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

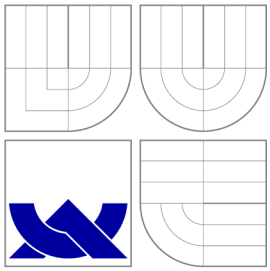
VÝPOČET OPTICKÉHO POLE PRO SYNTÉZU HOLOGRAMŮ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

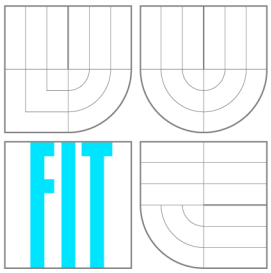
AUTOR PRÁCE
AUTHOR

MATÚŠ FEDORKO

BRNO 2013



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 PRO SYNTÉZU HOLOGRAMŮ

OPTICAL FIELD EVALUATIONS FOR HOLOGRAM SYNTHESIS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MATÚŠ FEDORKO

VEDOUCÍ PRÁCE
SUPERVISOR

prof. Dr. Ing. PAVEL ZEMČÍK

BRNO 2013

Abstrakt

Přimárním cílem této práce je počítačem generovaná holografie, konkrétněji urychlení počítaní syntetických hologramů prostředkami dnešních masivně paraleních grafických čipů. Práce začíná vysvělením základních holografických principů spolu s ich matematickým a fyzikálním pozadím. Následující diskuze pak pojednává o OpenCL, které je relativně novým, ale přesto všeobecně uznávaným standardem programování GP-GPU aplikací. Poslední dvě kapitole této práce pak popisují návrh a implementaci problému v C++.

Abstract

The primary concern of this work is computer generated holography, in particular accelerating computation of synthetic holograms by means of modern-day massively parallel graphics chips. It starts by introducing fundamental holography principles together with mathematical and physical concepts behind it. Following discussion then deals with OpenCL, which is a relatively new, but well recognized standard for GP-GPU programming. Finally, the last two chapters of this work give an overview of problem analysis and its implementation in C++.

Klíčová slova

holografie, hologram, optické pole, difrakce, interference, OpenCL, GPU

Keywords

holography, hologram, optical field, diffraction, interference, OpenCL, GPU

Citace

Matúš Fedorko: Výpočet optického pole pro syntézu
hologramů, bakalářská práce, Brno, FIT VUT v Brně, 2013

Výpočet optického pole pro syntézu hologramů

Prohlášení

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

.....
Matúš Fedorko
14. května 2013

Poděkování

Rád bych zde poděkoval vedoucímu práce prof. Dr. Ing. Pavlovi Zemčíkovi za odborné vedení a pomoc při její tvorbě. Poďekování patří i Ing. Petrovi Lobazovi ze Západočeské Univerzity v Plzni za poskytnutí studijích materiálů a ochotu při zodpovídaní dotazů ohledně holografie. Nakonec bych se ještě chtěl poděkovat svým rodičům za vytrvalou podporu při psaní práce.

Obsah

Kapitola 1

Úvod

S neustále rastúcou úrovňou dnešných technológií a stále dokonalejšou schopnosťou modelovať okolitú realitu prostredníctvom počítača rastú aj nároky jej konzumentov. To čo stačilo ľudom, keď prišli prvé pohyblivé obrázky dnes už zaujme snáď len nadšencov. Je preto logické, že ľudstvo sa nepretržite snaží prichádzať s novými postupmi a nápadmi ako prehĺbiť virtuálne zážitky, pričom jedným z týchto smerov je aj 3D zobrazovanie.

Práve tu prichádza do hry holografia, ktorá oproti súčasným riešeniam predstavuje výrazný skok vpred. Skutočnosť že dnes potrebujeme rôzne dodatočné vybavenie v podobe 3D okuliarov, či iných pomôcok, aby sme oklamali naše zmysly v holografii neplatí. Tá totiž poskytuje pravé trojdimenzionálne zobrazenie spolu so všetkými jeho spletitými detailami, presne tak ako by sme to intuitívne očakávali.

Hoci sú teda matematické a fyzikálne princípy holografie celkom dobre známe, stále nie je k dispozícii uspokojivé počítačové riešenie, ktoré by zvládlo vytvárať aj komplexné hologramy v rozumnom čase. To je dané najmä povahou potrebných výpočtov vychádzajúcou zo zložitosti simulácie svetelných javov.

V priebehu rokov tak bolo navrhnutých množstvo viac, či menej úspešných riešení, ktorých snahou bolo posunúť vývoj v tejto oblasti dopredu. Tým z nich ktorým, sa podarilo dosiahnuť výrazného zlepšenia rýchlosti výpočtu, sa však väčšinou nepodarilo zachovať aj výslednú kvalitu záznamu a naopak. Cieľom práce je teda pomocou dostupných technológií pre výpočty na grafickej karte optimalizovať vybranú metódu generovania hologramov.

V prvej kapitole budú predstavené základné princípy, na ktorých je holografia založená. Výklad začne vysvetlením fyzikálnych javov umožnujúcich vznik hologramu, z ktorého následne prejde do popisu optických spôsobov záznamu a na záver budú vysvetlené aj princípy digitálnej holografie a vybraných syntetizačných metód. V druhej kapitole budú popísané základné vlastnosti OpenCL, čo je štandard a zároveň aj programová knižnica umožňujúca využiť vysoký paralelný výkon dnešných grafických kariet. Kapitoly o návrhu a implementácii zas predstavia zvolené riešenie a spôsob jeho realizácie spolu s testovaním výsledkov. Na záver budú poslednou kapitolou zosumarizované získané poznatky a načtrnuté možné rozšírenia práce v budúcnosti.

Kapitola 2

Holografia

História holografie sa začala písať už v roku 1947, keď jej téoriu sformuloval britský fyzik maďarského pôvodu Denis Gabor počas svojich pokusov na vylepšenie elektrónovej mikroskopie. Prvým zostrojeným typom hologramu bol tzv. "inline hologram", ktorý však neposkytoval dostatočne kvalitné výsledky a mal rad významných obmedzení. Skutočný posun v holografickom zaznamenávaní nastal až v roku 1962 s vynájdením laseru a zostrojením "off-axis hologramu"americkými vedcami Emmetom Leithom a Jurisom Upatnieksom. Obe tieto konfigurácie sú podrobne opísané v sekcii *Optická holografia*.

Hoci myšlienka počítačom vytvoreného hologramu nie je ničím novým, jej realizácia kvôli nedostatočnému výkonu pomerne dlho stagnovala. To ako sme na tom dnes a aké problémy je potrebné v tejto oblasti vyriešiť ďalej rozvádza sekcia *Digitálna holografia*, v ktorej sú okrem iného naznačené aj možné postupy ich generovania.

A keďže holografia je založená na pomerne komplikovaných fyzikálnych zákonoch, tak prvá časť tejto kapitoly slúži ako stručný úvod k pochopeniu tých najnutnejších princípov.

Informácie v nasledujúcom texte boli čerpané hlavne z [8], [14] a [10].

2.1 Základné fyzikálne princípy

2.1.1 Svetlo a vlnenie

Svetlo sa dá definovať ako elektromagnetická vlna zložená z časovo premenného elektrického a magnetického poľa. Vzájomný vzťah týchto dvoch polí opisujú Maxwellove rovnice a za určitých podmienok môžu byť vyjadrené rovnicou skalárnej vlny:

$$\bigtriangledown^2 u(\mathbf{p},t) - \frac{n^2}{c^2}\frac{\partial^2 u(\mathbf{p},t)}{\partial t^2} = 0 \tag{2.1}$$

kde $u(\mathbf{p},t)$ reprezentuje zložku skalárneho poľa na pozícii $\mathbf{p}$ v čase t, c reprezentuje rýchlosť šírenia svetla vo vákuu a n index lomu materiálu, ktorým sa vlnenie šíri.

Pre holografiu je dôležitým typom svetla tzv. monochromatické, alebo jednofarebné svetlo, ktoré sa skladá len z vĺn jednej vlnovej dĺžky. Skalárne pole monochromatickej vlny sa dá vyjadriť rovnicou:

$$u(\mathbf{p},t) = A(\mathbf{p})cos[2\pi\nu t - \phi(\mathbf{p})] \tag{2.2}$$

kde $A(\mathbf{p})$ predstavuje amplitúdu a $\phi(\mathbf{p})$ fázu vlny v bode P a čase t. Symbolom ν sa označuje optická frekvencia vlny počítaná v jednotkách Hz. Kvôli jednoduchšej práci so vzorcami sa však predošlá rovnica veľmi často prepisuje do exponenciálneho tvaru:

$$u(\mathbf{p},t) = \Re\{\tilde{u}(\mathbf{p})exp(-i2\pi\nu t)\} = \Re\{\tilde{u}(\mathbf{p})exp(-i\omega t)\} \tag{2.3}$$

pričom ω je uhlová rýchlosť a notácia $\Re\{\}$ označuje, že za výsledok budeme považovať iba reálnu časť komplexného čísla v zátvorkách. Člen $\tilde{u}(\mathbf{p})$ sa nazýva komplexná amplitúda a jej definíciu ukazuje nasledujúci vzorec:

$$\tilde{u}(\mathbf{p}) = A(\mathbf{p})exp[i\phi(\mathbf{p})], \tag{2.4}$$

ktorého výsledkom je komplexné číslo nesúce informáciu o amplitúde aj fáze svetla a celá funkcia $u(\mathbf{p}, t)$ nesie názov vlnová funkcia.

2.1.2 Interferencia

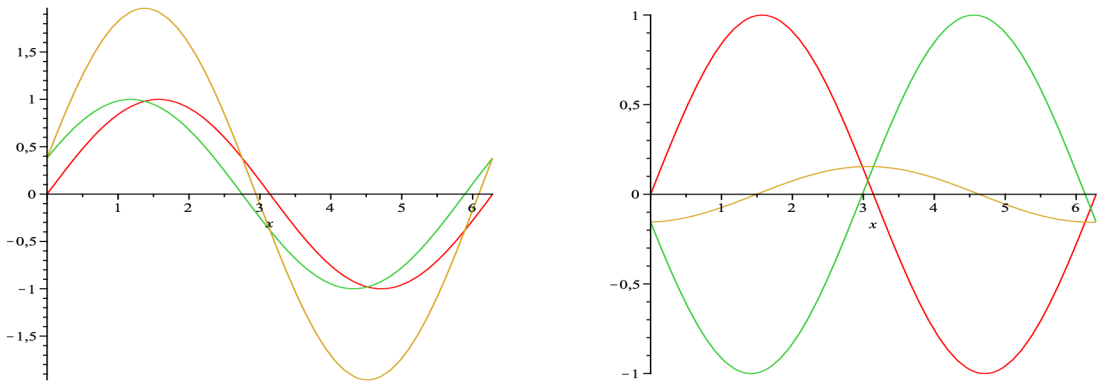
Interferencia je fyzikálny jav, ktorý vzniká pri superpozícii dvoch alebo viacerých vĺn. Tie sa môžu v závislosti od fázy buď vzájomne zosilovať, vtedy hovoríme o konštruktívnej interferencii, alebo sa môžu naopak vzájomne rušiť a vtedy hovoríme o deštruktívnej interferencii. Za predpokladu, že interferujúce vlny sú koherentné platí nasledujúci vzťah pre ich komplexné amplitúdy:

$$\tilde{u} = \tilde{u}_1 + \tilde{u}_2 + ... + \tilde{u}_n \tag{2.5}$$

Toto pôsobenie má zároveň vplyv aj na optickú intenzitu. Pri konštruktívnej interferencii sa intenzita zvyšuje, pri deštruktívnej interferencii sa zas znižuje. Výpočet intenzity pre dve vlny demonštruje tento výpočet:

$$\begin{aligned} I =& |\tilde{u}_1 + \tilde{u}_2|^2 \\ =& |\tilde{u}_1 + \tilde{u}_2||\tilde{u}_1 + \tilde{u}_2|^* \\ =& |\tilde{u}_1|^2 + |\tilde{u}_2|^2 + u_1 u_2^* + u_1^* u_2 \\ =& I_1 + I_2 + 2\sqrt{I_1 I_2}cos(\phi_1 - \phi_2) \end{aligned} \tag{2.6}$$

kde ϕ_1 a ϕ_2 predstavujú fázy vĺn $\tilde{u}_1$ a $\tilde{u}_2$. Štruktúra vytvorená takto vypočítanou intenzitou sa potom nazýva interferenčný vzor, čiže hologram.



Obrázek 2.1: Konštruktívna (vľavo) a deštruktívna (vpravo) interferencia. Žltou farbou je znázornená výsledná interferovaná vlna.

2.1.3 Koherencia

Ďalším veľmi dôležitým fyzikálnym javom, ktorý úzko súvisí s interferenciou je koherencia. Tá v podstate určuje ako dobre budú viditeľné jednotlivé interferenčné vzory. Za predpokladu, že existujú dva ideálne zdroje koherentného svetla sa táto viditeľnosť dá popísať vzorcom:

$$V = \frac{2(I_1 I_2)^{1/2}}{I_1 + I_2} cos(\psi) \tag{2.7}$$

kde I_1 a I_2 je intenzita prvej a druhej vlny a ψ je uhol medzi elektrickými vektormi týchto vĺn, ktorý zároveň predstavuje polarizáciu. Svetlo z koherentného svetelného zdroja produkuje jasne viditeľný interferenčný vzor, svetlo z nekoherentného zdroja naopak nevytvorí žiaden viditeľný vzor.

Existujú dva druhy koherencie, a to koherencia časová a koherencia priestorová. Prvý typ určuje ako veľmi je svetlo monochromatické, zatiaľ čo druhý vyjadruje schopnosť laseru zaostriť na veľmi malú plochu.

V skutočnosti ale neexistuje dokonale monochromatické svetlo, ktorého vlny by mali práve jednu presne stanovenú vlnovú dĺžku. V reálnom svete je k tomuto ideálu najbližšie laserové svetlo, ktoré obsahuje len veľmi úzky rozsah frekvencií.

Veľmi dôležitým dôsledkom svetla, ktoré je časovo aj priestorovo koherentné je fakt, že môžme zanedbať časovú zložku $-i\omega t$ z vlnovej rovnice 2.3 bez straty presnosti. K postačujúcemu popisu monochromatického svetla teda stačí len samotná komplexná amplitúda.

2.1.4 Elementárne vlny

Vlnenie svetla sa dá zredukovať na dva elementárne prípady, a to na rovinnú a na sférickú vlnu.

