Excel@FIT 2015
http://excel.fit.vutbr.cz

Progressive photon mapping na GPU

Tomáš Lysek*

Abstrakt

Pro tvorbu fotorealistických obrazů je nutné použít časově náročné výpočetní techniky - global illumination techniky. Mezi nejnovější techniky patří photon mapping a jeho progresivní varianta. Tento článek se bude zabývat implementací progressive photon mappingu (ppm) na dnešních grafických kartách a porovná rychlost implementace mezi CPU variantou a GPU variantou. Dále vybere nejpomalejší blok ppm a navrhne urychlení. Navrhnuté urychlení je zclustrování hitpointů do blízkých clusterů a použití pravidelné mřížky pro photony.

Klíčová slova: progressive photon mapping — global illumination — gpgpu

Přiložené materiály: *N/A*

*xlysek03@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Od začátku počítačové grafiky byla snaha o co nejrealističtější výsledné snímky. Pro co nejrealističtější obrazy je potřeba provést složitou fyzikální simulaci světla, která je velmi časově náročná. Poměrně novou fotorealistickou metodou je metoda fotonových map (photon mapping) a jeho nejnovější varianta progressive photon mapping. Poslední dobou nastal velký boom GPGPU počítání, grafickou kartu má v dnešní době skoro každý PC a proto se zdá jako zajmavá varianta vyzkoušet akcelerovat progressive photon mapping na grafické kartě. Tato práce nejprve navrhne jednoduchou gpu implementaci progressive photon mappingu a poté pro nejpomalejší blok navrhne urychlení. Nejpomalejší blok byl vybrán a tento blok se pomocí jednoduché akcelerační struktury podařilo urychlit oproti CPU implementace více než 30 krát.

2. Related work

Nejlepší fotorealistické výsledky se v dnešní době dosahují pomocí třídy technik, které se nazývají global illumination (globální osvětlení). Ne všechny global illumination techniky jsou rozumně implementovatelné na GPU a proto je potřeba vybrat vhodnou techniku. Tyto metody se snaží o korektní simulaci světla ve scéně a základním kamenem pro tyto metody byla zobrazovací rovnice (rendering equation), kterou v roce 1986 sepsal James T. Kajiya[1].

Zobrazovací rovnice popisuje výpočet odraženého jasu (radiance) v určitém bodě. Popisuje to jako součet (integrál) všech příspěvků osvětlení ze všech směrů přes hemisféru kolem vyšetřovaného bodu. Pomocí tohoto přístupu je možné analyticky vypočítat osvětlení scény, problém ovšem nastává s tím, že pro rozumně složité scény je toto řešení nerealizovatelné - z časových důvodu.

V článku o zobrazovací rovnici James T. Kajia[1] narvhnul způsob jak vypočítat tuto rovnici a to tím způsobem, že pro výpočet integrálu použil monte carlo metody. Tato metoda se označuje jako path tracing a všechny metody, které vychází z path tracingu se nazývají monte carlo raytracing metody.

Path tracing funguje podobně jako raytracing, vyšle se paprsek z oka (kamery), který putuje do scény. Při průsečíku paprsku se scénou v bodě x se vyšle sekundární paprsek náhodným směrem a ten se vyšetří. Protože náhodný paprsek nedokáže vypočítat přesně osvětlení bodu x vyšle se více paprsku z daného pixelu a prošetří více náhodných cest - odtud název path tracing.

Rozšířením path tracnigu byla v roce 1993 [2] a v roce 1994 [3] technika zvaná bidirectional path tracing. Tato technika prohledává cesty jak od kamery

tak od světla najednou a poté vypočte jaký příspěvek mají obě dvě cesty mezi sebou a podle toho spočte osvětlení. Dalším rozšířením path tracingu je technika zvaná Metropolis

Problém monte carlo metod je ten, že pro dobré fotorealistické výsledky je potřeba projít mnoho cest každým pixelem. V literatuře se píše, že běžné číslo jsou tisíce cest na pixel, z čehož vyplývá obrovská časová náročnost. Další možností je použít novou techniku zvanou Photon mapping. Photon mapping byl vymyšlen v Henrikem Wann Jensenem v roce 1996 [4].

Photon mapping je dvou-průchodová technika, která v prvním kroce spočítá příspěvek světla ve scéně vzorkováním světla. Jeden vzorek světla - foton - je prošetřen scénou podobně jako paprsek u raytracingu a při dopadu na difúzní povrch je foton uložen do fotonové mapy. Poté v druhém průchodu, se při výpočtu lokálního osvětlovacího modelu vypočte nepřímé osvětlení z nejbližších fotonů uložených ve fotonové mapě.

