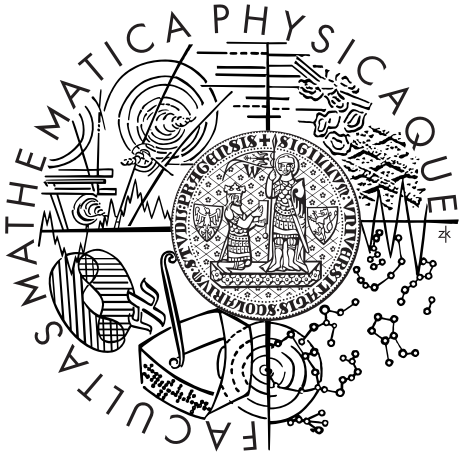
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Martin Bulant

GPU implementace algoritmů irradiance a radiance caching

Kabinet software a výuky informatiky

Vedoucí diplomové práce:	Ing. Jaroslav Křivánek, Ph.D.
Studijní program:	Informatika
Studijní obor:	Softwarové systémy

Praha 2015

Na tomto místě bych chtěl poděkovat lidem, bez kterých by vytvoření této práce nebylo možné a to především vedoucímu práce Ing. Jaroslavu Křivánkovi, Ph.D. za mnoho užitečných rad při psaní práce. Dále Ing. Tomáši Davidovičovi za zapůjčení frameworku pro implementaci zadaných algoritmů.

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Název práce: GPU implementace algoritmů irradiance a radiance caching

Autor: Martin Bulant

Katedra: Kabinet software a výuky informatiky

Vedoucí diplomové práce: Ing. Jaroslav Křivánek, Ph.D. , Kabinet software a výuky informatiky

Abstrakt: Předmětem této práce je vytvořit program implementující algoritmy pro výpočet globálního osvětlení radiance a irradiance caching za využití frameworku CUDA na GPU. Paralelní implementace na GPU by měla přinést výrazné zrychlení oproti sériové implementaci na CPU. Implementace bude probíhat v již existujícím frameworku pro výpočty globálního osvětlení, což umožní soustředit se čistě na implementaci samotných algoritmů. Díky této práci bude možné ušetřit čas při testování metod pro výpočet globálního osvětlení, neboť ukládání a přepoužití mezivýsledků je možné použít pro další algoritmy.

Klíčová slova: počítačová grafika, rendering, globální osvětlení, irradiance caching, radiance caching, GPU, CUDA

Title: GPU implementation of the irradiance and radiance caching algorithms

Author: Martin Bulant

Department: Department of Software and Computer Science Education

Supervisor: Ing. Jaroslav Křivánek, Ph.D. , Department of Software and Computer Science Education

Abstract: The object of this work is to create software implementing two algorithms for global ilumination computing. Iradiance and radiance caching should be implemented in CUDA framework on graphics card (GPU). Parallel implementation on GPU should dramatically improve algoritm speed compared to CPU implementation. The software will be written using already done framework for global illumunation computation. That allow to focus to algorithm implementation only. This work should speed up testing of new or existing methods for global illumination computing, because saving and reusing of intermediate results can be used for other algorithms too.

Keywords: computer graphics, rendering, global illumination, irradiance caching, radiance caching, GPU, CUDA

Obsah

Úvod

Vykreslování je dle mého názoru samou podstatou počítačové grafiky a přesně do této oblasti spadají zadané algoritmy. V současné době existuje mnoho způsobů, jak přistoupit k řešení tohoto problému z nichž největší zastopuení mají 2. Jsou to sledování paprsku a rasterizace. Přičemž v algoritmech, které se snaží o co největší fyzikální věrnost jsou zastoupeny především ty, které využívají sledování paprsku. Bohužel z obou těchto největších kategorií zrovna ty méně pasují na aktuální grafický hardware, který vychází z videoherního průmyslu, kde nejde ani tak o co největší přesnost, ale spíš o to, aby to vypadalo co nejlépe. Proto je zajímavým úkolem snažit se napasovat algoritmy využívající raytracing na grafický hardware.

V první kapitole podrobně rozeberu oba algoritmy a popíši jak fungují. Ovšem ještě předtím velmi stručně popíši problematiku samotného vykreslování, zmíním základní veličiny, které budou pro následující práci důležité a jaký mají vztah k vykreslování. Zmíním, že jde o algoritmy, které mají na starost výpočet nepřímého osvětlení a to, že oba využívají takzvané mezipaměti (angl. cache) pro uložení některých výsledků, které jsou v pozdější fázi algoritmu opět použity. To sice přináší jistou systematickou chybu, nicméně tuto chybu je možné měřit a parametry algoritmů nastavit tak, že se tato chyba vejde pod určitou předem zadanou mez.

Druhá kapitola bude věnována HW na kterém budu dané algoritmy implementovat a především frameworku CUDA, který to umožňuje. Rozeberu základní rozdíly mezi CPU a GPU a zmíním, proč zrovna GPU je pro dané výpočty vhodné. Dále se zmíním o způsobech, kterým lze s CUDA frameworkem pracovat, konkrétně příslušná API, půjde o běhové a ovladačové API. Rozeberu způsoby jak se s oběma rozhraními pracuje a jaké mají výhody a nevýhody. Zmíním co je to jádro, k čemu jsou vlákna a jak se organizují do bloků případně tabulky bloků. Rozeberu způsoby předávání dat mezi vlákny, jednotlivé druhy pamětí a způsoby synchronizace.