Vlnoplochy rovinnej vlny sú tvorené paralelnými nekonečne veľkými rovinami, zatiaľ čo o vlnoplochách sférickej vlny sa dá uvažovať ako o koncentrických guľových plochách vychádzajúcich z jedného nekonečne malého bodu alebo otvoru.

Komplexná amplitúda rovinnej môže byť zapísaná rovnicou:

$$\tilde{u}(\mathbf{r}) = \tilde{a}\, exp(i\mathbf{k}\cdot\mathbf{r}) = \tilde{a}\, exp[i(k_x x + k_y y + k_z z)] \tag{2.8}$$

kde $\mathbf{k} = (k_x, k_y, k_z)$ predstavuje vlnový vektor, ktorého absolútna hodnota $|\mathbf{k}|$ je rovná vlnovému číslu k a $\mathbf{r}$ reprezentuje polohový vektor určujúci bod v priestore, pre ktorý sa daná vlna počíta. Intenzita takejto rovinnej vlny je konštantná na celej rovine vlnoplochy s hodnotou $I = |\tilde{a}|^2$.

Na druhej strane komplexná amplitúda sférickej vlny sa dá vyjadriť vzťahom:

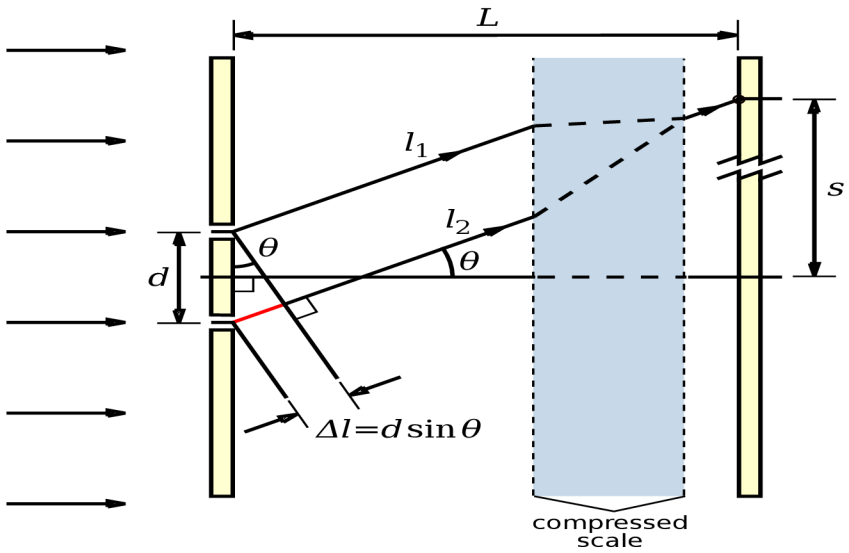
$$\tilde{u}(\mathbf{r}) = \frac{\tilde{a}}{r} exp(ikr) \tag{2.9}$$

pričom zlomok $\tilde{a}/r$ odráža skutočnosť, že s rastúcou vzdialenosťou od zdroja, vyjadrenou členom $r = |\mathbf{r}|$, klesá aj intenzita vlny. Zároveň ak je táto vzdialenosť relatívne veľká voči ploche, na ktorej sférickú vlnu skúmame, môžme túto veľmi dobre aproximovať vlnou rovinnou.

2.1.5 Difrakcia

Je to fyzikálny jav, počas ktorého sa vlnenie, a teda aj svetlo, vplyvom ohybu dostáva do oblasti geometrického tieňa za prekážkou. Takéto ohýbanie môže byť spôsobené buď prechodom veľmi malým otvorom porovnateľným s vlnovou dĺžkou, alebo prechodom okolo hrany

prekážky. Obrázok 2.2 demonštruje prípad difrakcie pri prechode svetla dvoma štrbinami na nekonečne veľkom tienidle.



Obrázek 2.2: Difrakcia svetla na dvoch štrbinách. Obrázok prebraný z Wikipédie [1].

Vo vzdialenosti L, ktorá je omnoho väčšia ako vzdialenosť d medzi dierkami dôjde k interferencii vĺn l_1 a l_2. To, či dôjde ku konštruktívnej interferencii a intenzita svetla tak bude maximálna, alebo naopak dôjde k deštruktívnej interferencii a bod v mieste interferencie zostane neosvetlený záleží na ich fázovom rozdiele a ten zas na rozdiele dráh, ktoré tieto vlny prejdú.

Keďže interferenciu meriame vo vzdialenosti násobne dlhšej ako je vzdialenosť medzi štrbinami, môžeme vlny aproximovať na rovnobežky. Takto v naznačenom trojuholníku za nimi vznikne pravý uhol a dráhový rozdiel bude potom určený hodnotou $\Delta l = d\,sin(\theta)$.

V prípade, že vlnenie z koherentného zdroja svetla dopadá kolmo na rovinu s dierkami bude počiatočná fáza v obidvoch štrbinách rovnaká a preto ju môžme z ďalsích výpočtov vynechať.

Po dosadení komplexnej amplitúdy rovinnej vlny do rovnice 2.6, získame nasledujúce vyjadrenie:

$$I = |A_{l_1}|^2 + |A_{l_2}|^2 + 2A_{l_1}A_{l_2}cos(kr - kr + kdsin(\theta)) \tag{2.10}$$

z ktorého je jasne vidieť, že maximálnu intenzitu dosiahneme, ak hodnota funkcie kosínus bude rovná číslu 1. Vychádzajúc z tejto úvahy môžme odvodiť vzťah pre prvé, resp. m-té difrakčné maximum:

$$\begin{aligned} k\,d\,sin(\theta) &= m2\pi \\ sin(\theta) &= \frac{m\lambda}{d} \end{aligned} \tag{2.11}$$

kde $sin(\theta)$ predstavuje uhol, o ktorý bude dané difrakčné maximum m vychýlené od pôvodného smeru. Ak by sme mali namiesto dvoch štrbín difrakčnú mriežku tvorenú veľkým množstvom jemných čiar, d by predstavovalo mriežkovú konštantu, čiže ich hustotu.

V prípade, že vlnenie bude dopadať pod uhlom iným ako 90 stupňov, rozšírime difrakčnú rovnicu nasledujúcim spôsobom:

$$sin(\theta_{out}) = \frac{m\lambda}{d} + sin(\theta_{in}) \tag{2.12}$$

kde člen $sin(\theta_{in})$ reprezentuje uhol pod ktorým vlnenie dopadá a $sin(\theta_{out})$ uhol, pod ktorým budeme pozorovať m-té difrakčné maximum.

2.2 Optická holografia

Optická holografia je časť odboru holografie, ktorá sa zaoberá záznamom a rekonštrukciou hologramov pomocou laserov a skutočného fyzického vybavenia. Jej základ tvoria fyzikálne princípy popísané v predchádzajúcej sekcii, ktoré jej umožňujú zachytiť kompletné svetelné pole snímaného predmetu a vytvoriť tak jeho takmer dokonalý 3D obraz, čiže hologram.

Holografia je v mnohom podobná fotografii, obe techniky zachytávajú svetelné pole prichádzajúceho svetla, ale rozdiel je v tom ako je táto informácia kódovaná na ploche záznamového filmu. V prípade fotografie je zaznamenaná len intenzita dopadajúceho svetla. V mieste vysokej intenzity film sčerná viac, v mieste nízkej intenzity sčerná menej. V prípade holografie však okrem intezity treba zachytiť aj smer svetelných lúčov, čo s klasickým fotografickým filmom nie je možné. Riešením je kódovať amplitúdu aj fázu vo forme difrakčného vzoru. Tento difrakčný vzor sa skladá z prúžkov (fringes) vytvorených interferenciou objektovej a referenčnej vlny, ktoré potom pri rekonštrukcii laserom spôsobujú difrakciu, čiže ohýbanie lúčov svetla v požadovanom smere.

Medzi najdôležitejšie typy hologramov patria osový hologram (ďalej už len inline hologram) a mimoosový hologram (ďalej už len off-axis hologram), ktoré budú postupne popísané v jednotlivých podkapitolách.

2.2.1 Inline hologram

Bol prvým typom hologramu zostrojený už Denisom Gaborom. Svoj názov získal podľa rozmiestnenia jednotlivých prvkov pri jeho vytváraní, kde zdroj monochromatického svetla spolu so zaznamenávaným predmetom ležia na jednej osi kolmej k rovine fotografického filmu.

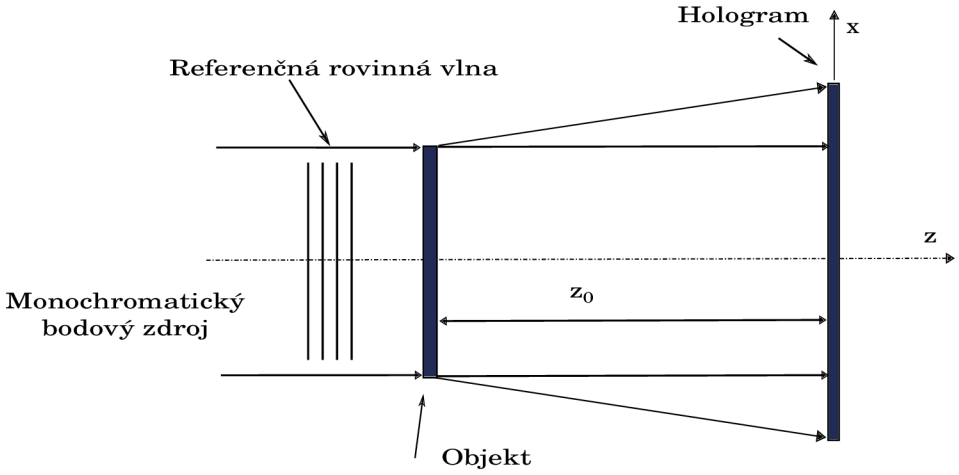
Toto rozostavenie je ukázané na obrázku 2.3, z ktorého vidno, že objekt musí byť dostatočne priehľadný, aby sa mohla časť zdrojového svetla dostať na záznamový film nevychýlená a fungovať tak ako referenčný lúč. Z tohto dôvodu inline hologram nedokáže zaznamenávať nepriehľadné objekty a je vhodný skôr na vytváranie hologramov priestorovo riedkych objektov.

Svetlo dopadajúce na fotografický film obsahuje dve zložky. Prvou je nevychýlené žiarenie priamo zo zdroja, ktorého komplexná amplitúda r je na celej rovine holografického filmu konštantná. Druhou je svetlo odrazené od čiastočiek objektu, ktorého komplexná amplitúda v bode (x, y) je $\tilde{o}(x, y)$ a zároveň $|\tilde{o}(x, y)|$ je oveľa menšie ako r.

Za predpokladu, že transmitancia (priepustnosť) fotografického filmu môže byť vyjadrená vzorcom:

$$\mathbf{t} = \mathbf{t}_0 + \beta T I \tag{2.13}$$

kde $\mathbf{t}_0$ určuje konštantnú transmitanciu pozadia, T je čas expozície (doba počas ktorej dopadalo svetlo na film) a β je parametrom fotografického materiálu, tak s pomocou rovnice 2.6



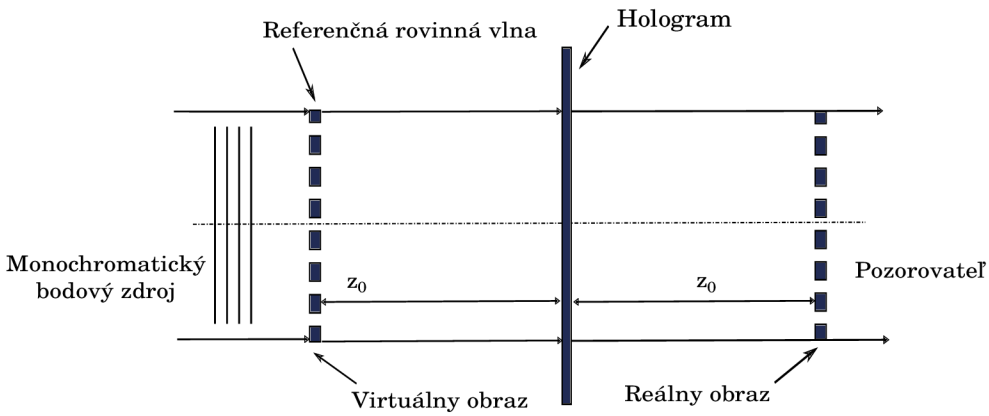
Obrázek 2.3: Záznam inline hologramu

môžme pre komplexnú amplitúdu transmitovanú hologramom odvodiť tento vzťah:

$$\tilde{u}(x,y) = r(\mathbf{t}_0 + \beta T r^2) + \beta T r|\tilde{o}(x,y)|^2 + \beta T r^2 \tilde{o}(x,y) + \beta T r^2 \tilde{o}^*(x,y) \tag{2.14}$$

Člen $r(\mathbf{t}_0 + \beta T r^2)$ tejto rovnice reprezentuje priamo transmitované svetlo. Druhý člen rovnice je kvôli prvotnému predpokladu, že $|\tilde{o}(x,y)|$ je oveľa menšie ako r veľmi malý a môžme ho zanedbať. Tretí člen je zodpovedný za vytvorenie virtuálneho obrazu a štvrtý za vytvorenie reálneho obrazu.

Obrázok 2.4 znázorňuje rekonštrukciu zaznamenaného inline hologramu. Tá prebieha tak, že sa film s hologramom osvieti monochromatickým svetlom s rovnakými vlastnosťami a v rovnakom smere aký mal referenčný lúč pri zázname.



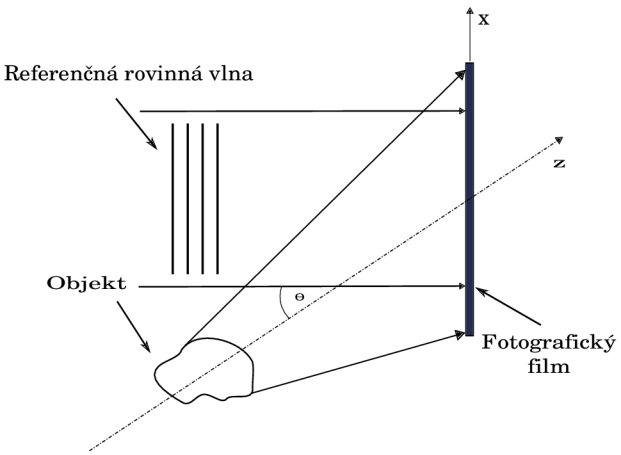
Obrázek 2.4: Rekonštrukcia inline hologramu

Dôvodom nízkej kvality tohto typu hologramov je fakt, že pozorovateľ pri rekonštrukcii vidí virtuálny obraz v superpozícii s reálnym obrazom a žiarením zo zdroja svetla. Toto spolu s nemožnosťou zaznamenať nepriehľadné predmety spôsobilo počiatočný veľmi nízky záujem o holografiu.