Pro dobré výsledky pomocí klasického photon mappingu je potřeba vyslat mnoho (desítky milonů) fotonů ze světelných zdrojů, což má za následek velkou mapu fotonů - z toho plyne velká paměťová složitost - a navíc čím víc fotonů je uložených ve fotonové mapě, tím pomalejší vyhledávání ve fotonové mapě je.

Problém s velikosti fotonové mapy se snaží řešit progresivní varianta photon mappingu - progressive photon mapping, která vyšle ze světelného zdroje pouze část fotonů, pomocí této části spočítá osvětlení, vyslané fotony zahodí a v další iteraci vyšle znova menší část fotonů, přičte osvětlení a takhle pořád dokola.

Progressive photon mapping

Progressive photon mapping se jeví jako jako zajmavá metoda pro implementaci na grafické kartě - malá paměťová složitost, iterační výpočet, velká koherence. Proto tato sekce rozepíše teorii za progressive photon mappingem trochu do hloubky.

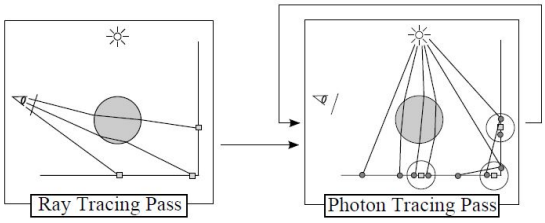
V klasickém photon mappingu se jas v bodě x a směru $\vec{\omega}$ spočítá jako:

$$L_r(x,\vec{\omega}) \approx \frac{1}{\pi r^2}\sum_{p=1}^{N} f(x,\vec{\omega},\vec{\omega}_p)\Delta\Phi_p(x,\vec{\omega}_p) \quad (1)$$

kde $\Delta\Phi_p(x,\vec{\omega}_p)$ je záře z fotonu p uložena ve fotonové mapě a $f(x,\vec{\omega},\vec{\omega}_p)$ je BRDF funkce. Suma zahrnuje N nejbližších fotonů z bodu x ve fotonové mapě.

Jak již bylo zmíněno, problém photon mappingu je v jeji časové a paměťové složitosti. Při výpočtu je photon mapping limitován operační pamětí PC a při velké fotonové mapě je prohledávání taky pomalejší. Právě proto byla snaha rozdělit jednu velkou fotonovou mapu na mnoho menších fotonových map a z nich spočítat osvětlení. Jednou jednoduchou variantou bylo vypočíst n-krát klasický foton mapping s menším počtem fotonů a poté výsledky jednotlivých iterací zprůměrovat, toto řešení bylo rychlejší ale nevedlo k konzistentnímu (správnému) výsledku.

Další možnost byla vymyšlena právě pomocí progressive photon mappingu, která vhodně připočítá osvětlení dané iterace k předcházející iteraci.



Obrázek 1. Schéma progressive photon mappingu[5]

Obrázek 1 zobrazuje schéma progresive photon mappingu. Nejprve je nad scénou vypočítán raytracing, v průsečících paprsků s difúzním povrchem je uložený bod - tzv. hitpoint. V hitpointu je uložena pozice, barva, souřadnice ve výsledném obrazku, poloměr a počet fotonů, které přispěly k osvětlení daného hitpointu.

Druhá část je rozdělena do iterací, kde se vždy vyšle určitá část fotonů, v každém hitpointu se nalezou všechny fotony v jeho poloměru, vypočte se podle nich příspěvek osvetlení, progresivně se připočte k již vypočtenému osvětlení. Pote se vypočítané fotony zahodí a celý proces vysílání fotonů se opakuje (teoreticky) donekonečna.

Poloměr v hitpointu by se měl v každé iteraci zmenšovat, což vede k zpřesňování již vypočteného řešení, nový poloměr se vypočítá jako:

$$\hat{R}(x) = R(x)\sqrt{\frac{N(x)+\alpha M(x)}{N(x)+M(x)}} \quad (2)$$

kde $R(x)$ respektive $\hat{R}(x)$ je poloměr v hitpointu x respektive nový poloměr v hitpointu x. $N(x)$ je počet již uložených fotonů (z předcházejících iterací) v hitpointu x, $M(x)$ je počet nových fotonů ze zrovna počítané iteraci v poloměru $R(x)$. α je hodnota v rozsahu 0 - 1 a udává jak moc nových fotonů bude přidáno do výsledného osvětlení a jak rychle se bude poloměr zmenšovat.