Ve třetí kapitole se budu věnovat popisu mé implementace a rozeberu její jednotlivé části. Konkrétně to, jak je algoritmus rozdělen do jednotlivých jader a jak se využívá uspořádání vláken do bloků případně tabulek bloků.

Poslední kapitola poslouží ke shrnutí získaných výsledků. Zmíním časové náročnosti obou algoritmů a porovnám je s případem, že by se osvětlení ve scéně počítalo pro každý vzorek skrz obrazový bod. Současně s tím ukážu jaké chyby algoritmy způsobují a jak jsou závislé na parametrech těchto algoritmů.

1. Irradiance a radiance caching

V této kapitole se budu zabývat renderováním, tedy převodem 3D scény z matematického popisu do obrázku. Toho se dá docílit mnoha různými způsoby, jímž odpovídají různé algoritmy, já se budu zabývat dvěma z nich, konkrétně irradiance a radiance cachingem.

1.1 Vykreslování

Vykreslování (rendering) je převod popisu scény do obrázku. Scéna se skládá ze samotné geometrie v ní obsažených objektů, jejich materiálů, zdrojů světla a kamery, kterou se do scény "díváme". Jak je v počítačové grafice běžné, obrázek se skládá z jednotlivých obrazových bodů(pixelů). Cílem vykreslování je tedy zjistit kolik světla se dostane ze světelných zdrojů ve scéně do konkrétních pixelů v obrázku. Světlo je elektromagnetické záření, jehož měřením se zabývá věda zvaná radiometrie případně fotometrie, která se zabývá pouze částí viditelnou lidským okem.

1.1.1 Radiometrie a Fotometrie

Definici pro radiometrii a fotometrii uvádí wikipedie [1]: "Radiometrie je část optiky, která se zabývá měřením elektromagnetického záření, včetně světla. Radiometrie se zabývá absolutními veličinami, zatímco fotometrie studuje obdobné veličiny, avšak z hlediska jejich působení na lidské oko."

Pro naše účely si z radiometie vezmeme následující definice, které využívá ve své knize i Křivánek [2] a kterým odpovídají fyzikální veličiny Tok, Ozáření a Zář.

Tok

Zářivý tok Φ, měřený ve Wattech [W], je celkové množství zářivé energie procházející zkrze plochu o kterou se zajímáme za jednotku času. Platí pro něj vzorec:

$$\Phi = \frac{\mathrm{d}Q}{\mathrm{d}t}.$$

Ozáření

Ozáření E, měřené ve $\mathrm{W.m}^{-2}$, je příchozí tok na jednotku plochy. Tedy dle vzorce:

$$E(\mathbf{p}) = \frac{\mathrm{d}\Phi(\mathbf{p})}{\mathrm{d}A}.$$

Ozáření popisuje prostorovou hustotu záření přicházejícího na plochu a je nezávislé na tom odkud záření přichází. Hodnota ozáření závisí na orientaci plochy $\mathrm{d}A$. To znamená, že k ozáření plochy $\mathbf{p}$ přispívá pouze záření přicházející do bodu $\mathbf{p}$ z horní polokoule H^+.

Zář

Zář L, měřená v $\mathrm{W.m^{-2}.sr^{-1}}$, je zářivý tok na promítnutou jednotku plochy a na prostorový úhel. Platí pro ní vzorec:

$$L(\mathbf{p},\omega) = \frac{\mathrm{d}^2\Phi(\mathbf{p},\omega)}{\mathrm{d}A\mathrm{d}\omega\cos\theta} = \frac{\mathrm{d}E(\mathbf{p},\omega)}{\mathrm{d}\omega\cos\theta},$$

kde $\mathrm{d}\omega$ je diferenciál prostorového úhlu a θ je úhel mezi normálou plošky a směrem ω. Člen $\cos\theta$ dělá zář nezávislou na orientaci plošky $\mathrm{d}A$ a směru šíření toku ω jelikož ji promítá do směru ω.

Ozáření lze vyjádřit pomocí záře následující rovnicí:

$$E(\mathbf{p}) = \int_{H+} L(\mathbf{p},\omega)\cos\theta\mathrm{d}\omega, \tag{1.1}$$

kde $H+$ je horní polokoule se středem v bodě $\mathbf{p}$ vzhledem k normále v tomto bodě.

1.2 Zobrazovací algoritmy

1.2.1 Vzorkování polokoule

$$E(\mathbf{p}) \approx \frac{1}{MN}\sum_{j=0}^{M-1}\sum_{k=0}^{N-1}\frac{f(\theta_{j,k},\phi_{j,k})}{p(\theta_{j,k},\phi_{j,k})},$$

kde f je integrand a p PDF pro výběr daného vzorku. V našem případě $f(\theta,\phi) = L(\theta,phi)\cos\theta$ a $p(\theta,\phi) = \frac{\cos\theta}{\pi}$, z čehož pak plyne výsledný vzorec pro odhad ozáření:

$$E(\mathbf{p}) \approx \frac{\pi}{MN}\sum_{j=0}^{M-1}\sum_{k=0}^{N-1}L_{j,k}, \tag{1.2}$$

kde $L_{j,k}$ je vzorek příchozí záře získaný sledováním paprsku ve směru:

$$(\theta_{j,k},\phi_{j,k}) = \left(\arccos\sqrt{1-\frac{j+\xi^1_{j,k}}{M}}, 2\pi\frac{k+\xi^2_{j,k}}{N}\right). \tag{1.3}$$