2.2.2 Off-axis hologram

Vynájdenie tejto záznamovej konfigurácie bol veľmi významný krok vpred, ktorý umožnil oddelenie virtuálneho obrazu od reálneho a zaznamenanie aj málo priehľadných objektov.

Jej podstata spočíva v rozštiepení zdrojového svetla na referenčný a objektový lúč. Referenčný lúč je namierený priamo na fotografický film, zatiaľ čo objektový lúč je smerovaný najprv na zaznamenávaný predmet a od neho sa ďalej odráža v rôznych smeroch na ten istý film, kde spolu obe vlny interferujú. Zjednodušený náčrt tejto techniky je na obrázku 2.5.



Obrázek 2.5: Záznam off-axis hologramu

Za predpokladu, že komplexná amplitúda referenčnej vlny môže byť vyjadrená vzťahom:

$$\tilde{r}(x,y) = r\, exp(i2\pi\xi_r x) \tag{2.15}$$

kde $\xi_r = sin\theta/\lambda$ a komplexná amplitúda objektovej vlny vzťahom:

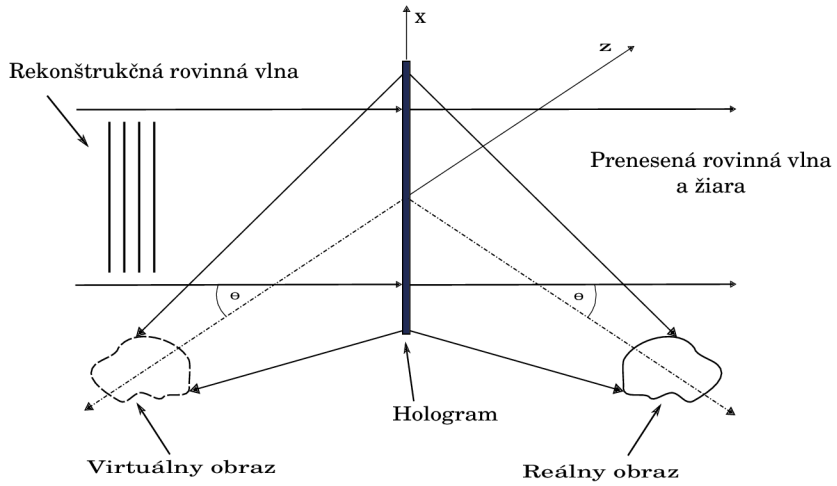
$$\tilde{o}(x,y) = |\tilde{o}(x,y)|exp[-i\Phi(x,y)] \tag{2.16}$$

tak s použitím vzorcov 2.6 a 2.13 môžme odvodiť výslednú amplitúdovú transmitanciu hologramu:

$$\begin{aligned}\tilde{u}(x,y) =& \tilde{r}(x,y)\mathbf{t}(x,y)\\ =& \mathbf{t}_0 r\, exp(i2\pi\xi_r x)+\\ & \beta T r|\tilde{o}(x,y)|^2 exp(i2\pi\xi_r x)+\\ & \beta T r^2\tilde{o}(x,y)+\\ & \beta T r^2\tilde{o}^*(x,y)exp(i4\pi\xi_r x)\end{aligned} \tag{2.17}$$

Prvý člen tejto rovnice odpovedá priamo transmitovanému svetlu, kým druhý predstavuje okolitú žiaru sprevádzajúcu toto svetlo. Tretí a štvrtý člen spôsobujú vytvorenie virtuálneho a reálneho obrazu. Obidva tieto obrazy sú odklonené od smeru, v ktorom prichádza

referenčná vlna o uhol θ, vďaka čomu potom nedochádza k ich prekrytiu počas rekonštrukcie. Vzhľadom na to, že reálny obraz je hĺbkovo prevrátený ho niekedy voláme *pseudoskopickým*, zatiaľ čo virtuálny obraz sa nazýva *ortoskopickým*. Rekonštrukcia je v zjednodušenej forme naznačená na obrázku 2.6.



Obrázek 2.6: Rekonštrukcia off-axis hologramu

2.3 Digitálna holografia

Ďalším veľkým odvetvím holografie je digitálna holografia. Tá využíva v zásade rovnaké princípy ako jej optická predchodkyňa, ale realizuje ich prostredníctvom vzorkovania a numerickej simulácie v počítačoch. Oproti klasickému optickému prístupu tak získame niekoľko nezanedbateľných výhod.

Asi najväčsou je možnosť vytvoriť hologram prakticky ľubovoľného aj neexistujúceho objektu. Stačí dodať jeho vhodný matematický popis a podporné algoritmy sa už o všetko postarajú. Ďalšou dôležitou výhodou, vďaka ktorej sa digitálna holografia stáva čoraz populárnejšou, je odstránenie komplikovanej optickej záznamovej zostavy. Tá zo samotnej fyzikálnej podstaty využívaných dejov a extrémne malej vlnovej dĺžky svetla býva veľmi často citlivá na akékoľvek vonkajšie rušenie. Stačí, aby došlo k zmene fázy používaného laserového svetla o pol vlnovej dĺžky a celý výsledok môže byť znehodnotený.

Samozrejme všetko má okrem svojich výhod aj svoje nevýhody. V prípade digitálnej holografie je to napríklad extrémna výpočtová zložitosť. Keďže sa pohybujeme v rádoch vlnovej dĺžky svetla, tak aj na hologramy o rozmeroch niekoľko centimetrov treba gigapixelové rozlíšenia. Taktiež vďaka tomu, že počítače dokážu pracovať len s konečnou presnosťou, sa nevyhneme niektorým typickým neduhom digitálneho spracovania ako je napríklad aliasing. Ten vzniká pri vzorkovaní spojitej informácie, keď kvôli konečnej pevne danej presnosti nie je možné od seba odlíšiť veľmi blízke hodnoty.

Z pohľadu tejto práce je najdôležitejšou oblasťou digitálnej holografie syntéza hologramov, ktorej bude venovaná prvá podkapitola. V druhej podkapitole potom budú stručne popísané aj zvyšné oblasti.

2.3.1 Syntéza hologramov

Jedná sa o oblasť holografie, ktorá sa zaoberá počítačovým generovaním hologramov na základe numerickej simulácie difrakcie a interferencie. Z pohľadu potrebného výpočtového výkonu je zo všetkých oblastí asi najnáročnejšia, ale to je dané už samotnou algoritmickou zložitosťou väčšiny známych postupov.

Tým najpriamočiarejším je algoritmus `Diffraction integral evaluation`, teda počítanie difrakčného integrálu. Jeho základ spočíva v dvoch zanorených cykloch, ktorými postupne, podľa vzorca 2.9, vyhodnotí sumu príspevkov komplexnej amplitúdy každého bodového zdroja v danom mieste optického poľa. V pseudokóde tento algoritmus vyzerá nasledovne:

foreach $sample_{x,y}$ **in** $optical\ field$ **do**
    $sample_{x,y} = 0 + 0\mathrm{i}$;
    foreach $point source$ **in** $point\ cloud$ **do**
        $sample_{x,y}$ += SphericalWaveEquation($point\ source$, x, y);
    end
end

Algorithm 1: Diffraction integral evaluation

Jeho sila spočíva najmä v jednoduchosti a veľmi dobrej paralelizovateľnosti a implementovateľnosti v hardvéri. Nevýhodou je pomerne náročná realizácia pokročilejších vlastností ako je výpočet osvetlovacieho modelu alebo viditeľnosti. Ďalším limitujúcim činiteľom je fakt, že vstupná scéna musí byť poskladaná výlučne z bodových zdrojov, ktorých počet však bude v zložitých scénach veľmi veľký.

Problémy s viditeľnosťou a osvetlovacím modelom rieši nasledovný algoritmus s názvom `Ray casting`. Ten môže byť implementovaný v dvoch základných variantách.

V prvej povedieme z každého bodového zdroja scény polpriamku do každého bodu na optickom poli. Ak táto polpriamka po ceste nepretne žiaden ďalší objekt, tak komplexnú amplitúdu bodu z jej počiatku započítame do výslednej sumy v danom mieste optického poľa, inak bodový zdroj ignorujeme.

Druhou variantou je opačný prístup, keď z každej vzorky optického poľa vedieme niekoľko polpriamok pod rôznym uhlom do scény a následne nájdené priesečníky považujeme za bodové zdroje.

Obrovskou výhodou druhej varianty je fakt, že počet vrhnutých lúčov je zakaždým konštantný a nezávisí na veľkosti vstupnej scény. Ďalším problémom, ktorý treba riešiť pri prvej variante je nerovnomernosť rozloženia bodových zdrojov v scéne, čiže situáciu, keď veľké množstvo bodov reprezentuje len jej nepomerne malú časť.

Výpočtová zložitosť tohto postupu je však oveľa vyššia ako pri akejkoľvek inej metóde a ani hardvérová implementácia nie je práve jednoduchá, čo je dané nutnosťou počítať priesečník lúča s objektami scény. Na druhej strane z pohľadu kvality získaných výsledkov táto metóda jasne dominuje.

Treťou možnosťou je algoritmus `Fourier hologram synthesis`, čiže využitie fourierovej transformácie. Toto je možné vďaka Fresnelovej aproximácii difrakcie, ktorá síce nebola popisovaná v tejto práci, ale zvedavý čitateľ sa s ňou môže bližšie zoznámiť napríklad v [7] alebo v [11].

V princípe metóda funguje tak, že scéna sa rozdelí na rezy, ktoré sú potom jednotlivo propagované na optické pole. Táto metóda je vďaka fourierovej transformácii pomerne

rýchla, ale nedovoľuje jednoducho implementovať niektoré pokročilejšie efekty počítačového zobrazovania, ako je napríklad už zmienený osvetľovací model.

Samozrejme existuje ešte množstvo ďalších algoritmov, ktoré sú viac alebo menej úspešné. Tie ktoré sú efektívnejšie však svoju rýchlosť väčšinou ťažia z optimalizácií a zjednodušení uplatniteľných len pre konkrétnu úzku skupinu špecifických podmienok. Za všetky stačí spomenúť napríklad `MIT holovideo`, ktoré je schopné zobrazovania takmer v reálnom čase aj pri dnešných technológiách, ale za cenu obetovania vertikálnej paralaxy a potreby špeciálneho rekonštručného zariadenia. Detailnejšie informácie o ďalších nepopisovaných postupoch sa dajú nájsť napríklad v [10].

2.3.2 Ostatné oblasti digitálnej holografie

Sem patrí napríklad reprodukcia hologramov, čiže prevod optického poľa z reprezentácie v pamäti počítača do takej formy, z ktorej bude možné hologram opticky rekonštruovať.

V závislosti na type zariadenia existuje viacero spôsobov. Asi najbežnejším je prevod na čiernobielu bitmapu, ktorá sa potom nechá vypáliť do skleného rezistu pomocou laserovej alebo elektrónovej litografie. Obe tieto technológie sú však pomerne drahé. V závislosti na zvolenej veľkosti pixelu a rozmeroch vyrábanej štruktúry sa môže cena za jeden kus pohybovať až v rádoch tisícov českých korún [1]. O niečo lacnejšia, ale zároveň aj menej kvalitná je osvitová jednotka, kde sa ceny pohybujú v rádoch stoviek korún [2]. Výroba je uskutočniteľná taktiež prostredníctvom obyčajnej laserovej tlačiarne. V tomto prípade sa však kvalita výsledku môže veľmi rôzniť v závislosti od výrobcu a rôznych nastavení špecifických pre konkrétne zariadenie. Neobvyké nie je ani to, že ovládač takéhoto zariadenia vykonáva dodatočné softvérové predspracovávanie zaslaných dát bez toho aby nad tým užívateľ mal nejakú kontrolu.

Ďalšími oblasťami digitálnej holografie sú numerická rekonštrukcia a digitálny záznam hologramov. Snahou rekonštrukcie je počítačová simulácia optickej rekonštrukcie. Toto je významné v prípade potreby overenia výsledkov syntézy ešte pred zahájením výroby.

Digitálne zaznamenávanie podobne ako optické má za cieľ zachytiť konkrétny stav svetelného poľa objektu v danom okamihu. Rozdiel ale tkvie v používanom záznamovom médiu. Zatiaľ čo pri klasickej holografii je to napríklad fotografická emulzia, v prípade digitánej holografie sa jedná o CCD (charge coupled device) kameru alebo podobné zariadenie. Výhodou je, že zachytené dáta sa dajú ďalej priamo spracovávať počítačom.

[1] Tieto údaje boli získané prevažne emailovou komunikáciou s [6] a [5]

[2] Údaje získané z [13]

Kapitola 3

OpenCL

Ako už možno z predchádzajúceho textu vyplynulo hlavnou prekážkou v širšom využití syntetických hologramov v praxi je ich neúnosná výpočtová náročnosť. Vzhľadom na to, že zatiaľ nebol objavený spôsob ako túto náročnosť algoritmicky zredukovať na prijateľnú úroveň a keďže sa prakticky jedná len o masívne množstvo opakovaných matematických výpočtov, tak vhodným riešením sa ukazuje byť využitie paralelného spracovania.

Prirodzeným typom paralelizácie je zapojenie viacerých počítačov do spoločnej siete a pomocou vhodnej stratégie medzi nich rozdeliť jednotlivé časti úlohy. Tento spôsob spracovania sa nazýva distribuovaný výpočet. Existuje viacero kvalitných riešení a knižníc pre distribuované výpočty, z ktorých neoficiálnym štandardom je protokol MPI (Message Passing Interface) a jeho implementácia MPICH [4].

Vďaka stále rastúcemu výkonu a lepšej programovateľnosti grafických kariet sa však ponúka ďalšia možnosť a tou je GP-GPU, čiže všeobecné výpočty na grafickom procesore. Jedná sa o pomerne mladú, no o to dynamickejšiu oblasť s množstvom rôznych rozhraní a knižníc od rozličných výrobcov. Za všetky stačí spomenúť napríklad paralelnú architektúru CUDA [3], aplikačné rozhranie (API) Direct Compute [2] od firmy Microsoft, alebo najnovší prírastok v podobe OpenCL.

Keďže v tejto práci boli využité prostriedky posledne menovaného, tak nasledujúce state budú venované práve tomuto štandardu. Najprv budú predstavené jeho základné princípy a vlastnosti a tie budú ďalej rozvinuté v nadväzujúcich pasážach. Kompletný výklad je však ponechaný na oficiálnu dokumentáciu [12], prípadne odbornú literatúru [15].

