční hodnotu, $\hat{l}'_i$ hodnotu vypočtenou, $\hat{e}_i$ představuje chybu řešení a $\hat{l}_{i+1}$ je hodnota zvolená pro další iteraci.

5. jdi na bod 2,

6. konec.

Z tohoto algoritmu je vidět že metoda použitá pro výpočet je stejná jako v předchozí kapitole. Zvolil jsem ji, protože je jednoduchou, přibližnou reprezentací systému. Dalším bodem algoritmu je výpočet chyby. Chyba je počítána jako suma čtverců odchylek jednotlivých bodů rastru optického pole. Princip je stejný jako v metodě $ErrorToBitmap()$. Výhodou tohoto postupu je stabilita a známý směr chyby. S dalším bodem se dostávám k úpravě rovnice. Je nutné si uvědomit, že tato úprava má zásadní vliv na konvergenci metody.

Obrázek 4.4 demonstruje situaci po výpočtu kroku iterační metody. Vystupuje zde referenční hodnota optického pole v daném bodě (to je vlastně vektor pravých stran použitý v metodě best-fit). Dále se zde objevila nějaká hodnota optického pole $\hat{l}'_i$. Tuto hodnotu získáme přičtením chyby vzniklé při minimalizaci k referenční hodnotě. Jednoduše je to hodnota odezvy nově vygenerovaného „patche“. Minimalizací chyby lze stanovit referenční hodnotu pro následující iteraci tak, že amplituda zůstane zachována a fáze se změní dle $\hat{l}'_i$. Tím se nová referenční hodnota dostane do červeně označeného bodu. Předcházející větu ilustrují následující rovnice.

$$|\hat{l}_{i+1}| = |\hat{l}_i| \tag{4.11}$$

$$\varphi(\hat{l}_{i+1}) = \varphi(\hat{l}'_i) \tag{4.12}$$

V době implementace to byl pouze nápad, nebylo jisté zda bude metoda konvergovat. Proto jsem do implementace zahrnul limit počtu průchodů metody. Později se ukázalo že nekonverguje, to je ale dle mého názoru způsobeno implementační chybou v metodě $BestFit()$. O tom však více v páté kapitole. Experimentálně jsem zkoušel implementovat různé druhy úprav rovnice. Jedním z nich byla změna fáze pouze o poloviční úhel než definuje rovnice 4.12. Další byla střídavá úprava fáze a amplitudy. Tyto metody však dosahovali spíše horších výsledků.

Kapitola 5

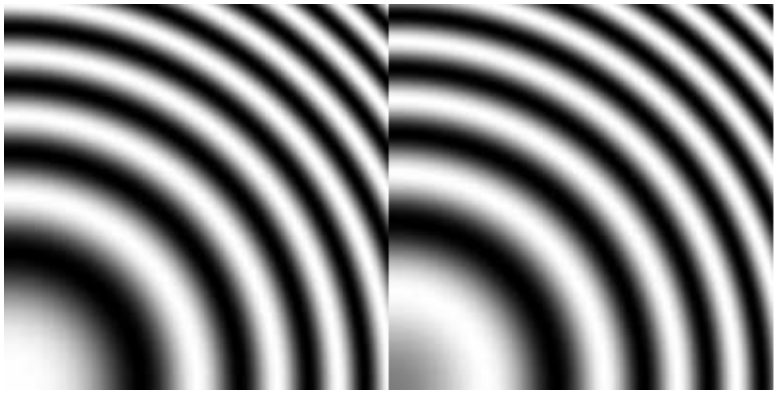
Výsledky

Tato kapitola uvádí výsledky kterých bylo dosaženo. Popisuje výstupy jednotlivých metod, diskutuje jejich přínos. Nejprve bude pro představu uvedena odezva bodového zdroje na optickém poli. Poté všechny tři implementované metody. První ukazuje současný stav – náhodně generovaný patch. Druhou metodou je best-fit, která dosahuje nad očekávání dobrých výsledků. Poslední je iterační metoda postižená zmíněnou chybou v metodě best-fit. Každý graf obsahuje legendu k reprezentaci barev.