Po převodu do kartézských souřadnic vypadají rovnice takto

$$\begin{aligned}
x_{j,k} &= \sin\theta_{j,k}\cos\phi_{j,k} &&= \sqrt{\frac{j+\xi^1_{j,k}}{M}}\cos 2\pi\frac{k+\xi^2_{j,k}}{N}\\
y_{j,k} &= \sin\theta_{j,k}\cos\phi_{j,k} &&= \sqrt{\frac{j+\xi^1_{j,k}}{M}}\sin 2\pi\frac{k+\xi^2_{j,k}}{N}\\
z_{j,k} &= \cos\theta_{j,k} &&= \sqrt{1-\frac{j+\xi^1_{j,k}}{M}}
\end{aligned}$$

$\xi^1_{j,k}, \xi^2_{j,k}$ jsou náhodná čísla rovnoměrně rozdělená v intervalu $[0,1)$, M je počet dělení polokoule dle úhlu θ a N je počet dělení dle úhlu ϕ.

1.2.2 Rotační gradient

Jak uvádí Křivánek [2], tak rotační gradient lze získat ze stejné sady vzorků $L_{j,k}$, ze které se počítá ozáření v daném bodě pomocí stratifikovaného vzorkování. K tomu se použije následující vzorec:

$$\nabla_r E \approx \frac{\pi}{MN} \sum_{k=0}^{N-1} \left(\mathbf{v}_k \sum_{j=0}^{M-1} - \tan\theta_j \cdot L_{j,k} \right) \tag{1.4}$$

, kde $\mathbf{v}_k$ je vektor ve výchozí rovině ve směru $\phi_k + \frac{\pi}{2}$, (θ_j, ϕ_k) jsou sférické souřadnice středu buňky na polokouli se souřadnicemi (j,k), $L_{j,k}$ je vzorek příchozí záře vypočítaný sledováním paprsku skrz buňku se souřadnicemi (j,k).

1.2.3 Translační gradient

V knize [2] je kromě rotačního gradientu uveden též vzorec pro gradient translační:

$$\begin{aligned} \nabla_t E \approx \sum_{k=0}^{N-1} \Bigg[\mathbf{u}_k \frac{2\pi}{N} \sum_{j=1}^{M-1} \frac{\cos^2\theta_{j_-} sin\theta_{j_-}}{\min\{r_{j,k}, r_{j-1,k}\}} \left(L_{j,k} - L_{j-1,k}\right) + \\ \mathbf{v}_{k_-} \sum_{j=0}^{M-1} \frac{\cos\theta_j \left(\cos\theta_{j_-} - \cos\theta_{j_+}\right)}{\sin\theta_{j,k} \min\{r_{j,k}, r_{j,k-1}\}} \left(L_{j,k} - L_{j,k-1}\right) \Bigg] \end{aligned} \tag{1.5}$$

Ve vzorci výše mají proměnné následující význam. (j,k) jsou souřadnice buňky,

$L_{j,k}$ je vzorek příchozí záře vypočítaný sledováním paprsku skrz buňku se souřadnicemi (j,k),

$r_{j,k}$ je vzdálenost prvního průsečíku paprsku vedeného skrz buňku se souřadnicemi (j,k) se scénou,

θ_{j_-} je úhel odpovídající rozhraní mezi aktuální buňkou se souřadnicemi (j,k) a předchozí buňkou se souřadnicemi $(j-1,k)$ a je možné ho vypočítat jako

$$\theta_{j_-} = \arccos\sqrt{1 - \frac{j}{M}},$$

θ_{j_+} je úhel odpovídající rozhraní mezi aktuální buňkou se souřadnicemi (j,k) a následují buňkou se souřadnicemi $(j-1,k)$ a je možné ho vypočítat jako

$$\theta_{j_-} = \arccos\sqrt{1 - \frac{j+1}{M}},$$

ϕ_{k_-} je úhel odpovídající rozhraní mezi aktuální buňkou se souřadnicemi (j,k) a předchozí buňkou se souřadnicemi $(j,k-1)$ a je možné ho vypočítat jako

$$\phi_{k_-} = 2\pi\frac{k}{N},$$

ϕ_k je úhel odpovídající středu aktuální buňky se souřadnicemi (j,k) a je možné ho vypočítat jako

$$\phi_k = 2\pi\frac{k+0{,}5}{N},$$

ϕ_{k_+} je úhel odpovídající rozhraní mezi aktuální buňkou se souřadnicemi (j, k) a následující buňkou se souřadnicemi $(j, k+1)$ a je možné ho vypočítat jako

$$\phi_{k_+} = 2\pi \frac{k+1}{N},$$

$\mathbf{u}_k$ je jednotkový vektor ve výchozí rovině ve směru $(\pi/2, \phi_k)$,
$\mathbf{v}_{k_-}$ je jednotkový vektor ve výchozí rovině ve směru $(\pi/2, \phi_{k_-} + \pi/2)$.
Oba vzorce pro výpočet gradientů 1.4 a 1.5 vypočítávají příslušný gradient v místním systému souřadnic.