3.1 Základné vlastnosti

OpenCL, čiže Open Computing Language, je otvorený priemyslový štandard na podporu paralelného programovania v heterogénnom prostredí. Jeho vznik a ďalší vývoj má na starosti konzorcium Khronos, ktoré združuje popredných svetových výrobcov hardvéru a softvéru ako je Apple, AMD, Intel, NVidia, IBM a mnohí iní. Toto isté konzorcium má na svedomí aj ďalšie populárne štandardy, napríklad OpenGL, WebGL, alebo COLLADA.

Samotné OpenCL, tak nepredstavuje žiadnu konkrétnu knižnicu funkcií, ale skôr sa jedná o definíciu spoločných vlastností a súbor požiadaviek na konformnú implementáciu. Vlastné skompilované knižnice sú potom poskytované vývojárom ako súčasť rôznych SDK (Software Developement Kit) výrobcov.

Typický OpenCL program je rozdelený na dve časti, a to na hostiteľskú časť a časť, ktorá beží na vybranom zariadení. Takýmto zariadením je väčšinou GPU, ale môže sa jednať aj

o CPU, či iné špecializované procesory. Vďaka dômyselnému návrhu samotného štandardu nie je dokonca vylúčené ani simultánne spracovanie výpočtov na obidvoch typoch čipov naraz, hoci zvyčajne v roli hostiteľa pôsobí CPU a v roli výpočtového zariadenia GPU.

Úlohou hostiteľa je riadenie výpočtov, alokácia pamäte, správa získaných zdrojov a všetka nevyhnutná inicializácia. V prevažnej väčšine prípadov je táto časť programu napísaná v C/C++, pre ktoré je OpenCL primárne určené. Z ostatných jazykov, býva rozhranie štandardu dostupné prostredníctvom tzv. bindings, čiže tenkej medzivrstvy, ktorá prekladá volania príslušných funkcií na ich ekvivalenty v konkrétnom jazyku. Bindings však už nie sú súčasťou OpenCL štandardu spravovaného združením Khronos.

Výpočtové zariadenia sa tak starajú len o čo najrýchlešie spracovanie zadaných úloh, ktoré sú programované v jazyku OpenCL C. Ten je odvodený z jazyka C, konkrétnejšie z jeho normy ISO C99 a rozširuje ho o dodatočné konštrukty vhodné najmä pre paralelné algoritmy. Napríklad OpenCL C zavádza nové vektorové dátové typy, ktoré umožňujú prirodzeným spôsobom využívať SIMD inštrukcie spracúvajúce niekoľko čísel naraz. Vo svete x86 procesorov sú tieto inštrukcie dobre známe pod skratkami SSE a AVX. Ďalším nemenej významným vylepšením je aj rozsiahla štandardná knižnica matematických funkcií, ktorá výrazne uľahčuje programovanie.

Avšak ešte predtým ako môže zariadenie začať vykonávať OpenCL kód je nutné ho skompilovať. Toto sa na rozdiel od klasických jazykov typu C a C++ deje za behu aplikácie. Dôvodom tohto rozhodnutia je najmä platformová nezávislosť, keďže inštrukčné sady GPU procesorov rôznych dodávateľov sú medzi sebou výrazne menej kompatibilné ako v prípade klasických procesorov. Navyše ani miera spätnej kompatibility v rámci modelových radov toho istého výrobcu nemusí byť úplne stopercentná. Ďalším dobrým dôvodom je fakt, že kompilátor na cieľovom stroji môže uskutočniť efektívnejšie optimalizácie, prípadne programátor má šancu prispôsobiť svoju implementáciu pre konkrétnu platformu (využitím preprocesoru, alebo výberom vhodnej verzie kódu tesne pred prekladom).

Pretože OpenCL štandard sa snaží obsiahnuť čo najširšie spektrum zariadení, tak definuje dva stupne splnenia jeho požiadaviek a to `Full Profile` a `Embedded profile`. Prvý je určený pre veľké stanice disponujúce množstvom výkonu, zatiaľ čo druhý je určený skôr pre malé vstavané pristroje a mobilné telefóny.

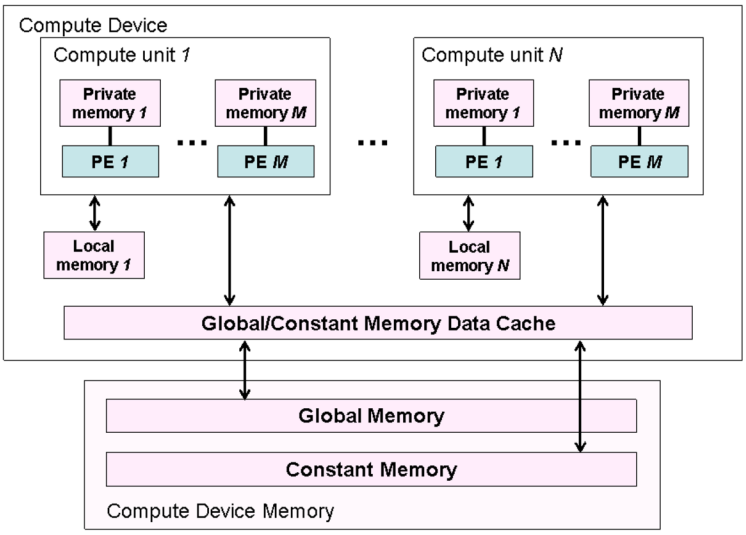
Hlavnými métami OpenCL je teda ponúknuť programový model pre dátový a úlohový paralelizmus, poskytnúť abstrakciu podporného hardvéru a vytvoriť jednotné API využívajúce rovnaké princípy ako už zavedený štandard OpenGL.

3.2 Abstrakcia hardvéru

OpenCL vo svojej špecifikácii definuje niekoľko rôznych modelov a abstrakcií uľahčujúcich konečnú platformovú nezávislosť.

Jednou z týchto abstrakcií je aj OpenCL zariadenie definované ako množina výpočtových jednotiek (compute units) ďalej rozdelených na procesné elementy (processing elements), v ktorých prebieha samotný výpočet. Procesné elementy sú zas definované ako SIMD (Single Instruction Multiple Data) alebo SPMD (Single Program Multiple Data) jednotky schopné sekvenčne spracovať vstupný prúd inštrukcií. Ich vzájomný beh však už ide plne paralelne.

Druhým významným modelom zavedeným štandardom OpenCL je abstrakcia hierarchie pamätí ukázaná na obrázku 3.1. Podľa nej existujú štyri druhy pamätí, a to globálna, koštantná, lokálna a privátna. Najväčšou, ale zároveň aj najpomalšou z nich je globálna pamäť. Tá je prístupná na čítanie i zápis všetkým výpočtovým jednotkám a procesným



Obrázek 3.1: Pamäťová hierarchia v OpenCL. Obrázok je prebraný z [12]

elementom po celú dobu ich výpočtu. Zapisovať a čítať túto pamäť môže aj hostiteľ, ktorý prostredníctvom nej predáva zariadeniu vstupné dáta a získava výsledky.

Druhým typom pamäte je konštantná pamäť. Tá je väčšinou implementovaná ako súčasť globálnej pamäte, do ktorej zariadenie nemá právo zápisu, a preto býva veľmi často cacheovaná za účelom zvýšenia rýchlosti prístupu.

Lokálna pamäť býva typicky umiestňovaná priamo na zariadení a preto je prístup k nej rýchleší ako v predošlých dvoch prípadoch. Využíva sa hlavne na zdieľanie dát medzi pracovnými jednotkami (work-items) vrámci pracovnej skupiny (work-group). Hostiteľ k nej nemá žiadny prístup a môže ju pre zariadenia len alokovať.

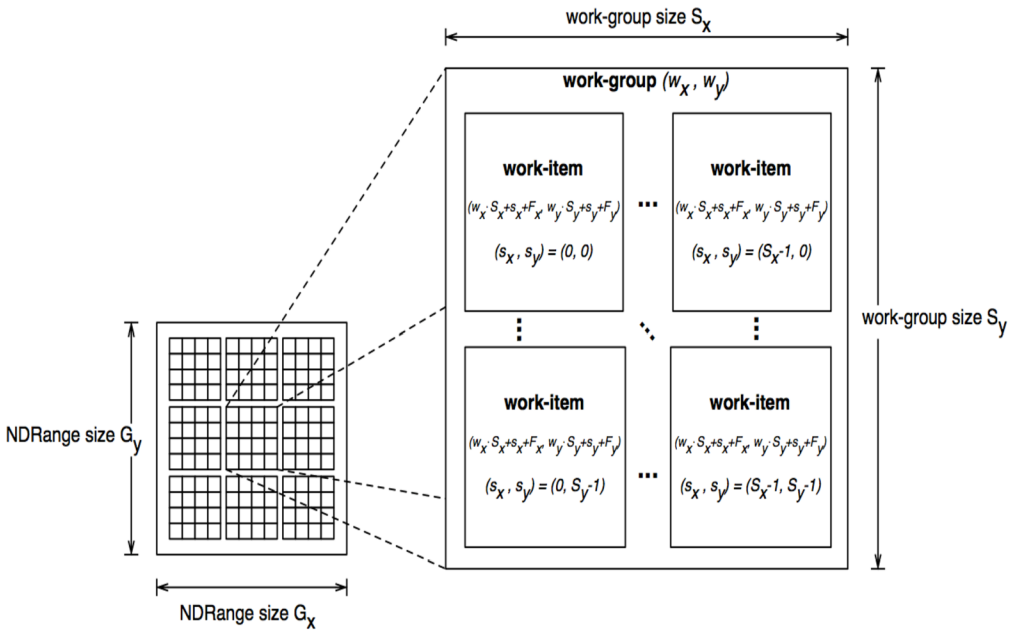
Posledným typom je privátna pamäť, ktorá je najrýchlejšia, ale zároveň aj najmenšia zo všetkých spomenutých. Využitie nachádza medzi pracovnými jednotkami, ktoré do nej ukladajú svoje lokálne premenné a medzivýpočty.

3.3 Exekučný model

Okrem už spomenutých konceptov definuje OpenCL štandard aj spôsob, akým má byť spracovanie paralelnej úlohy v zariadení rozdelené.

Toto rozdelenie je vyjadrené indexným priestorom nazývaným NDRange a znázorneným na obrázku 3.2. Jeho fungovanie sa dá prirovnať k sérii zanorených cyklov používaných v mnohých klasických programovacích jazykoch. Hĺbka zanorenia potom predstavuje počet dimenzií indexného priestoru a každá iterácia jednu pracovnú jednotku pomenovanú v terminológii OpenCL ako work-item. Tieto sú ďalej združované do pracovných skupín umožňujúcich ich synchronizáciu a zdieľanie medzivýsledkov.

Aby boli jednotlivé pracovné jednotky od seba navzájom jednoznačne odlíšiteľné definuje OpenCL niekoľko typov identifikátorov, ktorých návrh vychádza z prirovnania k zanoreným cyklom. Každý z nich je totiž n-ticou čísel odpovedajúcou aktuálnemu počtu dimenzií. Ak by teda programátor NDRange stanovil ako dvoj-dimenzionálny, bude každý identifikátor dvo-



Obrázek 3.2: Indexný priestor NDRange. Obrázok je prebraný z [12]

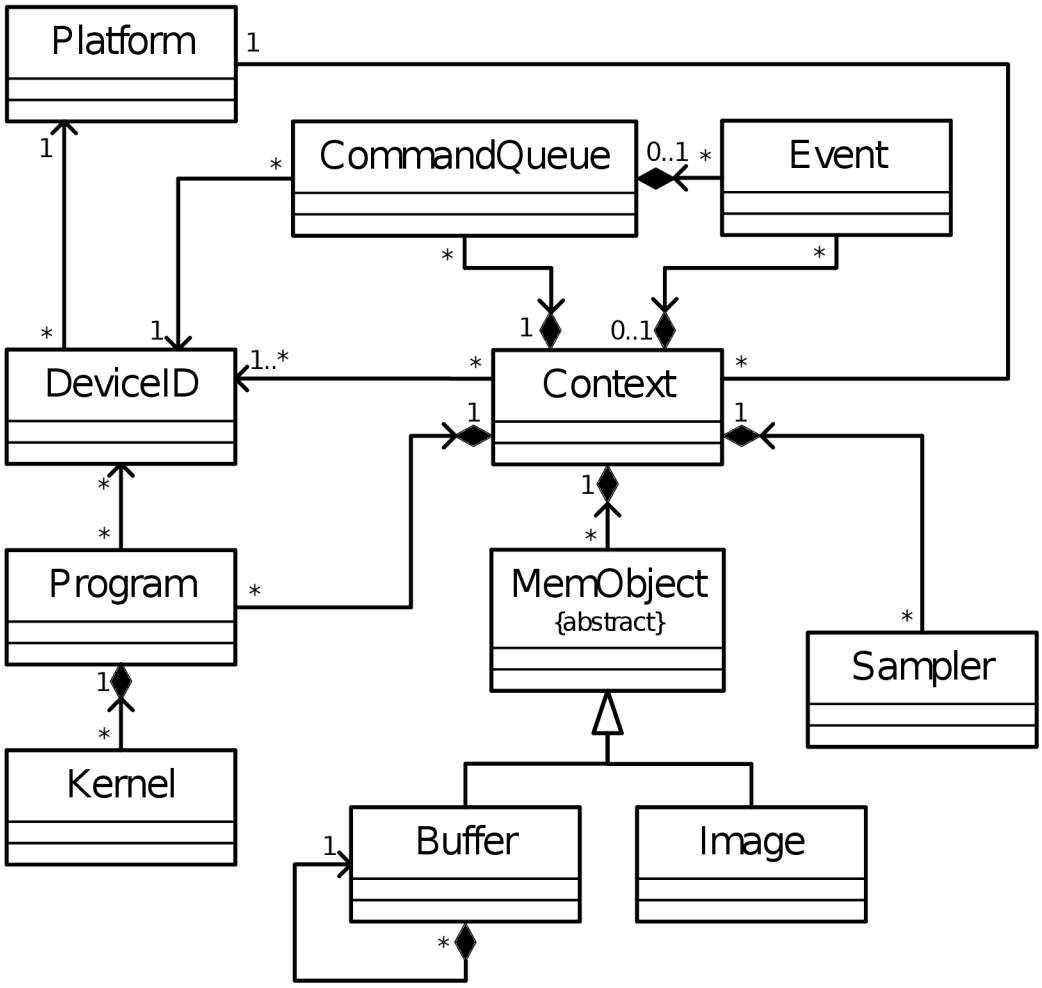
jicou indexov, ak by ho definoval ako trojdimenzionálny bude trojicou, atď. Identifikátory sú dvojakého druhu a to lokálne, ktoré odlišujú pracovnú jednotku len vrámci skupiny a globálne, ktoré ju odlišujú od všetkých ostatných definovaných pracovných jednotiek. Taktiež každá skupina má svoj vlastný skupinový identifikátor.