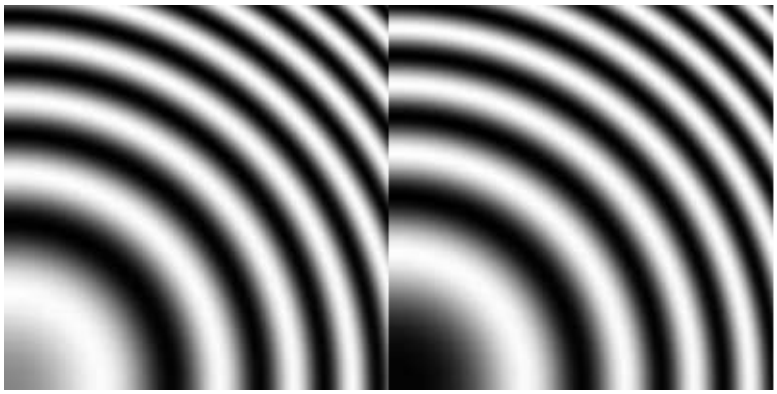
Odezva bodového zdroje Pro ilustraci uvádím odezvu bodového zdroje na optickém poli (obrázek 5.1). Bodový zdroj se nachází v levém dolním rohu obrázku. Data byla získána spuštěním programu patch s následujícími parametry: rozlišení optického pole 200x200 bodů, rozměr optického pole 2mm, vlnová délka světla 500nm, počet bodových zdrojů 1 a rozměr „patche“ 10nm.

Odezva náhodného „patche“ První implementovanou metodou je odezva „patche“ dnes běžně používaného při syntéze hologramu. Tento patch má jednotkovou amplitudu a fázi nagenerovanou náhodně. Tato metoda je rychlá a dává dobré výsledky pokud vyžadujeme difuzní vyzařovací charakteristiku. Patch je nad optickým polem opět umístěn v levém dolním rohu obrázku. Data byla získána spuštěním programu *patch* s následujícími parametry: rozlišení optického pole 200x200 bodů, rozměr optického pole 2mm, vlnová délka světla 500nm, počet bodových zdrojů 32x32 a rozměr „patche“ 10nm. Obrázek 5.2.

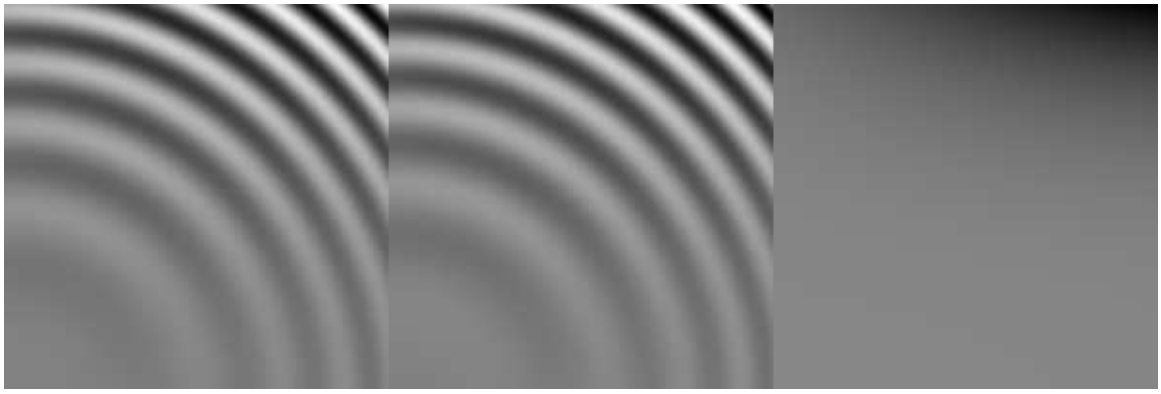
Metoda best-fit Druhou implementovanou metodou je best-fit, která analytickým řešením soustavy rovnic minimalizací nalezne patch s nejmenší chybou proti danému řešení. Dané řešení má v tomto případě spekulární vyzařovací charakteristiku a je získáno pomocí Phong-Blinnova osvětlovacího modelu. Světlo i pozorovatel jsou umístěni ve středu „patche“. Metoda je vzhledem k vykonávaným operacím relativně rychlá – řádově jednotky sekund. Nejdéle trvá zmíněná minimalizace neboť pracuje s maticemi o milionech prvků. Její složitost je přibližně $O(S \cdot K)$. V této a následující metodě je důležitý především obrázek reprezentující chybu. Ukazuje totiž jak je metoda kvalitní a výsledek stabilní. Výsledky této metody jsou nad očekávání dobré. Výsledná odchylka je malá, ačkoliv hodnoty výsledného optického pole jsou vysoké. To je dáno tím, že vygenerovaný patch má vysoké hodnoty amplitudy. Následující data byla získána spuštěním programu se stejnými argumenty jako předchozí metoda, pouze počet bodových zdrojů byl omezen kvůli paměťovým nárokům na 8x8. Obrázek 5.3.