1.2.4 Interpolace

Pro výpočet ozáření v bodě $\mathbf{p}$ se využívá podmožina E_i všech záznamů ozáření z mezipaměti. Ozáření se vypočítá jako jejich vážený průměr dle vzorce:

$$E(\mathbf{p}) = \frac{\sum_{i \in S(\mathbf{p})} E_i(\mathbf{p}) w_i(\mathbf{p})}{\sum_{i \in S(\mathbf{p})} w_i(\mathbf{p})}, \tag{1.6}$$

kde $E_i(\mathbf{p})$ je vzorek ozáření v mezipaměti extrapolovaný do $\mathbf{p}$. K tomu lze použít buď přímo uloženou hodnotu, pak platí $E_i(\mathbf{p}) = E_i$ nebo je možné pomocí rotačních a translačních gradientů vylepšit přesnost extrapolace za využití následujícího vzorce:

$$E_i(\mathbf{p}) = E_i + (\mathbf{n}_i \times \mathbf{n}) \cdot \nabla_r Ei + (\mathbf{p} - \mathbf{p}_i) \cdot \nabla_t E_i, \tag{1.7}$$

kde ∇_r je rotační gradient a ∇_r gradient translační. Váhové koeficienty pro interpolaci definuje Křivánek [2, p. 27] jako inverzi odhadu chyby pro model rozdělené koule, jeho odvození lze nalézt ve stejné knize. Počítají se dle vzorce:

$$\omega_i(\mathbf{p}) = \frac{1}{\frac{\|\mathbf{p}-\mathbf{p}_i\|}{R_i} + \sqrt{1 - \mathbf{n} \cdot \mathbf{n}_i}} - \frac{1}{a} \tag{1.8}$$

kde:
$\mathbf{p}$ je bod ve kterém interpolujeme,
$\mathbf{n}$ je kolmice k tečné rovině v $\mathbf{p}$,
$\mathbf{p}_i$ je pozice i-tého uloženého záznamu,
$\mathbf{n}_i$ je kolmice k tečné rovině v $\mathbf{p}_i$
R_i je vzdálenost k povrchům viditelným z bodu p_i (vypočítaná jako harmonický průměr nebo jako minimum z délek paprsků při vzorkování polokoule uložené ve vyrovnávací paměti.
a je konstanta definovaná uživatelem určující povolenou chybu aproximace.
Pomocí funkce pro výpočet vah 1.8 je snadné určit množinu všech záznamů v mezipaměti, které se mohou přispět k hodnotě v bodě $\mathbf{p}$ takto:

$$S(\mathbf{p}) = \{i;\ \omega_i(\mathbf{p}) > 0\}. \tag{1.9}$$

Ze všech vzorků, které splňují výše uvedenou podmínku je třeba ještě vyřadit ty vzorky, které se nacházejí na ploškách se stejnou normálou jako normála v bodu $\mathbf{p}$ do kterého interpolujeme. Pro takové případy uvádí Křivánek[2] následující vzorec:

$$d_i(\mathbf{p}) = (\mathbf{p} - \mathbf{p}_i) \cdot (\mathbf{n} + \mathbf{n}_i)/2, \tag{1.10}$$

kde $\mathbf{p}$ je bod ve kterém aktuálně interpolujeme a $\mathbf{n}$ je jeho normála, $\mathbf{p}_i$ je kandidát pro interpolaci a $\mathbf{p}_i$ je jeho normála. Pokud je v tomto vzorci $d_i(\mathbf{p})$ menší než jistá malá negativní hodnota, tak je bod $\mathbf{p}_i$ vyřazen z interpolace.

Křívánek[2] zmiňuje ještě následující vzorce pro možný alternativní výpočet váhových koeficientů:

$$\omega_i(\mathbf{p}) = 1 - \kappa \max\{\varepsilon_{pi}(\mathbf{p}), \varepsilon_{ni}(\mathbf{n})\} \tag{1.11}$$

$$\varepsilon_{pi}(\mathbf{p}) = \frac{\|\mathbf{p} - \mathbf{p}_i\|}{R_i/2} \tag{1.12}$$

$$\varepsilon_{ni}(\mathbf{n}) = \frac{\sqrt{1 - \mathbf{n} \cdot \mathbf{n}_i}}{\sqrt{1 - \cos 10^\circ}} \tag{1.13}$$

1.2.5 Vzdálenosti k povrchům

Vzdálenost k povrchům viditelným z bodu, pro který je záznam v mezipaměti se počítá z délek paprsků, užitých pro výpočet nového záznamu do cache, při vzorkování hemisféry. Hodnota se dle modelu rozdělených polokoulí tak jak ho uvádí křivánek[2] počítá harmonickým průměrem, jeho vzorec je následující:

$$R_i^{HMD} = \frac{MN}{\sum_{j,k} \frac{1}{r_{j,k}}}, \tag{1.14}$$

kde MN je celkový počet paprsků při vzorkování hemisféry. Stejně jako tomu bylo i u váhových koeficientů, i zde [2] uvádí alternativní vzorec:

$$R_i^{\min} = \min_{j,k} r_{j,k} \tag{1.15}$$

2. CUDA

Manuál [3] říká, že CUDA je platforma pro paralelní výpočty a model programování vyvinutý společností NVIDIA. Umožňuje výrazné zvýšení výpočetního výkonu s využitím výkonu grafického procesoru (GPU). Při jejím vývoji bylo myšleno zejména na:

- Přidání pouze malého množství rozšíření do standartních programovacích jazyků, jako je C, které umožňují přímočarou implementaci paralelních algoritmů. S CUDA C/C++ se mohou programátoři soustředit na úkol paralelizace algoritmů, místo toho, aby trávili čas jejich implementací.
- Podporu různorodých výpočtů, ve kterých aplikace využívají jak CPU tak GPU. Sériové části se vykonávají na CPU, zatímco paralelní části se přesunout na GPU. Díky tomu může být CUDA postupně aplikována na existující aplikace. To je umožněno i tím, že CPU i GPU jsou brána jako oddělená zařízení s vlastními paměťovými prostory a díky tomu mohou výpočty běžet na obou zařízeních naráz.

V manuálu [3] je dále popsán hlavní rozdíl mezi GPU a CPU. Je to především to, že CPU obsahuje pouze jednotky výpočetních jader, kdežto GPU jich obsahuje stovky. CUDA (Compute unified device architecture)

2.1 Proč používat CUDA framework

Jak se uvádí v příručce ke CUDA frameworku [4], díky požadavkům trhu na stále realističtější a lépe vypadající výstupy grafických aplikací roste hrubý výkon grafických procesorů (GPU) mnohem rychleji, než je tomu u procesorů klasických (CPU). Tento trend je znázorněn na obrázku 2.1. Je z toho patrné, že výkon současných grafických čipů je několikanásobně vyšší než je tomu u CPU.

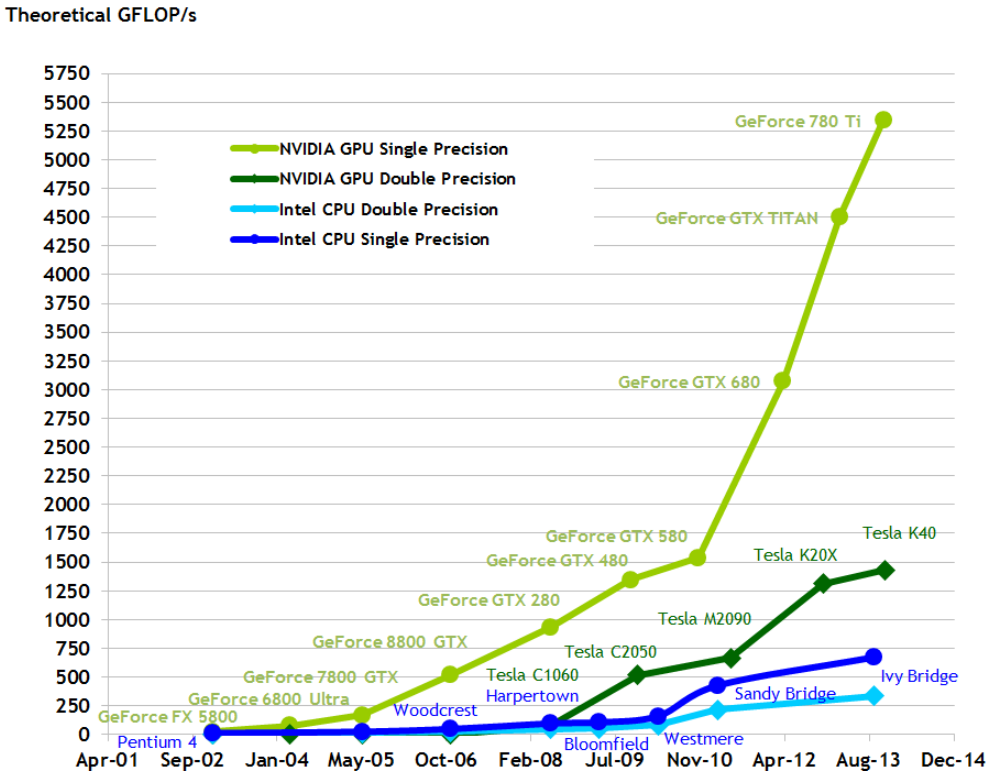
Jak se uvádí dále v dokumentu, je tento propastný rozdíl způsoben především tím, že GPU je vysoce specializované na zobrazování. Což je přesně problém, který je výpočetně velmí náročný a velmi dobře paralelizovatelný. Z toho důvodu je větší část grafického čipu vyhrazena výpočetním jednotkám než je tomu u CPU, kde podstatnou část čipu zabírá paměť pro uchovávání mezivýsledků (cache) a dále jednotky starající se o řízení toku výpočtu. To je vidět na obrázku 2.2

2.2 Model programování

Programování v CUDA C se vyznačuje tím, že na místo volání klasických funkcí známých z jazyka C, umožňuje programátorům definovat takzvaná jádra (angl. kernels), která při zavolání způsobí to, že se vykonají N krát paralelně a jejich výpočet běží v různých CUDA vláknech jak je uvedeno v manuálu [4].