3.4 Hlavné prvky API

Aj napriek tomu, že špecifikácia OpenCL je určená pre jazyk C, ktorý je procedurálnym jazykom a sám o sebe neobsahuje priamu podporu objektovo orientovaného programovania je hostiteľské aplikačné rozhranie navrhnuté v duchu týchto princípov. Dokazuje to aj nižšie uvedený UML diagram tried znázorňujúci jednotlivé štruktúry OpenCL a ich vzájomné prepojenie.

Z obrázku vidno, že centrálnym objektom je kontext. Ten združuje všetky ostatné objekty a umožnuje medzi nimi komunikáciu a synchronizáciu. Súčasťou kontextu sú aj výpočtové zariadenia, avšak nie je podmienkou, aby v ňom bolo zahrnuté úplne každé dostupné zariadenie. Navyše v rámci jedného kontextu sa môžu nachádzať len zariadenia z tej istej platformy, takže spravovať naraz grafickú kartu od NVidie a procesor od Intel-u je vylúčené. Riešením tohto problému môže byť vytvorenie osobitného kontextu pre každú platformu zvlášť. V kóde hostiteľa reprezentuje kontext štruktúra `cl_context`.

Ďalšími dvoma významnými abstrakciami sú platforma a zariadenie. Prvá menovaná identifikuje konkrétnu implementáciu OpenCL štandardu od konkrétneho dodávateľa. Jej úlohou je sprostredkovať informácie o podporovanej verzii, profile, rozšíreniach a dostupných zariadeniach. Druhá abstrakcia predstavuje skutočné GPU, alebo viacjadrový procesor prítomný na počítači. Pomocou nej sa v súčinnosti s platným indentifikátorom platformy, alebo platným kontextom dajú získať informácie o zastupovanom zariadení a jeho schopnostiach. Platformu v kóde reprezentuje štruktúra `cl_platform` a zariadenie štruktúra `cl_device`.



Obrázek 3.3: UML diagram OpenCL. Obrázok je prebraný z [12]

Keďže sa OpenCL skladá z dvoch častí, kde jedna je napísaná v klasickom C/C++ a druhá v OpenCL C, tak existuje objekt program, ktorý reprezentuje zdrojový kód tohto jazyka. Ten potom umožnuje spustiť kompiláciu, zistiť informácie o jej priebehu a získať výsledné preložené binárne súbory. Od úspechu kompilácie sa ďalej priamo odvíja schopnosť hostiteľa vyvárať kernel objekty a manipulovať s nimi. Kernely sú špeciálne funkcie OpenCL C kódu uvedené kľúčovým slovom `__kernel__` a označujú vstupný bod do programu. Toto je možné si predstaviť podobne ako funkciu `main` v mnohých iných programovacích jazykoch s tým rozdielom, že v jednom zdrojovom súbore môže byť hneď niekoľko kernelov. Kernel objekt je tak reprezentáciou kernel funkcie na strane hostiteľa, ktorý ho využíva k predávaniu parametrov a zahájeniu výpočtu.

Priamo na samotné spustenie kernelu sa ale využíva ešte jedna veľmi dôležitá štruktúra a tou je fronta príkazov, anglicky command queue. Každé zariadenie v kontexte musí mať svoju vlastnú frontu príkazov, prostredníctvom ktorej može hostiteľ poveriť zariadenie vykonaním určitého kernelu, zapísať doňho alebo čítať z neho dáta a kopírovať dáta medzi jednotlivými zariadeniami. Existujú dva rozdielne režimy manipulácie s frontou. Prvý sa nazýva in-order, čiže synchrónne spracovanie príkazov v poradí v akom boli zadané, pričom zároveň platí, že vykonávanie nasledujúceho príkazu sa nezačne skôr ako budú spracované

všetky predošlé úlohy. Druhý režim sa nazýva out-of-order, teda asynchrónne spracovanie príkazov mimo zadaného poradia.

Ako prostriedok prenosu vstupných a výstupných dát sa využívajú pamäťové objekty (Mem Objects), ktoré sú dvojakého typu. Buď sa jedná o tzv. buffer objects, čiže buferové objekty, alebo sú to tzv. image objects, tj. obrázkové objekty. Buferové objekty sa využívajú v prípade prenosu jedno-, dvoj- alebo troj-dimenzionálnych dát, ktoré nemajú štruktúru obrázkov. Naopak obrázkové objekty sú špeciálne určené pre obrazové dáta, čo dáva OpenCL možnosť využiť určité špecifické optimalizácie. Napríklad image object môže byť na GPU umiestnený v textúrovacej pamäti. Ďalším rozdielom je, že obrázkové objekty môžu byť alebo len na čítanie, alebo len na zápis oproti všeobecným buferovým objektom, ktoré možu mať nastavené obe tieto prístupové práva naraz.

Sampler objekt taktiež úzko súsvisí s obrázkami, jeho hlavným poslaním je totižto definovať spôsob akým budú obrázky pri čítaní v kerneli vzorkované. Napríklad určuje čo sa stane ak zadané súradnice budú mimo definované rozmery obrázku a či sa jedná o ich normalizovanú, alebo nenormalizovanú verziu.

Posledným objektom z diagramu je udalosť. Tá zapúzdruje stav príkazu vykonávaného zariadením a umožnuje tak programátorovi ich synchronizáciu.

Kapitola 4

Návrh

Ešte predtým ako sa rozbehne celý proces výroby hologramu je nutné si ujasniť predstavu o jeho výsledných parametroch.

Toto je dôležité hneď z dvoch rôznych dôvodov. Po prvé, výroba kvalitného hologramu v dnešnej dobe predstavuje pomerne drahú záležitosť a keď sa pri počiatočnom určení parametrov urobí chyba, nie je možné výsledok jednoducho opraviť.

Po druhé, vzhľadom na povahu výpočtov je generovanie aj na súčasných výkonných počítačoch veľmi náročná operácia, ktorá môže trvať relatívne dlhý čas a tak neustále prepočítavanie výsledku taktiež neprichádza do úvahy.

V nasledujúcich kapitolách budú preto stanovené vhodné parametre pre počítané hologramy a načrtnutá zložitosť vykonávaných operácií na konkrétnych výpočtoch a číslach. Nakoniec bude zhrnutý súčasný stav, dostupné možnosti a odôvodnené vybrané riešenie.

4.1 Stanovenie parametrov riešenia

Kvalita výsledného hologramu je priamo závislá na kvalite dostupného zariadenia, najmä však na veľkosti jeho rozlíšenia. Tabuľka 4.1 zobrazuje prehľad najtypickejších kategórií zariadení spolu s ich najpodstatnejšími parametrami.

Typ	**Rozlíšenie** [µm/*pixel*]	**Maximálny uhol difrakcie** [°]	**Cena**
elektrónová litografia	0.1	90	až 30000 Kč/4x4 cm
laserová litografia	1	20	cca 6000 Kč/4x4 cm
osvitová jednotka	10	2	cca 100 Kč/A4
laserová tlačiareň	100	0.5	jednotky Kč/A4

Tabulka 4.1: Zariadenia na výrobu hologramov a ich parametre. Údaje boli získané emailovou komunikáciou s [6] a [5].

Ako to vyplýva aj zo vzorca 2.12, hustota rozlíšenia priamo ovplyvňuje maximálnu veľkosť uhla, o ktorý sa môže rekonštrukčná vlna vychýliť od pôvodného smeru referenčnej vlny.

Ak by tento uhol nebol dodržaný, tak výsledný obraz hologramu sa síce vytvorí, ale bude rôzne deformovaný (do hĺbky, do šírky, do výšky, atď), pričom miera deformácie rastie so zväčšujúcou sa vzdialenosťou od roviny hologramu. Navyše, aby bolo možné vidieť celý hologram aj pri malom uhle, musí byť zaznamenávaný objekt v dostatočnej vzdialenosti

od roviny, na ktorej záznam vytvárame. Problémom je, že so zväčšujúcou sa vzdialenosťou sa zároveň zhoršuje aj miera deformácie výsledného obrazu, iným slovom rastie miera aberácií.

Taktiež veľkosť tohto uhla bezprostredne vplýva na vzdialenosť medzi nultým difrakčným rádom, čiže priamym svetlom vyžiareným z rekonštrukčného lúča a prvým difrakčným rádom, ktorý predstavuje rekonštruovaný hologram. Pretože svetlo z nultého difrakčného maxima má podstatne vyššiu silu ako svetlo z hocijakého iného rádu, tak jeho intenzita hravo prekryje vytvorený hologram a z výsledku nebude vidieť takmer nič.

Po predchádzajúcej úvahe by sa tak mohla elektrónová litografia javiť ako najperspektívnejšia metóda výroby hologramu. Jej privysoká cena a množstvo výpočtov, ktoré by bolo na vygenerovanie takéhoto hologramu potreba, ju ale z ďalších úvah diskvalifikujú.

Pri pohľade na opačný koniec rebríčka sa zas ponúka možnosť tlače na laserovej tlačiarni. Táto druhá možnosť ale predstavuje opačný extrém, ktorý by značne limitoval rozsah rozumne rekonštruovateľných scén. Takisto, ako to bolo popísané aj v kapitole o digitálnej holografii, skutočné schopnosti zariadenia sa môžu značne líšiť od papierových údajov a preto nemá zmysel optimalizovať program pre toto riešenie.

Najvhodnejšou voľbou sa preto ukazuje byť navrhnutie aplikácie tak, aby bola schopná spracovať rozumne veľké hologramy s rozlíšením bodu 1–10µm.

Ak budeme teda uvažovať hologram o veľkosti 2x2cm a s rozlíšením jedného bodu optického poľa 2µm zistíme, že výsledné rozmery hologramu sa vyšplhajú na hodnotu 10000x10000 komplexných vzoriek optického poľa. Ak budeme ďalej uvažovať, že vstupná scéna sa skladá z 10000 bodových zdrojov, dostaneme sa na hodnotu 10000x10000x10000, čiže trilión potrebných výpočtov, ktoré navyše zahŕňajú operácie s odmocninou a goniometrickými funkciami.

Ďalej za predpokladu, že veľkosť desatinného čísla reprezentovaného dátovým typom float, ktorého bežná veľkosť v pamäti počítača sú 4B, budeme potrebovať na reprezentáciu každej komplexnej amplitúdy 8B celkovo. Vynásobením tejto hodnoty s rozmermi optického poľa získame číslo 800000000B, čo predstavuje zhruba 762MB potrebných na uloženie tohto poľa do pamäte počítača.

4.2 Zhodnotenie súčasného stavu a návrh riešenia

Z príkladov predvedených v predošlej sekcii je teda vidno, že výpočtové požiadavky sú skutočne nekompromisné aj napriek tomu, že rozmery samotného hologramu nie sú nijako zvlášť ohromujúce. Navrhnuté riešenie sa tak musí zaoberať dvomi základnými otázkami a to aký algoritmus zvoliť, aby výsledný čas výpočtu bol čo najkratší a akú technológiu k tomu použiť.

Po zvážení všetkých negatív a pozitív bola nakoniec vybraná metóda počítania difrakčného integrálu. Hoci tento spôsob neposkytuje najrealistickejšie výsledky a priamočiaru podporu riešenia viditeľnosti a osvetlovacieho modelu, zavážila jeho jednoduchosť a vynikajúca paralelizovateľnosť. Vďaka týmto vlastnostiam je algoritmus perfektným kandidátom na urýchlenie v GPU.

Druhá metóda vrhania svetelných lúčov do scény síce umožňuje riešenie spomenutých nedostatkov, avšak jej implementácia je znateľne náročnejšia a urýchlenie v HW nemusí byť kvôli potrebe počítať priesečníky lúčov so scénou až tak výrazné. Navyše sila vlastností poskytovaných touto metódou sa začne prejavovať až pri počítaní náročnejších priestorových scén, čo z pohľadu tejto práce nie je až tak dôležité.

Ďalšou riešenou otázkou je výber vhodnej technológie na implementáciu. Existuje množstvo rôznych paradigiem a frameworkov umožňujúcich písanie paralených aplikácií.

Jedným z takýchto prístupov sú distribuované výpočty reprezentované napríklad knižnicami MPICH a PVM. Plné využitie tejto metódy však vyžaduje prístup k výpočtovému clusteru počítačov a preto ako dostupnejší spôsob paralelizácie sa javí byť využitie sily súčasných grafických kariet.

Tie od svojich počiatkov, keď ako prísne grafický koprocesor obsahovali len fixný vykresľovací reťazec, prešli dramatickým vývojom a dnešné moderné karty už bývajú plne programovateľné. Táto programovateľnosť bola spočiatku dostupná len cez grafické API ako OpenGL a DirectX v podobe tzv. shader programov. Písanie všeobecných negrafických výpočtov cez tieto knižnice tak bolo pomerne náročné, vyžadujúce aby programátor vedel riešený problém vyjadriť v reči textúr a polygónov.

Túto situáciu zmenil príchod špecializovaného aplikačného rozhrania v podobe frameworku CUDA. Tento bol a stále je veľmi populárnou alternatívou, ktorej využitie je však možné len na grafických kartách od NVidie.

Ďalším z dostupných prostriedkov na programovanie GPU je rozhranie DirectCompute navrhnuté firmou Microsoft. Táto technológia je ale podobne ako aj ostatné od Microsoftu (napríklad DirectX) viazaná na platformu Windows a teda výsledné riešenie by bolo platformovo závislé.

Všetky spomenuté problémy sa tak snaží riešiť štandard OpenCL, ktorý je navyše navrhnutý s myšlienkou podpory heterogénneho prostredia, ktoré by obsiahlo v jednom programovacom rozhraní aj ostatné hardvérové architektúry ako sú klasické procesory, špecializované signálové zariadenia, či iné akcelerátory. Miernou nevýhodou tejto technológie je možný suboptimálny výkon aplikácie, čo je dané jej širokým záberom na množstvo rôzneho HW. Pri opatrnom programovaní a dodržovaní všetkých zásad je však možné dosiahnuť porovnateľný výkon ako napríklad s proprietárnym riešením CUDA.

Z tohto dôvodu a z dôvodu vlastníctva grafickej karty od AMD bola zvolená technológia OpenCL.