Obrázek 5.1: Odezva bodového zdroje s parametry: rozlišení optického pole 200x200 bodů, rozměr optického pole 2mm, vlnová délka světla 500nm, počet bodových zdrojů 1 a rozměr patche 10nm. Vlevo reálná část (černá: -0.999999, bílá 0.999999), uprostřed imaginární část (černá: -1, bílá 1).

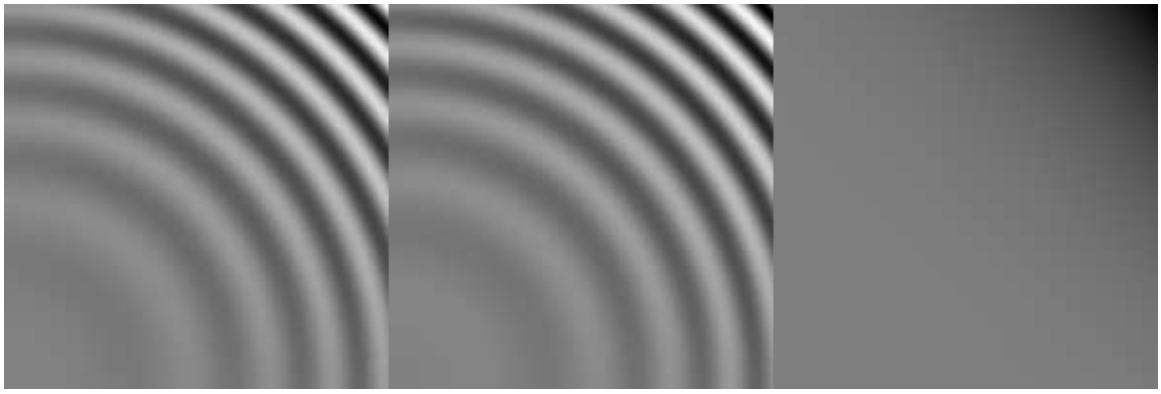


Obrázek 5.2: Odezva náhodného „patche“ s parametry: rozlišení optického pole 200x200 bodů, rozměr optického pole 2mm, vlnová délka světla 500nm, počet bodových zdrojů 32x32 a rozměr „patche“ 10nm. Vlevo reálná část (černá: -40.3884, bílá 40.3884), uprostřed imaginární část (černá: -40.3607, bílá 40.3607).



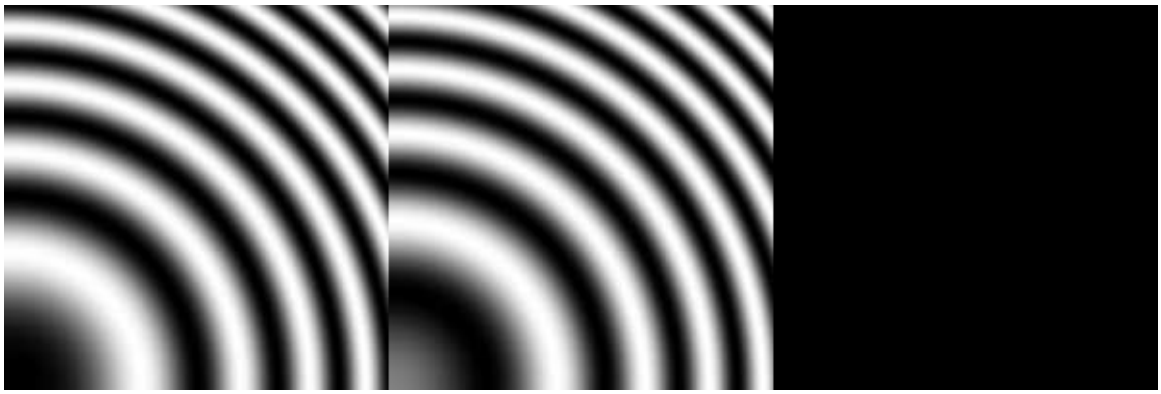
Obrázek 5.3: Metoda best-fit s parametry: rozlišení optického pole 200x200 bodů, rozměr optického pole 2mm, vlnová délka světla 500nm, počet bodových zdrojů 8x8 a rozměr „patche“ 10nm. Vlevo reálná část (černá: -0.0173285, bílá 0.0173285), uprostřed imaginární část (černá: -0.0195801, bílá 0.0195801), vpravo chyba (černá: -1.68997, bílá 1.68997).