Nová hodnota osvětlení $\tau_{\hat{N}}(x,\vec{\omega})$ v bodě x je spočtena jako:

$$\tau_{\hat{N}}(x,\vec{\omega}) = \tau_{N+M}(x,\vec{\omega})\frac{N(x)+\alpha M(x)}{N(x)+M(x)} \qquad (3)$$

kde $\tau_{N+M}(x,\vec{\omega})$ je suma jasu z předchozích iterací a z nynější iterace spočtena rovnicí 1 a hodnoty $N(x), M(x)$ a α jsou stejné hodnoty jako v rovnici 2.

Výsledné osvětlení v bodě x směřující směrem $\vec{\omega}$ je spočteno jako:

$$L(x,\vec{\omega}) \approx \frac{1}{\pi \hat{R}(x)^2}\frac{\tau_{\hat{N}}(x,\vec{\omega})}{N_{emmited}} \qquad (4)$$

Použitím progressive photon mappingu je navíc možné docílit některých fotorealistických elementu, které jsou velmi složité u ostatních metod globálního osvětlení, jako například kaustiky, SDS cesty a jiné. (SDS cesty jsou z názvu specular-diffuse-specular a klasická ukázka těchto cest jsou například kaustiky na dně bazénu).

Jednoduchá GPGPU dekompozice

Progressive photon mapping může být rozdělen do několika nezávislých bloků: raytracing, photon tracing, hitpoint illumination a syntéza obrazu.

Raytracing

Raytracing může být chápán jako sekvenční procházení pixelů výsledného obrazu a prohledávání scény pomocí patřičných paprsků. Procházecí rutina je pro všechny paprsky stejná, takže paralelizace je velmi jednoduchá - jedno vlákno - jeden pixel. Oproti klasickému raytracingu, tento raytracing je rozšířen o uložení hitpointu do paměti.

Jedna věc musí být poznamenána, a to ta, že raytracing je v základní podobě rekurzivní algoritmus. V dnešní době grafické karty neumí zpracovat rekurzivní volání funkce a proto je potřeba upravit raytracing aby fungoval iteračně.

Photon tracing

Blok photon tracingu je velmi podobný raytracingu, neukládá ovšem hitpointy, ale ukládá fotony. Pro procházení fotonového paprsku je použit velmi podobná rutina jako u raytracingu.

Hitpoint illumination

V tomto bloku se paralelně pro každý hitpoint vypočte nové osvětlení hitpointu x. Nejjednodušší (a taky nejnaivnější) varianta je projít všechny fotony na daném objektu a ty které mají vzdálenost od bodu x menší než $R(x)$ se zahrnou do výpočtu.

Syntéza obrazu

Poslední blok načte paralelně hitpoint, vypočte barvu podle rovnice 4 a uloží spočtenou barvu na pozici ve framebufferu.

3. Evaluace jednoduché implementace

Byla provedena jednoduchá implementace jednotlivých bloků. I když většina bloků je velmi naivních, tak tato implementace slouží pouze k nalezení nejslabších míst photon mappingu. Tato implementace byla otestovaná na jednoduché scéně a obrázek 2 ukazuje postupné - progresivní - zlepšování výsledné kvality obrazu.

Byla také provedena CPU implementace, která byla rozšířena o rychlejší hledání fotonů pomocí kd-stromu. CPU a GPU implementace byla evaluována na notebooku s Intel i7-4702MQ processoru, napsána v c++ a kompilována Intel C++ 15.0 kompilátorem. GPU na které běžela OpenCL implementace byla spuštěna na grafice GeForce GT 750M. Rychlost CPU implementace byla měřena pomocí std::chrono knihovny a GPU implementace pomocí nvidia nsight.