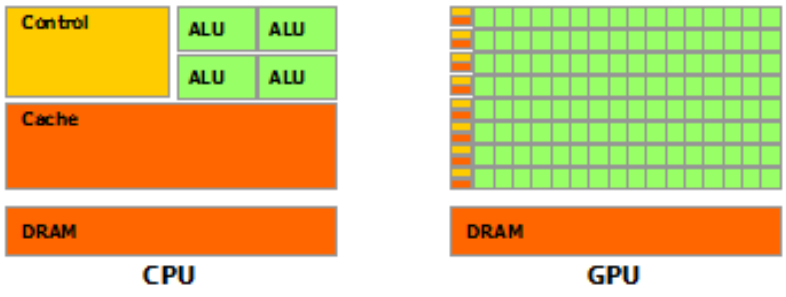
2.2.1 Jádra

Jak jsem již napsal v předchozím odstavci, jádro je prostředek, jak vykonávat kód paralelně. Definuje se pomocí deklarace `__global__` a počet vláken, která mají



Obrázek 2.1: Počet operací v plovoucí řádové čárce pro různá GPU a CPU

Zdroj: NVIDIA CUDA C Programming Guide[4]



Obrázek 2.2: Porovnání rozložení různých částí procesoru na CPU a GPU

Zdroj: NVIDIA CUDA C Programming Guide[4]

dané jádro vykonávat se určuje pomocí nové syntaxe `<<<...>>>` pro konfiguraci výpočtu. Každé vlákno, které vykonává dané jádro, je označeno unikátním ID vlákna, které je z daného jádra dostupné skrze proměnnou `threadIdx`. Všechny tyto informace o jádrech jsou uvedeny v návodu [4].

2.2.2 Hierarchie vláken

Jelikož vláken může být pro jedno jádro velmi mnoho, je v návodu [4] zmíněna hierarchie vláken, která je zobrazená na obrázku 2.3. Proměnná `threadIdx` je ve skutečnosti 3-složkový vektor, díky tomu mohou být vlákna vybírána pomocí jedno-, dvou- nebo tří-dimenzionálního indexu a tvořit tak jedno-, dvou- nebo tří-dimenzionální blok vláken. To umožňuje velice jednoduše provádět výpočty pro struktury jako vektor, matice, či objem.

Vztah mezi indexem vlákna a jeho ID je následující:

- pro jednodimenzionální index platí, že ID vlákna = thread index,
- pro dvoudimenzionální index platí, že ID vlákna pro index (x, y) je $(x+yD_x)$ při velikost bloku (D_x, D_y),
- pro třídimenzionální index platí, že ID vlákna pro index (x, y, z) je $(x + yD_x + zD_xD_y)$ při velikost bloku (D_x, D_y, D_z).

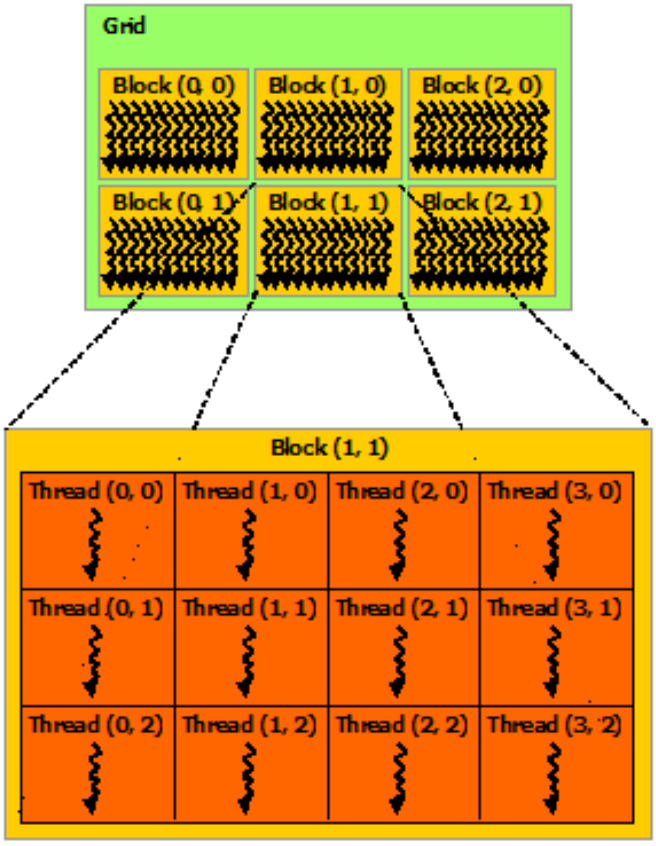
Jelikož se očekává, že všechna vlákna bloku poběží na stejném jádře procesoru a musí si tedy vystačit s omezenými paměťovými prostředky, je na aktuálních grafických počet vláken v bloku omezen na 1024.

2.2.3 Hierarchie pamětí

Dle návodu [4] mohou CUDA vlákna během svého běhu přistupovat k datům v různých adresních prostorech jak je vidět na 2.4. Každé vlákno má vlastní lokální pamět. Všechna vlákna v bloku mají přístup do sdílené paměti bloku, která má stejnou životnost jako blok samý. Úplně všechna vlákna mají přístup do globální paměti. Přičemž přístup do globální paměti je nejpomalejší, uprostřed je přístup do sdílené paměti a nejrychlejší je přístup do lokální paměti.