Rád bych zmínil ještě experimentální úpravu metody best-fit s dvojitým krokem. Tento pokus se ukázal jako velice užitečný, neboť při zachování kvality řešení snížil výrazně datovou zátěž algoritmu a tím i čas řešení. Obrázek 5.4.



Obrázek 5.4: Metoda best-fit s parametry: dvojitý krok, rozlišení optického pole 200x200 bodů, rozměr optického pole 2mm, vlnová délka světla 500nm, počet bodových zdrojů 8x8 a rozměr „patche“ 10nm. Vlevo reálná část (černá: -27.6608, bílá 27.6608), uprostřed imaginární část (černá: -26.6789, bílá 26.6789), vpravo chyba (černá: -764.913, bílá 764.913).

Iterační metoda Poslední implementovaná metoda získává výsledné parametry bodových zdrojů iteračně. Metoda byla spuštěna se stejnými argumenty jako předchozí. Navíc byla přidána maximální přípustná odchylka ukončující itearční algoritmus. Ten se bohužel ukončí až na definovaném bezpečnostním limitu počtu iterací. Po experimentování s algoritmem jsem přišel na spojitost tohoto problému s uvedeným problémem metody best-fit. Amplitudy se totiž iterací pořád zvětšují a chyba tak roste exponenciálně. Předpokládám, že vyřešením tohoto problému by metoda měla bez problému konvergovat. Její složitost je také $O(S \cdot K)$, avšak doba řešení je závislá na počtu iterací. Obrázek 5.5.



Obrázek 5.5: Iterační metoda s parametry: rozlišení optického pole 200x200 bodů, rozměr optického pole 2mm, vlnová délka světla 500nm, počet bodových zdrojů 8x8 a rozměr „patche“ 10nm. Vlevo reálná část (černá: -4.92542e+008, bílá 4.92542e+008), uprostřed imaginární část (černá: -4.92539e+008, bílá 4.92539e+008), vpravo chyba (černá: -2.42597e+017, bílá 2.42597e+017).

Kapitola 6

Závěr

Cílem mé práce bylo představit architekturu Intel Larrabee a její přínos pro výpočty v oborech jako jsou počítačová grafika a zpracování obrazu či signálu. Larrabee je nová architektura založená na masivní paralelizaci jednoduchými jádry. Navíc přidává širokou vektorovou jednotku a instrukce pro práci s pamětí. Nejdůležitějším přínosem architektury jsou instrukce typu scatter a gather umožňující nezarovnaný přístup do paměti bez větší penalizace. Tím se také liší od jiných paralelních architektur. Bohužel Intel do data odevzdání práce neuvolnil architekturu Larrabee, naopak ji vrátil zpět do vývoje. K tomuto kroku společnost neposkytla žádné tiskové prohlášení. Pouze to, že projekt nebyl prozatím zrušen. Po dohodě s vedoucím práce jsem zvolil jako testovací algoritmus výpočet parametrů bodových světelných zdrojů pro syntézu hologramu. Tento úkol v sobě skrývá výzkumný charakter, neboť v dnešní době není uspokojivé řešení tohoto problému známo. Způsob jakým bych implementoval navržené algoritmy je diskutován ve třetí kapitole. Tím je splněn další bod zadání, ale dále už se v práci orientuji pouze na řešení problému jako takového. Po dlouhém tápání jsem definoval formálně problematiku stanovení amplitudy a fáze bodových zdrojů. A navrhl jsem tři možné metody. První je analytické řešení přeurčené soustavy komplexních rovnic. Řešení takovéto úlohy není však dnes známo. Druhou metodou je zjednodušení rovnice za cenu zanesení jisté chyby a její řešení minimalizací metodou nejmenších čtverců. A poslední metodou je iterační řešení daného problému. Poslední dvě jmenovaná řešení jsem implementoval a konfrontoval jejich výsledky s dnes běžně používaným řešením. Vlastní implementace je uvedena v kapitole 4. Pátá kapitola pak obsahuje výsledky a jejich hodnocení.