Kapitola 5

Implementácia

V predošlých kapitolách boli popísané najnutnejšie fyzikálne princípy holografie a hlavné vlastnosti paralelného programovacieho API s názvom OpenCL. Nasledujúce riadky sa preto pokúsia priblížiť spôsob ich využitia v práci a realizáciu samotného diela.

Výsledný program má formu konzolovej aplikácie napísanej v jazyku C++ a to hlavne z dôvodu všeobecne vyššieho výkonu natívneho kódu a zároveň možnosti plného využitia moderných techník objektovo orientovaného programovania.

Počas vývoja bol tiež braný ohľad na maximálnu možnú prenositeľnosť, či už vrámci jednotlivých operačných systémov, alebo aj vrámci rôznych kompilátorov. Program je teda preložiteľný a spustiteľný jak v prostredí Windows, tak i v prostredí Linux.

Ďalšou dôležitou vlastnosťou aplikácie je jej kompatibilita so sadou holografických nástrojov HoloToolkit vyvíjaných Západočeskou Univerzitou v Plzni [1]. Nástroje z tohto toolkitu umožňujú okrem iného previesť binarizáciu vypočítaného optického poľa a otestovať získané výsledky numerickou rekonštrukciou. Program tak neimplementuje celý renderovací reťazec od prevodu scény na bodové zdroje až po spracovanie výsledného hologramu, ale v posledných krokoch sa spolieha na spomenuté externé nástroje.

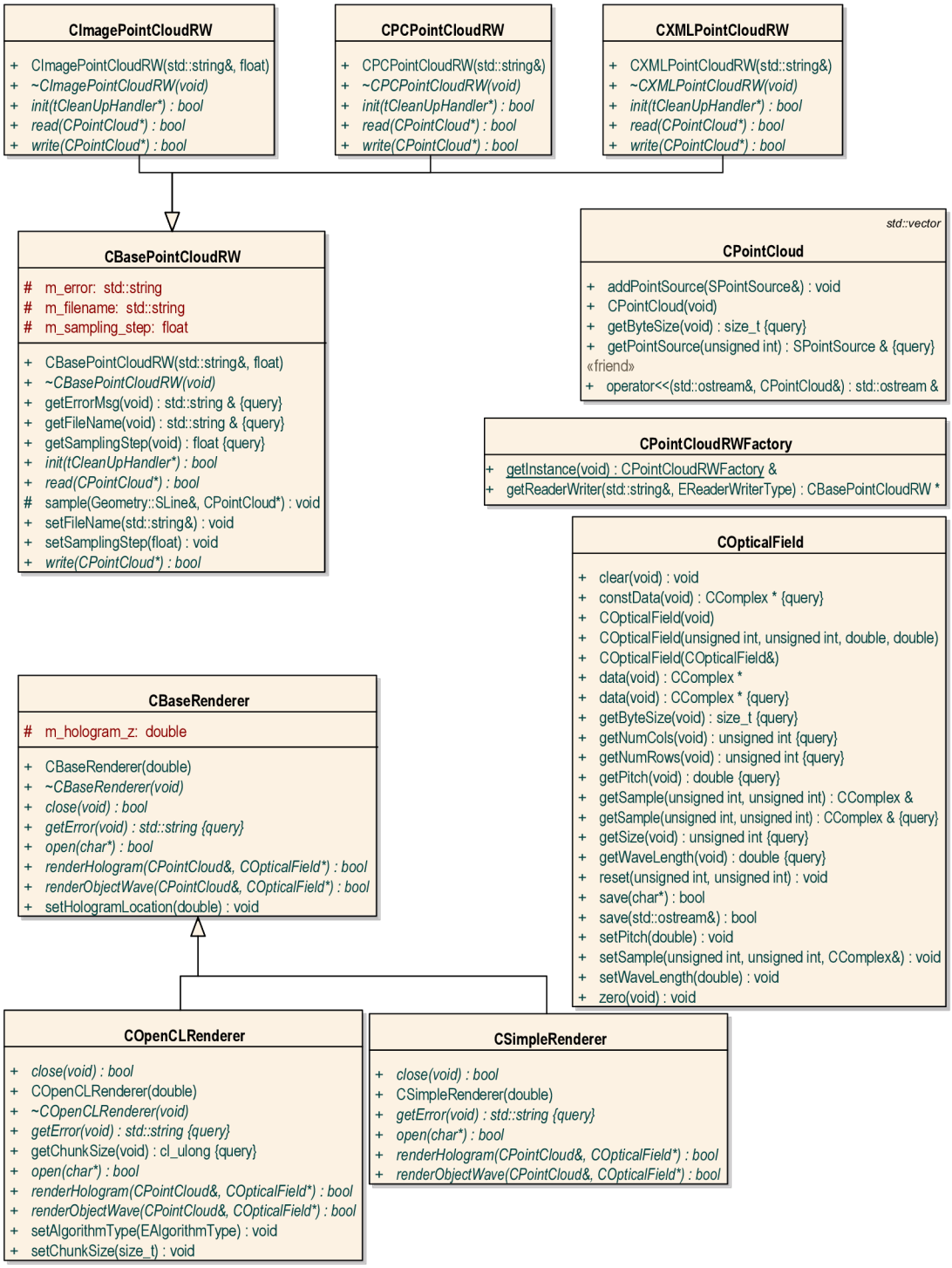
V prvej podkapitole bude predstavená štruktúra implementovanej aplikácie nasledovaná vysvetlením prevodu vstupnej reprezentácie dát na bodové zdroje. Tretia časť tejto kapitoly sa zas zameria na predstavenie spôsobu generovania hologramov a v poslednej sekcii bude prezentované vyhodnotenie získaných výsledkov.

5.1 Štruktúra aplikácie

Základnú štruktúru zhrňuje diagram 5.1 zobrazujúci desať najvýznamnejších tried implementovaného programu.

Prvými dvoma sú `CPointCloud` a `COpticalField`, ktoré slúžia ako pamäťové úložištia bodových zdrojov a vzoriek komplexných amplitúd optického poľa. V prípade `CPointCloud` v podstate ide len o obyčajný vektor známy aj zo štandardnej knižnice jazyka C++ doplnený niekoľkými dodatočnými funkciami. Jeho hlavnou úlohou je obaliť použitú implementáciu do vlastného rozhrania a získať tak väčší priestor pre manipuláciu v budúcnosti. `COpticalField` naopak okrem dvojrozmerného poľa samotných komplexných vzoriek obsahuje aj popisné metainformácie dôležité pri jeho generovaní. Konkrétne sa jedná o vlnovú dĺžku potrebnú pri výpočte vlnového čísla a fyzickú vzdialenosť medzi jednotlivými vzorkami nazvanú pitch. Ďalej je jeho súčasťou funkcia na zápis optického poľa do súboru

[1] Dostupné z http://www.kiv.zcu.cz/en/research/downloads/product-detail-en.html?produkt_id=31



Obrázek 5.1: UML diagram najdôležitejších tried v aplikácii.

vo formáte df. Tento formát je definovaný a popísaný holografickým toolkit-om zmieneným v úvode kapitoly. V podstate sa jedná sa o binárny súbor, kde na začiatku sú uložené uve-

dené metainformácie a za nimi v nekomprimovanej podobe nasledujú jednotlivé komplexné amplitúdy. Podrobnejšie informácie sú však súčasťou dokumentácie HoloToolkit-u a nebudú v tejto práci bližšie popisované.

Keďže zadávať bodové zdroje priamo by bolo značne nepraktické umožňuje program ich prevod z pohodlnejšej reprezentácie. Takouto reprezentáciou je napríklad obrázok, alebo XML súbor, v ktorom je možné pomocou geometrických primitív definovať vstupnú scénu. O prevod tejto scény na mračno bodových zdrojov (point cloud) sa potom stará jedna z trojice tried `CImagePointCloudRW`, `CPCPointCloudRW` alebo `CXMLPointCloudRW`. Každá z nich je pritom odvodená z tej istej bázovej triedy `CBasePointCloudRW`, ktorá zahŕňa definíciu spoločného rozhrania a bežne používaných funkcií a dátových členov.

Aby sa však užívateľský kód nemusel starať o celý komplikovaný proces ich inštanciácie, tak bola vytvorená trieda `CPointCloudRWFactory` reprezentujúca továreň takýchto konvertorov. Tá je navyše definovaná ako singleton, čo zabezpečuje jej okamžitú dostupnosť z ľubovoľnej časti kódu a integritu volaní samotných inicializačných a deinicializačných rutín.

V zostávajúcej časti diagramu sa tak nachádzajú už len renderovacie triedy, ktoré však tvoria samotné gro programu. Tie sú opäť všetky odvodené od spoločnej základnej triedy `CBaseRenderer`. Tento návrh vychádza z predpokladu, že v budúcnosti môže byť aplikácia rozšírená o nové metódy generovania hologramov vyžadujúce použitie odlišných technológií.

Zatiaľ sú však definované len dve triedy a to `CSimpleRenderer` a `COpenCLRenderer`. Prvá menovaná predstavuje úplne najzákladnejšiu softvérovú metódu generovania hologramu. V podstate sa jedná len o pokusnú implementáciu, na ktorej boli overené počiatočné princípy. Dôležitejšou je až druhá uvedená, ktorá využíva prostriedky OpenCL. Jej podrobný popis však bude súčasťou tretej sekcie tejto kapitoly.

Ďalej už aplikácia obsahuje viac-menej len pomocné triedy a funkcie združované do špecifických menných priestorov podľa ich určenia. Aktuálne sú definované štyri rôzne menné priestory a to `OpenCL`, `Debug`, `Geometry` a `Utils`. Prvý menovaný obsahuje kód zjednodušujúci prácu s OpenCL ako napríklad dopytovanie sa na rôzne vlastnosti OpenCL objektov, alebo výber najlepšieho dostupného zariadenia. Menný priestor `Geometry` zas definuje niekoľko základných geometrických primitív používaných pri konverzii vstupu na point cloud. Posledné dva vymenované definujú pomocné funkcie pre ladenie kódu a funkcie rôzneho druhu zamerané väčšinou na prácu so súborovým systémom a konverziu číselných typov.

5.2 Načítavanie vstupu

Načítanie vstupu predstavuje úvodnú fázu činnosti programu definovanú pomocou konverzných tried, z ktorých každá obsahuje implementáciu troch základných metód. Menovite sú to `init`, `read` a `write`.

Prvá z nich, `init`, slúži ako prostriedok implementácie nevyhnutných inicializačných krokov, ktorých vykonanie je však v programe vyžadované len raz pred zahájením práce ľubovoľného konvertoru toho istého typu. Takúto inicializáciu využíva množstvo knižníc napísaných v jazyku C, čo im dáva šancu predalokovať si statickú pamäť a nastaviť globálne dáta. Výstupom metódy je booleovská návratová hodnota indikujúca priebeh inicializácie a ukazovateľ na deštruktor, ktorý musí byť zavolaný na konci programu.

Samotné načítanie a prevod na bodové zdroje je potom súčasťou metódy `read`. Aktuálne program definuje tri rôzne podoby vstupu, z ktorých najjednoduchším je súbor s príponou pc. Jedná sa o vlastný textový formát definovaný špecificky pre účely tejto aplikácie, kde

každý riadok pozostávajúci z trojice súradníc oddelených čiarkami definuje jeden bodový zdroj. Čítanie a zápis tohto formátu má na starosti trieda `CPCPointCloudRW`.

Ďalším spôsobom ako definovať vstup je pomocou obrázku. Spracovanie tohto formátu má na starosti trieda `CImagePointCloudRW`. Tá funguje tak, že vstup načítaný knižnicou SDL_image najprv zredukuje do binárnej podoby na čiernobiely obraz a následne všetky čierne pixely pridá do `CPointCloud` ako bodové zdroje. Súradnice pôvodných pixelov sú však ešte prenásobené užívateľom zvolenou konštantou, ktorá odpovedá ich požadovanej fyzickej veľkosti. Táto konštanta teda napokon určuje aký veľký objekt bude obrázok v skutočnosti reprezentovať.

Posledným typom vstupu je XML súbor. Ten definuje niekoľko geometrických primitív, ktoré sú následne skonvertované na bodové zdroje. V súčasnosti sú týmito primitívami kváder, čiara a bod. Formát súboru i obslužná trieda sú však navrhnuté tak, aby boli jednoducho rozšíriteľné o ľubovoľný ďalší objekt. Jeho spracovanie má na starosti posledná definovaná trieda, `CXMLPointCloudRW`, ktorá pritom využíva služieb multiplatformnej knižnice libxml2.

Keďže pri čítaní a konverzii vstupnej reprezentácie dochádza k strate informácií, nie je vždy možné zapísať point cloud v pôvodnom tvare. Zapisovacia metóda je tak definovaná len pre súbory typu pc, kde rekonštrukcia pôvodného formátu nepredstavuje žiadne ťažkosti.

5.3 Generovanie hologramov

Túto fázu má na starosti trieda `COpenCLRenderer`, ktorej činnosť je rozdelená do dvoch hlavných krokov.

Prvým z nich je inicializácia odohrávajúca sa v metóde `open`. Počas tejto aktivity sú postupne vytvorené jednotlivé OpenCL štruktúry počínajúc kontextom až po samotnú frontu príkazov, ktorá následne slúži na komunikáciu so zariadením. Druhou súčasťou tohto kroku je kompilácia zdrojových kódov jazyka OpenCL C. Zaradenie tejto činnosti na začiatok programu do inicializácie umožňuje rýchlejší beh aplikácie v prípade potreby generovať viacero hologramov naraz pri jednom spustení.

Druhým krokom je samotná syntéza výsledného hologramu, ktorej však ešte tiež predchádza určitá inicializácia. Tá pozostáva najmä z alokácie pamäte potrebnej pre vstupné a výstupné dáta, ktorých veľkosť a obsah nemusia byť známe už v priebehu metódy `open`. Tie sú potom spolu s ďalšími argumentami predané kernelu, ktorý je tak pripravený na spustenie. Týmito dodatočnými argumentami sú napríklad vlnové číslo, rozpätie bodov na optickom poli, alebo súradnice ľavého horného rohu tohto poľa. Po dokončení všetkých príprav je kernel odoslaný do zariadenia funkciou `clEnqueueNDRangeKernel`, z ktorého sú potom výsledky získané späť funkciou `clEnqueueReadBuffer`.