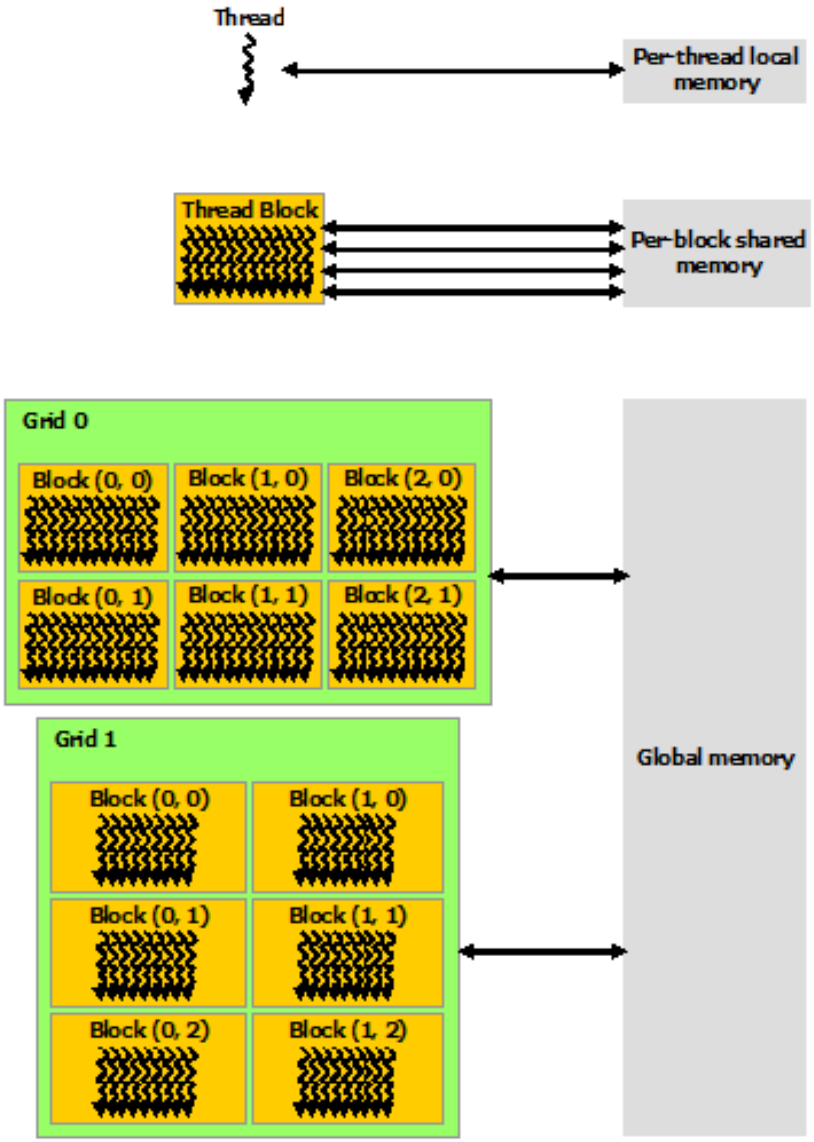
2.2.4 Driver vs Runtime API

Jak jsem již zmínil programování v CUDA C je možné provádět dvěma různými způsoby. Jedním z nich je běhové API, které bylo zmíněno již výše a z pohledu uživatele je jednodušší a tedy z mého podhledu i přístupnější. Druhým je tzv. API ovladače, které je na nižší úrovni než prvně zmíněné, což umožňuje mít celý paralelní výpočet pod větší kontrolou. S tím souvisí i větší rozsah možností při hlášení chyb. Ve frameworku, do kterého implementuji zadané algoritmy je použit právě tento přístup.



Obrázek 2.3: Rozdělení vláken do hierarchických struktur

Zdroj: NVIDIA CUDA C Programming Guide[4]



Obrázek 2.4: Rozdělení paměti do hierarchických struktur a přístupu k nim z vláken

Zdroj: NVIDIA CUDA C Programming Guide[4]

3. Implementace

3.1 Algoritmus

Oproti verzím algoritmů na CPU je v tomto případě upraven základní způsob práce. Místo toho, aby se postupně vykreslovaly jednotlivé body a ozáření se počítalo v případě že je třeba a rovnou se uložilo do mezipaměti, tak zde se pracuje v jednotlivých iteracích. Před spuštění algoritmu je obraz rozdělen na bloky o velikosti 16x16 obrazových bodů.

V každé iteraci se dle jejího čísla rozhodne, kterého konkrétního obrazového bodu z bloku o velikoti 16x16 se daná iterace bude týkat. Poté se z kamery vyšle skrz daný obrazový bod do scény paprsek. Na jeho prvním průsečíku se scénou se zjistí, jestli je v mezipaměti uloženo dost vzorků ozáření pro jeho interpolaci z těchto bodů do aktuálního průsečíku. Pokud ano, vypočte se hodnota ozáření v daném bodě a pomocí ní se spočítá výsledná zář pro daný paprsek. V opačném případě se daný bod uloží do zásobníku bodů pro výpočet ozáření vzorkováním hemisféry.

V druhé části iterace se dalším kernelem spočítá pro všechny body v zásobníků ozáření vzorkováním hemisféry. Souběžně s tím se spočítají rotační 1.4 a translační 1.5 gradienty a další veličiny potřebné pro interpolaci a vše se uloží do zásobníku záznamů ozáření.

V poslední části iterace se všechny nově vypočítané záznamy ozáření vloží do urychlovací struktury pro jejich budoucí vyhledávání.