První navržená metoda poskytuje velice dobré výsledky. I přes chybu, kterou jsem zanesl do řešení zjednodušením rovnice, přináší dosud nemožné definování vyzařovací charakteristiky „patche“ bodových zdrojů. Iterační metoda v aktuální implementaci nekonverguje. To je však dle mého názoru způsobeno neodhalenou chybou v implementaci a tak je i tato metoda přes její časovou náročnost vhodným kandidátem k dalším experimentům.

Práce jako celek je dle mého názoru i přes nepříznivé vlivy uceleným pohledem na problematiku časově náročných výpočtů a jednu z nadějných architektur, která by mohla tomuto fenoménu pomoci. Druhá část práce pak uvádí čtenáře do tajů syntézy hologramu, která je silně závislá na architekturách jako je Larrabee. Zvláště se pak noří do jedné z dnes zatím nepříliš probádaných oblastí a přináší několik nápadů na řešení. Implementace potřebuje ještě doladit, ale metody zde jsou. V další práci bych rád vyřešil neduhy navržených metod. A soutředil se na nalezení dalších. Jak to tak bývá – rychlejších, přesnějších, atd. Jisté úsilý by také stálo zato věnovat nalezení analytického řešení. Ale rád bych se také podíval na další části procesu syntézy hologramu.

Literatura

[1] Abrash, M.: A First Look at the Larrabee New Instructions (LRBni) [online]. *Dr. Dobb's Journal*, 04 2009 [cit. 2010-01-03].
URL http://www.ddj.com/architect/216402188

[2] Abrash, M.: Rasterization on Larrabee [online]. *Dr. Dobb's Journal*, 05 2009 [cit. 2010-01-03].
URL http://www.ddj.com/architect/217200602

[3] Hanák, I., Herout, A., Zemčík, P.: Acceleration of the Detail Driven Method for Hologram Generation. *The Visual Computer*, ročník 26, č. 2, 02 2010: s. 83–96, ISSN 1432-2315.

[4] Horel, J. D.: Complex Principal Component Analysis: Theory and Examples. *Journal of Climate and Applied Meteorology*, ročník 23, č. 12, 1984: s. 1660–1673.
URL http://journals.ametsoc.org/doi/abs/10.1175/1520-0450%281984%29023%3C1660%3ACPCATA%3E2.0.CO%3B2

[5] Janda, M., Hanák, I., Skala, V.: Holography Principles. Technická zpráva, 12 2006.

[6] Janda, M., Hanák, I., Skala, V.: HPO hologram synthesis for full-parallax reconstruction setup. In *3DTV-CON 2007*, Piscataway: IEEE, 2007, ISBN 1-4244-0721-4, s. 1–4.

[7] Kolektiv autorů: Larrabee: A Many-Core x86 Architecture for Visual Computing. ACM Trans. Graph., 08 2008, str. 15.

[8] Kolektiv autorů: ON-CHIP MEMORY SYSTEM OPTIMIZATION DESIGN FOR THE FT64 SCIENTIFIC STREAM ACCELERATOR. IEEE Computer Society, 08 2008, str. 20.

[9] Kolektiv autorů: *Intel 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture*. 2009.

[10] Pitzel, S.: C++ Larrabee Prototype Library [online]. 06 2009 [cit. 2010-01-04].
URL
http://software.intel.com/en-us/articles/prototype-primitives-guide/

[11] Zemčík, P., Hanák, I.: Numerical Method for Accelerated Calculation of Point Light Source Optical Field [online]. Skala, V., Hildenbrand, D., 2009 [cit. 2010-05-14], str. 131.
URL http://gravisma.zcu.cz/GraVisMa-2009/Papers_2009/!_2009_GraVisMa_proceedings-FINAL.pdf

[12] Zemčík, P., Hanák, I., Kovář, L., Havel, J., Seeman, M.: Holographic display (specimen). 2009.

Dodatek A

Diagram tříd

Obrázek A.1: Diagram tříd demonstrační aplikace.

Dodatek B

Obsah CD

- bin – binární soubor pro platformu MS Windows
- cwmtx-0.6.2 – zdrojové soubory a licence knihovny pro maticové operace CwMtx
- doc – dokumentace k programu vygenerovaná systémem Doxygen
- examples – grafy použité v textu diplomové práce
- src – zdrojové soubory aplikace, Makefile a projektový soubor pro MS Visual Studio 2008, Doxyfile a readme.txt
- text – zdrojové soubory textu diplomové práce se vším co je potřeba k překladu a výsledné pdf