Predošlý prístup funguje dobre, ale len do chvíle, kým veľkosť vstupných dát je menšia ako veľkosť dostupnej pamäte zariadenia. Po presiahnutí tohto limitu sa nepodarí naalokovať pamäťové objekty a výpočet sa ani nespustí. Keďže jedným z cieľov návrhu je aj schopnosť generovať optické polia o veľkosti presahujúcej 700MB je potrebné výpočet rozdeliť na niekoľko menších celkov, ktoré budú do zariadenia posielané postupne. Okrem toho, že bude takto možné generovať hologramy o prakticky ľubovoľnej veľkosti, prináša táto zmena ešte niekoľko zaujímavých výhod navyše.

Keďže grafické karty nepodporujú multitasking tak dobre ako bežné procesory, sú veľmi citlivé na dlhotrvajúce výpočty. Toto sa vo výsledku môže prejaviť zdanlivým "zamrznutím"obrazovky počítača a kurzoru myši, pričom v skutočnosti je príčinou dlhý výpočet, počas ktorého GPU nestíha spracovávať výpočty ostatných aplikácií a patrične podľa toho

obnovovať obrazovku. V mnohých operačným systémoch je tento problém riešený pomocou watchdog časovača, ktorý takéto správanie detekuje a vo vhodných okamihoch grafickú kartu reštartuje. Ďalším prejavom príliš dlhotrvajúceho výpočtu, tak môže byť neočakávane skoré ukončenie výpočtu na strane OpenCL aplikácie.

Spomínané rozdelenie výpočtu, tak pomáha predísť týmto problémom. Kedže však každé zariadenie môže mať iné parametre, tak `COpenCLRenderer` dovoľuje užívateľovi definovať vlastnú veľkosť dát spracovaných v jednej dávke. Ak táto voľba nie je špecifikovaná, je jej hodnota odvodená z maximálnej možnej veľkosti buffer objektu.

Ďalším drobným vylepšením, ktoré však pridalo znateľný nárast výkonu bolo použitie vstavaných funkcií s prefixom `native_`. Tie fungujú podľa OpenCL špecifikácie v princípe rovnako ako ich štandarné ekvivalenty, len s tým rozdielom, že u nich nie je požadovaná žiadna konkrétna presnosť. To potom v zásade znamená, že ich implementácia môže byť optimalizovaná na jedinú hardvérovú inštrukciu z čoho aj ťažia svoju fenomenálnu rýchlosť.

5.4 Testovanie a vyhodnotenie

Prvá časť testovania predstavovala overenie vypočítaných výsledkov k čomu boli využité prostriedky ponúkané holografickým toolkitom spomenutým už v úvode k tejto kapitole.

Ten okrem množstva ďalších nástrojov disponuje aj programom nazvaným HoloPropagLarge, ktorý umožňuje vypočítať propagáciu medzi dvoma paralelnými rovinami v priestore. Tento výpočet sa však dá mimo iného využiť i na numerickú rekonštrukciu vygenerovaného hologramu.

Činnosť HoloPropagLarge je riadená XML konfiguračným súborom, kde sú zapísané informácie o vlnovej dĺžke a rozpätí medzi vzorkami propagovaného optického poľa. Ďalším podstatným parametrom je voľba operations. Tá umožňuje prostredníctvom jednoduchého interpretovaného proprietárneho jazyka založeného na podobnom princípe ako PostScript kontrolovať proces propagácie. Z pohľadu tejto práce je však dôležitá len operácia `prop`, ktorou sa pole propaguje do zvolenej vzdialenosti od roviny hologramu. Výsledkom je potom obrázok vo formáte png, z ktorého sa dá približne odhadnúť ako by vyzeral výsledok v skutočnosti počas optickej rekonštrukcie.

Keďže výstupom aplikácie vytvorenej vrámci tejto práce je súbor vo formáte df, je potrebné ho ešte prekonvertovať na obrázok, ktorý môže slúžiť ako vstup pre utilitu HoloPropagLarge, alebo byť použitý na výrobu fyzického hologramu. O túto konverziu sa stará ďalší nástroj z HoloToolKit-u nazvaný DFtoHologram.

Aby sa dala práca s uvedenými externými aplikáciami aspoň čiastočne zautomatizovať a uľahčiť si tak testovanie, bola pomocou frameworku Qt napísaná jednoduchá aplikácia HTConfigManager. Tá umožňuje z grafického prostredia rýchlo editovať ich nastavenia, zariadiť spustenie a následne zobraziť konečné výsledky. Snímky vygenerovaných hologramov a ich rekoštrukcií sú postupne uvedené na obrázkoch 5.2, 5.3, 5.4 a 5.5. Ako predloha pre generovanie bolo zvolené logo FIT VUT v Brne.

Druhou súčasťou testovania bolo meranie výkonu implementovaného riešenia, počas ktorého boli porovnávané tri rôzne verzie programu. Zatiaľ čo prvá na svoj beh využívala procesor dostupný prostredníctvom OpenCL, zvyšné dve už ťažili z plne paralelizovaného spracovania v grafickej karte. Rozdiel medzi nimi spočíval najmä v použití natívnych (s prefixom `native_`) inštrukcií GPU. Pôvodne bola do testu zaradená aj naivná softvérová implmentácia, ktorej výkon však bol neporovnateľne nižší ako vo všetkých ostatných prípadoch a preto nie je v celkových výsledkoch zohľadnená.

Testovanie zahŕňalo päť opakovaných spustení programu, počas ktorých bol pre väčšiu objektivitu započítavaný len samotný čas behu kernelu, čiže inicializácia OpenCL štruktúr a kompilácia OpenCL C programu nie sú vo výsledkoch uvažované. Dôvod tohto rozhodnutia spočíva vo fakte, že sa viac-menej jedná o konštantnú časť aplikácie, ktorej dĺžka trvania je nezávislá od veľkosti počítaného vstupu. S narastajúcou zložitosťou a počtom bytov prekladaného zdrojového kódu sa síce bude zvyšovať aj čas kompilácie, ale OpenCL dovoľuje jednoducho získať preložené binárne súbory späť a umožniť tým ich efektívne cache-ovanie pred opätovným spustením programu. Vo výsledku tak za prepokladu, že spracovanie vstupných dát bude trvať dlhšie než doba inicializácie, nemá príliš veľký praktický význam ju do porovnania zahŕňať.

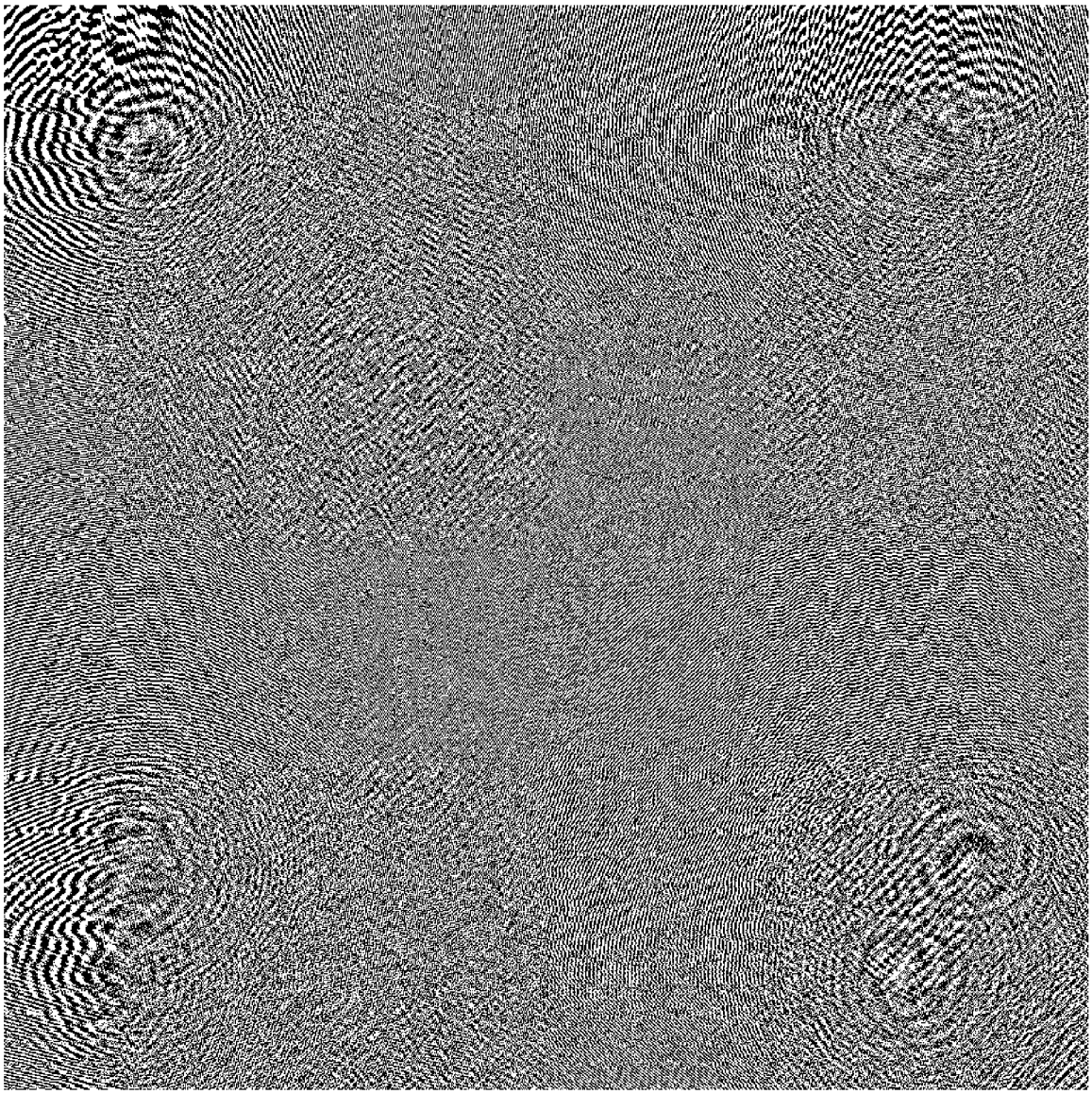
Vďaka obmedzeniu závislostí na výlučne platformovo neutrálne riešenia je program preložiteľný a spustiteľný v prostredí Windows i v prostredí Linux. Konkrétne bol testovaný na 64-bitovej verzii Windows 7, ktorý bol zároveň aj primárnou vývojovou platformou a na 32-bitovej a 64-bitovej verzii Ubuntu. Z dostupných kompilátorov bol preklad otestovaný prekladačom gcc verzie 4.8 a v prostredí Windows prekladačmi dodávanými s vývojovými prostrediami Microsoft Visual Studio 2010 a Microsoft Visual Studio 2012.

Z pohľadu hadrvéru sa testy odohrávali na notebook-u značky Acer so 4GB operačnej pamäte, procesorom Intel Core i5 a grafickou kartou ATI Mobility Radeon HD 5650. Z dostupných OpenCL implementácií bolo nainštalované SDK od Intel-u aj Accelerated Parallel Processing od AMD.

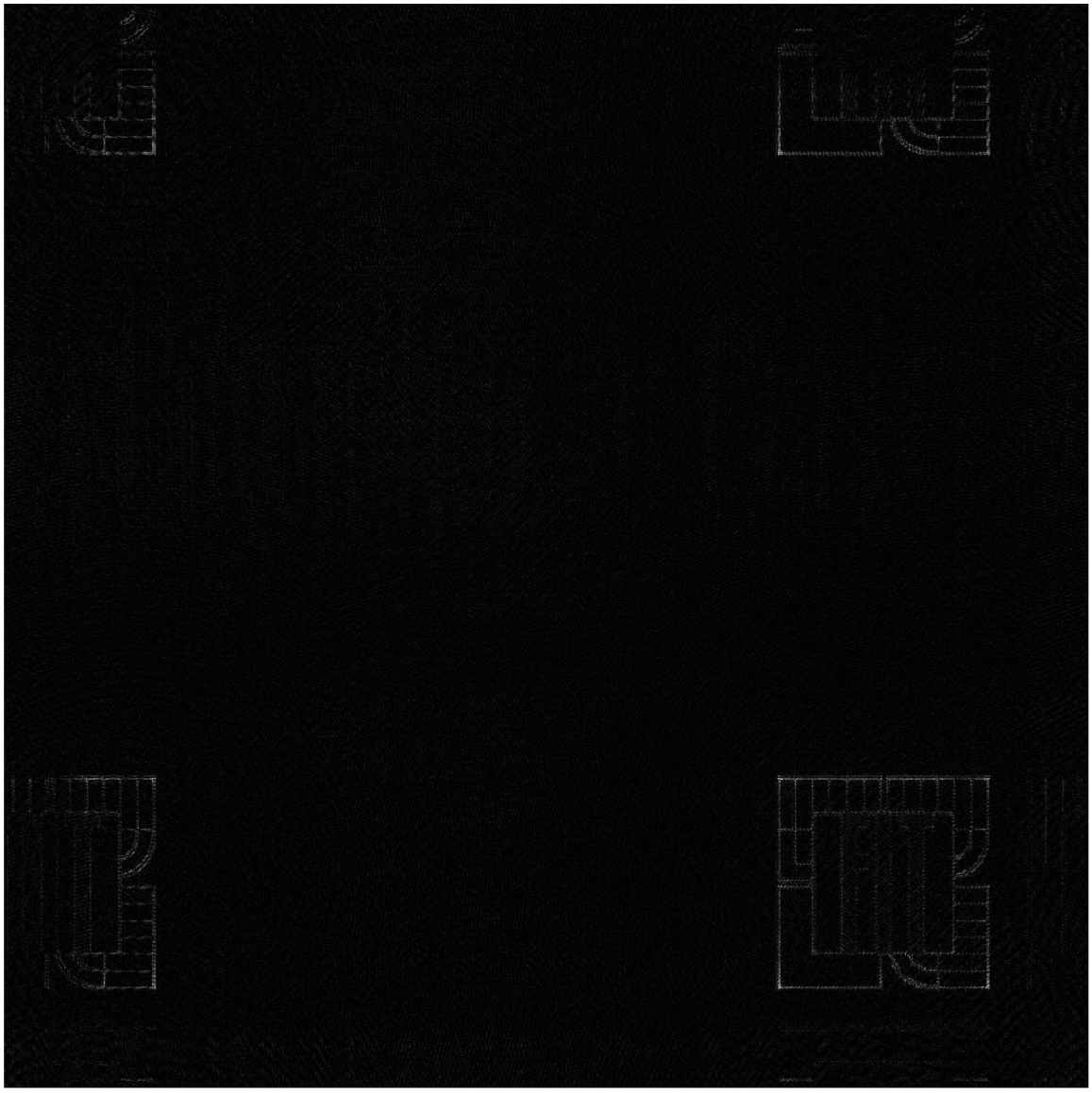
	Windows 7 64-bit	**Ubuntu 13.04 64-bit**
OpenCL CPU	8min 19s	4min 20s
GPU	9s	15s
GPU s nativnymi inštrukciami	2s	3s