3.1.1 Vzorkování bloků

Pro určení pořadí pixelů v jednotlivých blocích jsem použil algoritmus, který využívá Jones [5] pro vzorkování jednoho obrazového bodu. V původním použití algoritmus pracuje s čísly s plovoucí desetinnou čárkou, pro užití v této práci jsem ho upravil do celočíselné podoby. Druhou změnou oproti originálu bylo zvětšení velikosti použitého bloku z původních 8x8 vzorků na mnou používanou velikost 16x16. V tabulce 3.1 je zobrazeno pořadí prvních 64 vzorků z 256 které jsou potřeba pro pokrytí celé plochy bloku obrazových bodů.

3.1.2 Vzorkování polokoule

Vzorkování polokoule obstarává jeden CUDA kernel. Jak je v CUDA zvykem vlákna v tomto kernelu jsou rozdělena do bloků, kde každý blok odpovídá jedné vzorkované polokouli. Jelikož hodnoty M a N ze vzorce 1.2 mají dodržovat pravidlo $N \approx \pi M$ jsou pro rozměry bloku zvoleny hodnoty $M = 16$ a $N = 64$. Mocniny 2 jsou zvoleny z důvodu jednoduššího výpočtu finální hodnoty ze všech vláken. Hodnota získaná vzorkováním je uložen do stílené paměti a po ukončení výpočtu vzorků ve všech vláknech bloku je z těchto hodnot v jednom vlákně vytvořen výsledný záznam ozáření pro mezipaměť.

Tabulka 3.1: Tabulka zobrazující pořadí pixelů pro prvních 64 iterací

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1		61		13		50		2		62		14		49	
1									42							
2	41		21		37		26				22		38		25	
3																
4	9		53		5		58		10		54		6		57	
5																
6	35		31		47		16		32		28		44		19	
7																
8	3		63		15		48		0		60		12		51	
9																
10	43		23		39		24		40		20		36		27	
11																
12	11		55		7		56		8		52		4		59	
13																
14	33		29		45		18		34		30		46		17	
15																

4. Výsledky

V této kapitole se budu zabývat výsledky, tedy obrázky, které generují oba zadané algoritmy.

4.1 Etalon

Pro kontrolu toho, že implementace zadaných algoritmů funguje správně jsem si připravil etalony. Jde o různé scény, ve kterých bylo globální osvětlení spočítáno algoritmem sledování cesty. Který je nestranný a nevytváří žádnou systematickou chybu. Jediná jeho vada je, že běží celkem dlouho. První scénou je klasický CornellBox, který obsahuje pouze difúzní povrchy. Pro tvorbu jeho etalonu bylo každým obrazovým bodem simulováno 100 000 světelných cest.

Závěr

V předkládané práci jsem představil algoritmy iradiance a radiance caching, ukázal jsem jaké jsou jejich hlavní výhody a proč je obtížné je implementovat na grafické kartě. K tomu jsem využil framework CUDA, který jsem popsal v druhé kapitole a zmínil, proč je pro tento úkol vhodný.

Při implementaci jsem si ověřil, že ladění programů běžících na grafické kartě je obtížnější, než je tomu u programů běžících na CPU, ač framework CUDA k tomu nabízí dostatek nástrojů. V tomto měl nezanedbatelný vliv i použitý operační systém, který sice umožnil použití velmi kvalitního vývojového software, ale to bylo převáženo jeho neočekávaným chováním při výpočtech.

Seznam použité literatury

1 WIKIPEDIE. *Radiometrie — Wikipedie: Otevřená encyklopedie*. 2014. [Online; navštíveno 4. 05. 2015]. Dostupný také z WWW: ⟨`http://cs.wikipedia.org/w/index.php?title=Radiometrie&oldid=11666098`⟩.

2 KŘIVÁNEK, Jaroslav; GAUTRON, Pascal; WARD, Greg et al. Practical global illumination with irradiance caching. In. *ACM SIGGRAPH 2008 classes*. Los Angeles, California: ACM, 2008, s. 60:1–60:20. SIGGRAPH '08. Dostupný také z WWW: ⟨`http://doi.acm.org/10.1145/1401132.1401213`⟩.

3 CORPORATION, NVIDIA. *NVIDIA CUDA Getting Started Guide for Microsoft Windows*. 2015. Dostupný také z WWW: ⟨`http://docs.nvidia.com/cuda/cuda-getting-started-guide-for-microsoft-windows/index.html`⟩.

4 CORPORATION, NVIDIA. *NVIDIA CUDA C Programming Guide*. 2013. Dostupný také z WWW: ⟨`http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html`⟩.

5 JONES, Nathaniel. Parallel Irradiance Caching on the GPU. 2013.

Seznam tabulek

Seznam obrázků

Seznam použitých zkratek

CPU Central Processing Unit

CUDA Compute Unified Device Architecture

GFLOP Giga FLoating point OPeration

GPU Graphics Processing Unit

PDF Probability distribution function

HW Hardware

API Application Programming Interface

Přílohy

1 Obsah DVD

Součástí této práce je i přiložené DVD, které obsahuje text práce, zdrojové soubory včetně příkladů. Adresářová struktura přiloženého DVD:

- `dokumenty` – obsahuje text práce ve formátu PDF.
- `zdrojove_soubory` – obsahuje zdrojové soubory programu a projekt pro Microsoft Visual Studio.