	GPU	CPU
Raytracing	0.706 ms	1172 ms
Photon tracing	7.897 ms	317 ms
Hitpoint Illumination	2041 ms	1593 ms
Synthesize	0.247 ms	3 ms

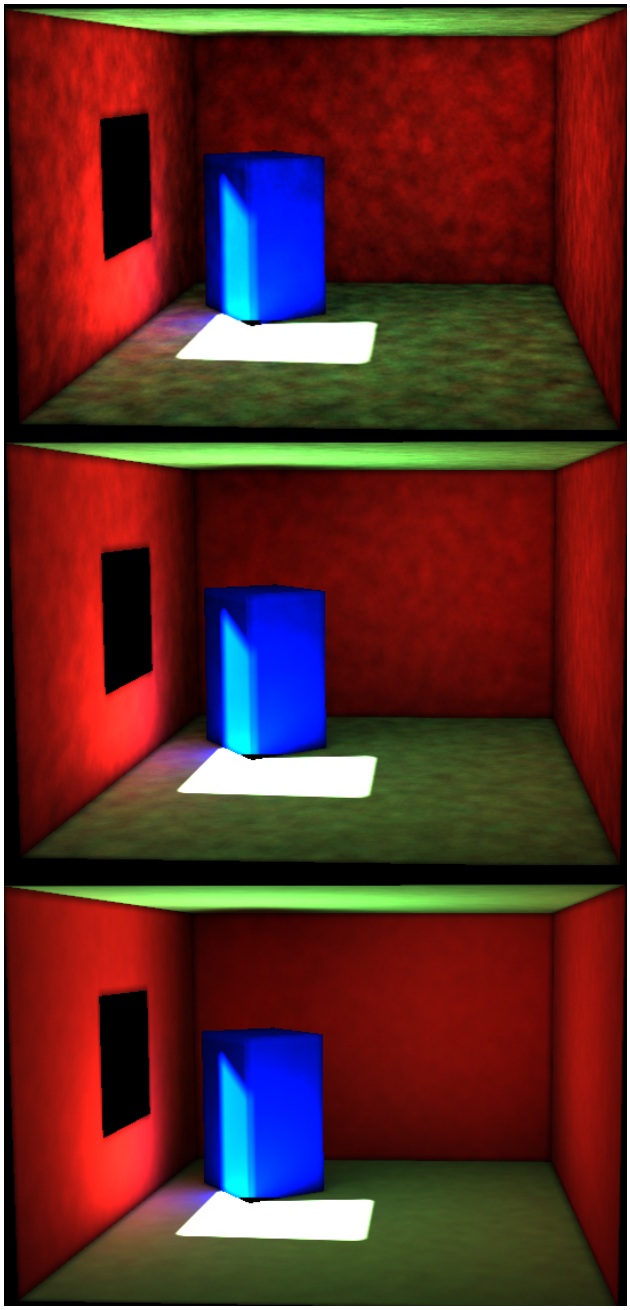
Tabulka 1. Výkonost mezi GPU a CPU variantou algoritmu při rozlišení 320*280 a při velikosti fotonové mapy 100 tisíc fotonů.

	GPU	CPU
Raytracing	12.107 ms	6 197 ms
Photon tracing	7.897 ms	317 ms
Hitpoint Illumination	38 603 ms	23 957 ms
Synthesize	4.924 ms	91 ms

Tabulka 2. Výkonost mezi GPU a CPU variantou algoritmu při rozlišení 1280*1080 a při velikosti fotonové mapy 100 tisíc fotonů.

Tabulky 1 a 2 ukazují výkonost mezi GPU a CPU implementací. Jak je možné z těchto tabulek vidět, všechny bloky kromě Hitpoint illumination se na GPU značně urychlily. Jednu věc je potřeba podotknout, na CPU funguje v bloku Hitpoint illumination vyledávání pomocí kd-stromu, pokud se na cpu použije stejná metoda - tedy bruteforce - tak při malém rozlišení 320*280 trvá tento blok 68 sekund.

Rychlost raytracingu a photon tracingu je ovlivněna počtem pixelů / fotonů a složitostí scény. I když byla testovací scéna velmi jednoduchá, photon tracing a raytracing nejsou nejslabší místa. Navíc pro komplexnější scény existují sofistikované algoritmy a datové struktury, pomocí kterých se dá dosáhnout vysoká výkonost



Obrázek 2. Výsledky postupného počítání progressive photon mappingu. Na vrchním obrázku je výsledek s jednou iterací, uprostřed s deseti iteracemi a dole se sto iteracemi.

i při komplexních scénách a pokud by se netvořil jednoduchý renderer, tak je možné použít nvidia OptiX framework, který má zabudovaný a vysoce optimalizovaný raytracing, který by se na tyto dva bloky dal použít.

Hitpoint illumination blok je nejpomalejší blok. Je to dáno tím, že používaná technika je velmi naivní a tento blok testuje velké množství fotonů, které nejsou vůbec (dáno topologií scény) potřeba. Blok vybraný k akceleraci je teda hitpoint illumination blok. To že je scéna velmi jednoduchá v tomto případě vůbec nevadí, protože i v mnohem komplexnější scéně se řeší naprosto stejné problémy a časově (pokud se zachová počet fotonů a rozlišení) budou stejné.