Tabulka 5.1: Porovnanie priemernej rýchlosti jednotlivých verzií aplikácie

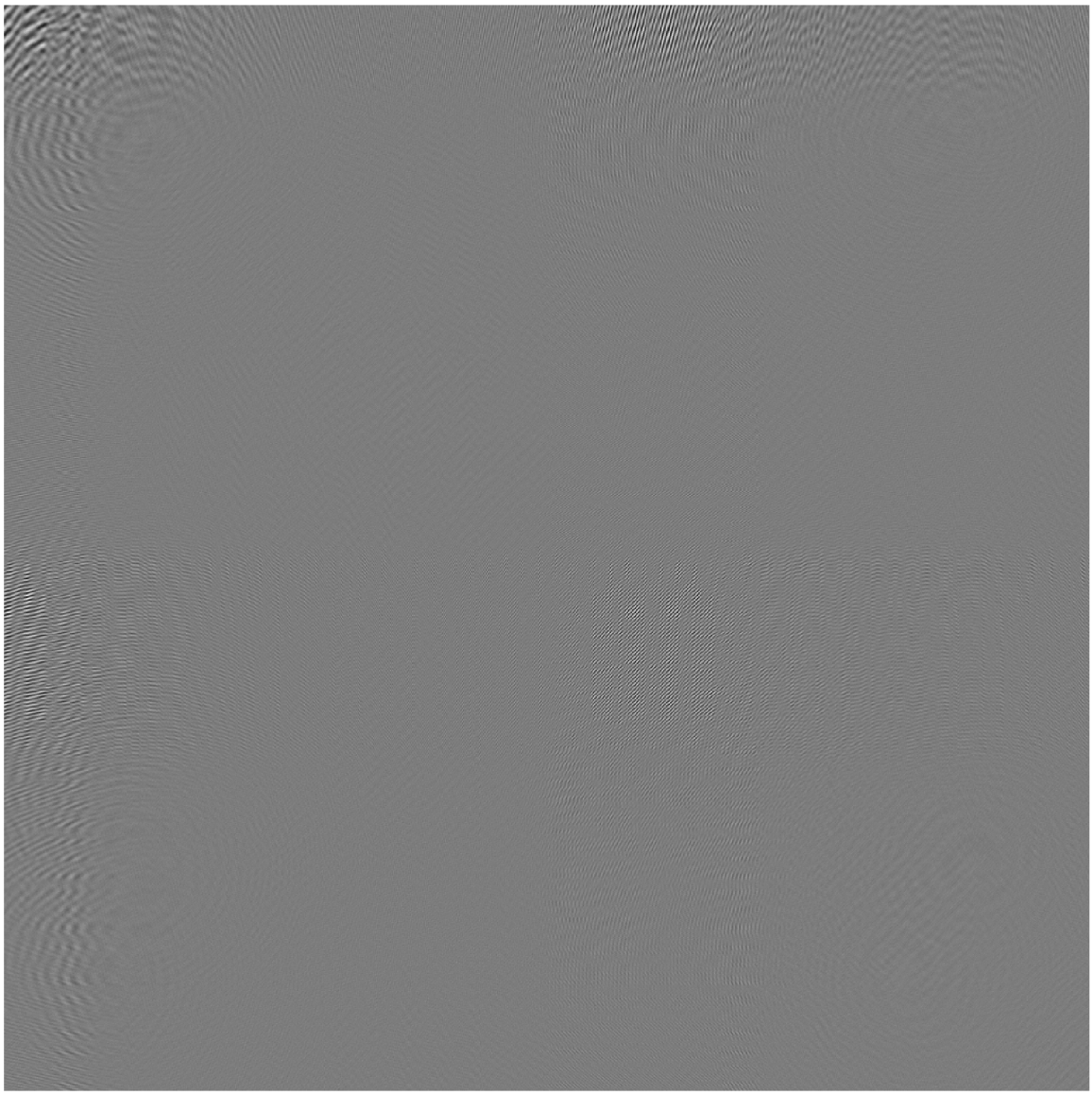
Tabuľka 5.1 tak sumarizuje priemerné časy výpočtu namerané na obidvoch spomenutých systémoch. Z nej sa dá konštatovať, že v súlade s predpokladmi je najrýchlejšou verziou posledná vyžívajúca akceleráciu natívnymi inštrukciami. Zaujímavý je tiež rozdiel medzi jednotlivými CPU verziami, kde program na Ubuntu dosiahol v priemere dva-krát vyšší výkon ako na Windows. Príčinu tohto priepastného rozdielu sa však nepodarilo zistiť. Jedným z možných vysvetlení je fakt, že v prípade Ubuntu sa jednalo o čistú inštaláciu s minimom ďalších aplikácií, čo vo výsledku zrejme ovplynilo aj rozdelenie procesorového času medzi jednolivé programy v systéme. Rozdiel medzi operačnými systémami v prípade GPU je pravdepodobne spôsobený odlišnými verziami ovládačov a OpenCL SDK.



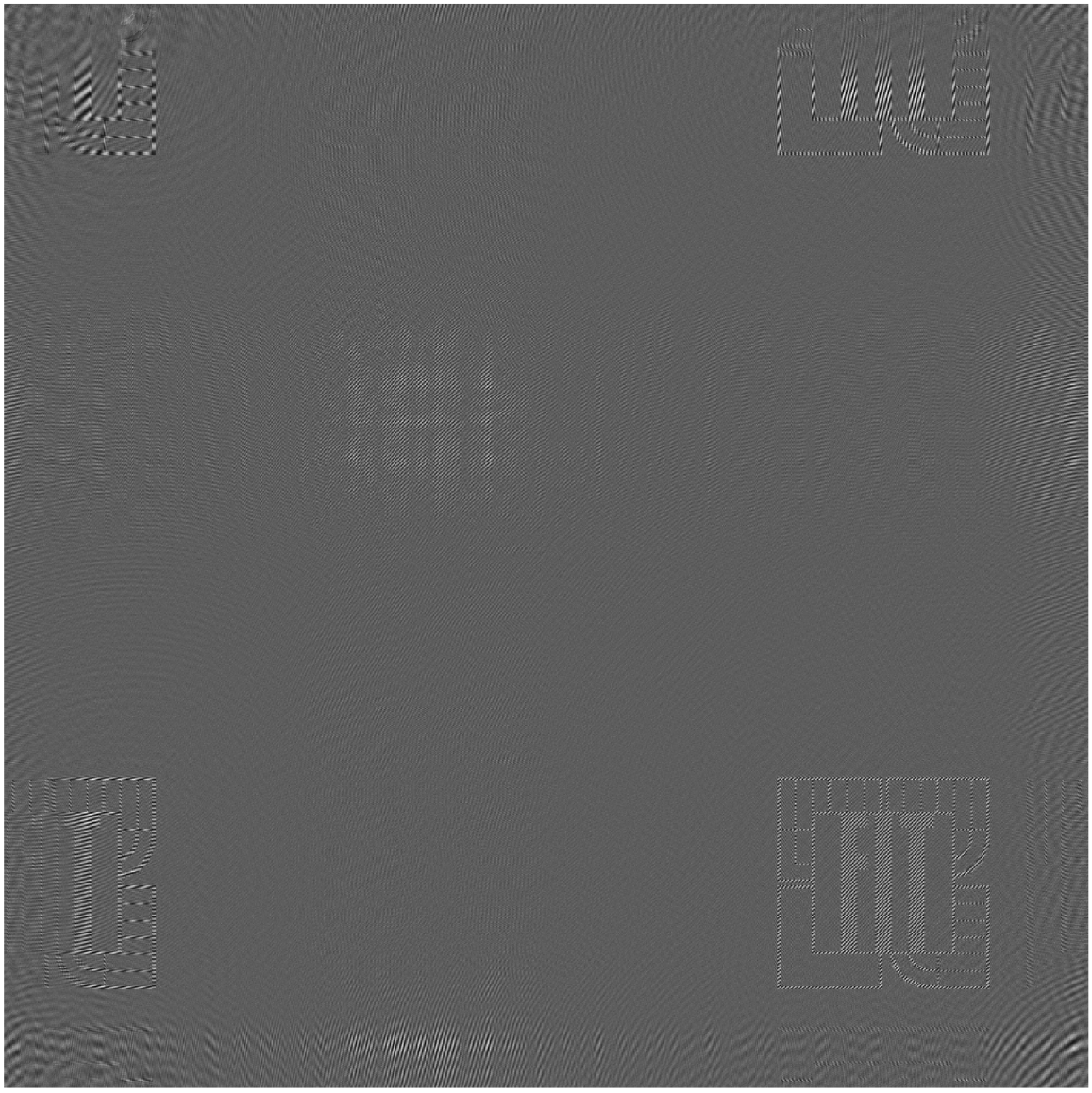
Obrázek 5.2: Binárny hologram loga FIT VUT v Brne. Hologram je v rozlíšení 1024x1024 pixelov vytvorený pri vlnovej dĺžke svetla 630nm a rozpätí vzoriek optického poľa 20µm



Obrázek 5.3: Rekonštrukcia vytvorená z binárneho hologramu loga FIT VUT v Brne.



Obrázek 5.4: Grayscale hologram loga FIT VUT v Brne. Hologram je v rozlíšení 1024x1024 pixelov vytvorený pri vlnovej dĺžke svetla 630nm a rozpätí vzoriek optického poľa 20µm



Obrázek 5.5: Rekonštrukcia vytvorená z grayscale hologramu loga FIT VUT v Brne.

Kapitola 6

Záver

Cieľom práce bolo pomocou dostupných technológií pre programovanie GPU akcelerovať výpočet syntetických hologramov.

Myslím, že v práci sa mi podarilo splniť všetky hlavné body zadania počnúc naštudovaním základných princípov až po konečnú implementáciu vybraného algoritmu. Tým bola metóda Diffraction integral evaluation, čiže výpočet difrakčného integrálu. Na akceleráciu tejto metódy som využil prostriedky pomerne nového, avšak stále populárnejšieho štandardu OpenCL. Posledná časť práce bola venovaná overeniu výsledkov zvolenej metódy. Hoci numerické simulácie potvrdili správnosť vytvorenej implementácie, je škoda, že z časových dôvodov sa nepodarilo zrealizovať aj optickú rekonštrukciu, ktorá by mohla odhaliť dodatočné skryté nedostatky a priniesť množstvo nových skúseností.

Výsledný program je navrhnutý tak, aby dokázal spracovať prakticky ľubovoľne veľké optické pole. Okrem menších polí, prezentovaných napríklad v kapitole o testovaní, bol vygenerovaný aj hologram s rozmermi 10000x10000 vzoriek optického poľa, pričom naň dopadalo svetlo z takmer 50000 bodových zdrojov. Spolu sa tak jednalo o približne päť triliónov potrebných výpočtov, pričom pamäťové nároky na uloženie vstupných a výstupných údajov predstavovali niečo vyše `760` MB. Na bežnom približne 3 roky starom notebook-u so 4GB dostupnej operačnej pamäte a grafickou kartou ATI Mobility Radeon HD 5650 trval výpočet menej ako 2 hodiny.

Práca mi teda ukázala, že holografia je zaujímavá oblasť a čaká ju veľká budúcnosť. Aj keď sa tak možno nestane v najbližšej dobe, tak pri dnešnom tempe vývoja a raste výkonu počítačov si časom určite nájde svoje miesto medzi technológiami realistického 3D zobrazovania.

Ako jedno z možných rozšírení práce v budúcnosti vidím implementáciu niektorej z pokročilejších metód syntézy. Takouto metódou by mohol byť napríklad algoritmus ray-casting, pomocou ktorého by sa následne dali plnohodnotne generovať aj hologramy komplexných priestorových scén. Ďalším prípadným rozšírením by mohlo byť využitie iných paralených technológií ako sú napríklad distribuované výpočty, alebo implementácia v hardvéri pomocou FPGA.

Literatura

[1] Two-Slit Experiment Approximation.svg. Wikibooks [online]. 12 May 2009.
URL http://commons.wikimedia.org/wiki/File:Two-Slit_Experiment_Approximation.svg

[2] Compute Shader Overview. MICROSOFT. Microsoft Developer Network [online]. 2013 [cit. 2013-05-10].
URL http://msdn.microsoft.com/en-us/library/ff476331%28v=VS.85%29.aspx

[3] CUDA. NVIDIA. NVidia [online]. 2013 [cit. 2013-05-10].
URL http://www.nvidia.com/object/cuda_home_new.html

[4] MPICH [online]. 2013 [cit. 2013-05-10].
URL http://www.mpich.org/

[5] Alisi [online]. ©2009 [cit. 2013-05-13].
URL http://alisi.isibrno.cz/

[6] Ceitec VUT [online]. ©2009 [cit. 2013-05-13].
URL http://www.ceitec.vutbr.cz/index.php

[7] Goodman, J. W.: *Introduction to Fourier Optics*. Roberts & Company Publishers, třetí vydání, 2005, ISBN-13 978-0974707723.

[8] Hanák, I.: *Urychlení výpočtu číslicového hologramu*. Dizertační práce, University of West Bohemia, November 2009.

[9] Hariharan, P.: *Basics of holography*. Cambridge University Press, 2002, ISBN-13 978-0-521-80741-8.

[10] Janda, M.: Digital Hologram Synthesis. Technická Zpráva DCSE/TR-2007-02, Department of Computer Science and Engineering, University of West Bohemia, Univerzitni 8, 30614 Pilsen, Czech Republic, April 2007.

[11] Janda, M.; Hanák, I.; Skala, V.: Holography Principles. Technická Zpráva DCSE/TR-2006-08, Department of Computer Science and Engineering, University of West Bohemia, Univerzitni 8, 30614 Pilsen, Czech Republic, December 2006.

[12] Khronos OpenCL Working Group and Aaftab Munshi: The OpenCL Specification Version: 1.1 Document Revision: 44 [online]. 2011 [cit. 2013-05-07].
URL http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf

[13] Lobaz, P.: Holografie: cesta k dokonalému displeji. 2012, prednáška z predmetu PGR (Pokročilá počítačová grafika).

[14] Reichl, J.; Všetička, M.: Encyklopedie fyziky [online]. © 2006 - 2013 [cit. 2013-05-14]. URL http://fyzika.jreichl.com/main.article/view/431-optika

[15] Scarpino, M.: *OpenCL in Action*. Manning Publications Co., 2012, ISBN 9781617290176.