4. Návrh akcelerace

Zmiňovaný hitpoint illumination blok je naprogramovaný velmi naivně. Tato sekce popíše postupně jaké akcelerační techniky se navrhly, použily a byly stavební bloky pro nejrychlejší řešení.

Lokální pamět

V naivním řešení každý thread načítal postupně z globální paměti jeden foton za druhým. Přístup do globální paměti je na GPU velmi drahý, proto první akcelerační technika je využití lokální paměťi, do které je mnohem rychlejší přístup.

V každé workgroupě se připraví lokální paměť a přetáhne se blok globální paměti do lokální paměti. Všechny thready v workgroupě zpracují všechny fotony v lokální paměti a poté se načte další blok paměti a tak pořád dokola.

Řazení fotonů

Když se počítá osvětlení hitpointu x, tak se do výpočtu osvětlení musí zařadit pouze ty fotony, které leží na stejném objektu jako hitpoint x. Proto když máme jediné pole fotonů, tak velká část fotonů se zavrhne, protože leží na jiném objektu a drahé načítání bylo zbytečné.

Navrhnuté řešení je rozdělit jedno pole fotonů na N polí fotonů, kde v každém poli jsou fotony pouze z jednoho objektu. Proto se mezi photon tracing blok a hitpoint illumination blok vloží blok na rozřazení fotonů a tento blok roztřídí fotony do příslušných polí. Rychlost tohoto bloku je zanedbatelná (1.5ms) v porovnání s rychlostí bloku hitpoint illumination.

Protože není zajištěno, že všechny hitpointy v jedné workgroupě budou ze stejného objektu, není možné použít v tomto případě lokální paměť.

Řazení hitpointů

Aby se dosáhla větší koherence, je nutné zajistit, že všechny hitpointy v jedné workgroupě budou z jednoho objektu, je nutné použít nějaký systém rozřazení.

Toto rozřazení je možné provést na CPU, protože bude v celém procesu použité pouze jednou. Jediný problém, který nastává je při přechodu (v poli) mezi jednotlivými objekty. Je nutné vložit do pole s hitpointy takzvané filler hitpointy, které jsou neaktivní a zajišťují pouze to, že v workgoupě budou hitpointy pouze z jednoho objektu, nebo právě filler hitpointy.

Řazení hitpointů a lokální paměť

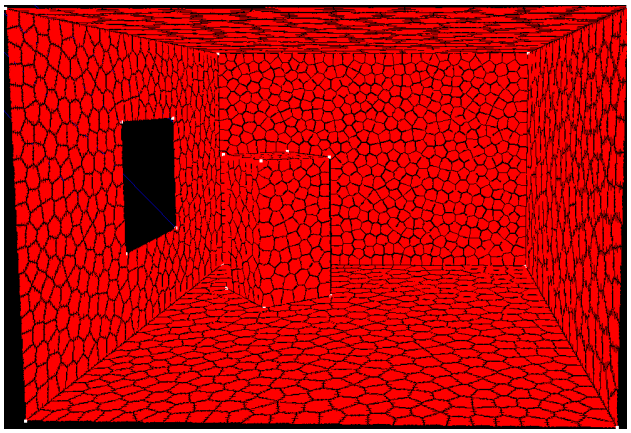
Protože nyní jsou v jedné workgroupě pouze hitpointy z jednoho objektu, je možné sloučit všechny předchozí přístupy a použít lokální paměť a rozřazené fotony najednou.

Klastrování hitpointů a spatial grid

Předchozí řešení i když je rychlé, pořád prohledává velké množství zbytečných fotonů. Když se podíváme na CPU implementaci pomocí kd-stromu, jde vidět, že CPU akcelerační technika na vyhledávání fotonů přinesla obrovské zrychlení, proto by bylo vhodné použít podobnou akcelerační techiku i na GPU.

Kd-strom je ovšem těžko sestrojitelný za běhu na GPU. Proto vybrané řešení je pravidelná mřížka. Při tvorbě pravidelné mřížky pro jeden objekt, se nejprve vypočte bounding box daného objektu. Pro jednoduchost a adaptabilitu se využije dělící faktor n, který určuje kolikrát se bude dělit nejdelší strana bounding boxu a z jednoho dílku se poté vypočte šířka pravidelné mřížky.

Při určování v jaké části mřížky se bod x nachází, stačí pouze vypočíst vektor směřující z minimálního bodu bounding boxu do bodu x a poté podělit každou složku šířkou jednoho boxu pravidelné mřížky. Tímto po zaokrouhlení směrem dolů vypadnou celočíselné hodnoty x, y, z, které udávají pozici v mřížce.



Obrázek 3. Klastry hitpointů vytvořené algoritmem k-means

Posledním problémem, který je potřeba vyřešit, je maximální koherence hitpointů, což znamená, že je nutné zajistit, aby hitpointy v jedné workgroupě byly co nejblíže (prostorově) u sebe a tímpádem budou procházet stejné podprostory mřížky. Pro toto zajištění je využito na CPU straně algoritmu k-means, který vytvoří na poli hitpointů klastry. Velký problém algoritmu k-means je ten, že není možné zajistit aby vracel klustry fixní velikosti a tedy mnoho filler hitpointů je použito. Obrázek 3 znázorňuje jak vypadají klastry ve scéně. Velikost klastrů je omezena podle velikosti workgroupy a nejefektivnější velikost worgkroupy se může lišit dle rozlišení.

5. Vyhodnocení výsledků

Všechny akcelerační techniky, které byly představené v minulé sekci byly naimplementovány a byla změřena jejich rychlost na rozlišení 1920*1080 a 320*280.

	320*280	1920*1080
Naive solution	2041 ms	38 603 ms
Local memory	1109 ms	17 750 ms
Photon sorting	611 ms	10 539 ms
Hitpoint sorting	484 ms	8 384 ms
Hitpoint sorting, local memory	327 ms	5 422 ms

Tabulka 3. Vyhodnocení výsledku navrhnutých akceleračních řešení.

Tabulka 3 ukazuje, že postupně všechny postupy bez pravidelné přinášely žádané urychlení a postupně se podařilo překonat a získat zrychlení oproti CPU variantě.

Všechny tyto testy byly vykonány pomocí OpenCL. Pro výpočet vzdálenosti byla použita zabudovaná funkce distance(). Jakmile se funkce distance() nahradila za ruční vypočítání vzdálenosti, vedlo to k skoro 3x rychlejšímu výsledku. A tedy rychlost hitpoint illumination bloku je při rozlišení 320*280 125 ms a při rozlišení 1920*1080 1 941ms.

Testování pravidelné mřížky je nutné provést odděleně a to z důvodu velkého množství filler hitpointů. Při rozlišení 1920*1080 ve výsledném výpočtu bylo přítomno 33% filler hitpointů a tedy algoritmus fungoval pouze na 66%. Je nutné tedy uvést časy i bez pravidelné mřížky. Tabulka 4 ukazuje finální a maximální zrychlení navrhovaných metod.

	320*280	1920*1080
Hitpoint sorting, local memory	172 ms	2 852 ms
Grid structure	87 ms	730 ms

Tabulka 4. Rychlost pravidelné mřížky.

6. Závěr

Tato práce popisuje experimentální implementaci progresivního photon mappingu na grafické kartě. Pro každý dekomponovaný blok je navrhnuta gpu implementace a nejpomalejší blok byl postupně zrychlený 52 krát oproti naivní implementaci a 32krát oproti CPU implementaci.

V dalším pokračování práce by bylo vhodné vymyslet jiný algoritmus klastrování, například pomocí evolučních algoritmů navrhnout takové rozřazení hitpointů, které sníží chybu z 33% na menší hodnotu a využije se větší potenciál grafické karty.

Poděkování

Chtěl bych poděkovat vedoucímu mé diplomové práce profesorovi Pavlovi Zemčíkovi za užitečné rady a trpělivost, kterou se mnou měl.

Literatura

[1] James T. Kajiya. The rendering equation. *SIGGRAPH Comput. Graph.*, 20(4):143–150, August 1986.

[2] Eric P. Lafortune and Yves D. Willems. Bidirectional path tracing. 1993.

[3] Eric Veach and Leonidas Guibas. Bidirectional Estimators for Light Transport. 1994.

[4] Henrik Wann Jensen. Global illumination using photon maps. In *Proceedings of the eurographics workshop on Rendering techniques '96*, pages 21–30, London, UK, UK, 1996. Springer-Verlag.

[5] Toshiya Hachisuka, Shinji Ogaki, and Henrik Wann Jensen. Progressive photon mapping. *ACM Trans. Graph.*, 27(5):130:1–130:8, December 2008.