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

RAYTRACING NA GPU

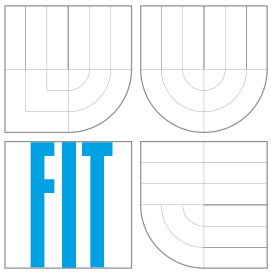
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MAREK STRAŇÁK

BRNO 2011

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

RAYTRACING NA GPU

RAYTRACING ON GPU

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MAREK STRAŇÁK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. LUKÁŠ POLOK

BRNO 2011

Zadání diplomové práce/11989/2010/xstran00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2010/2011

Zadání diplomové práce

Řešitel: **Straňák Marek, Bc.**
Obor: Počítačová grafika a multimédia
Téma: **Raytracing na GPU**
Raytracing on GPU
Kategorie: Počítačová grafika

Pokyny:

1. Prostudujte dostupné API pro programování GPU (OpenGL, CUDA, OpenCL).
2. Prostudujte a popište metody osvětlování scény pomocí sledování paprsku.
3. Navrhněte jednoduchý systém výpočtu sledování paprsku.
4. Navržený systém implementujte, vytvořte jednoduchou demonstrační aplikaci.
5. Zhodnoťte dosažené výsledky, diskutujte možnosti dalšího vývoje.
6. Vytvořte stručný plakátek, reprezentující výsledky Vaší práce.

Literatura:

- Dle zadání vedoucího.

Při obhajobě semestrální části diplomového projektu je požadováno:

- Body 1 - 3.

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

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

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

Vedoucí: **Polok Lukáš, Ing.**, UPGM FIT VUT
Datum zadání: 20. září 2010
Datum odevzdání: 25. května 2011

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

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Raytracing je základnou technikou pro vizualizaci trojrozměrných objektů. Cílem práce je demonstrovat možnost implementace sledovaní paprsků pomocí grafického akcelerátoru. Popíšem základní algoritmus a jeho modifikovanou verzi, která byla implementována pomocí jazyka CUDA C. Výsledný raytracer je optimalizovaný pro dynamické scény. Pro tento účel byla použita akcelerační struktura KD strom, hierarchické obalové tělesa a přenos dat pomocí PBO. Pro realističtější výstupy byla také implementována fotonová mapa zobrazující kaustiky.

Abstract

Raytracing is a basic technique for displaying 3D objects. The goal of this thesis is to demonstrate the possibility of implementing raytracer using a programmable GPU. The algorithm and its modified version, implemented using „C for CUDA“ language, are described. The raytracer is focused on displaying dynamic scenes. For this purpose the KD tree structure, bounding volume hierarchies and PBO transfer are used. To achieve realistic output, photon mapping was implemented.

Klíčová slova

Raytracer, metoda sledování paprsků, NVIDIA CUDA, GPGPU, osvětlovací model, stínění, průsečík paprsku s trojúhelníkem, KD strom, fotonové mapy.

Keywords

Raytracer, raytracing, NVIDIA CUDA, GPGPU, lighting model, shading model, ray-triangle intersection, KD tree, photon mapping.

Citace

Marek Straňák: Raytracing na GPU, diplomová práce, Brno, FIT VUT v Brně, 2011

Raytracing na GPU

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Lukáša Poloka.

.....
Marek Straňák
18. května 2011

Poděkování

Na tomto mieste by som rád poďakoval všetkým, ktorí ma podporovali pri písaní tejto diplomovej práce. Moje poďakovanie patrí predovšetkým Ing. Lukášovi Polokovi za jeho odbornú pomoc, trpezlivosť a cenné konzultácie. Taktiež ďakujem svojim rodičom a kamarátom, ktorí mi boli oporou počas celého štúdia.

Obsah

Zoznam obrázkov

Kapitola 1

Úvod

Už od vzniku počítačov bola snaha zobraziť na monitore prvé informácie. Najskôr to boli len rôzne znaky reprezentujúce písmená, neskôr však bolo možné zobrazovať obrázky a dokonca prvé trojrozmerné scény. Začali vznikať samostatné jednotky pre výpočet a zobrazenie vizuálnych dát nazývané GPU (*Graphics Processing Unit*). Odvtedy počítačová grafika zaznamenala obrovský rozmach a svoje uplatnenie našla v rôznych oblastiach priemyslu, vedy, medicíny ale aj herného odvetvia.

Prudký nárast počítačových hier znamenal neustále zlepšovanie kvality zobrazenia a podmieňoval vývoj grafických akcelerátorov. Súčasná generácia grafických kariet predstavuje obrovský výpočtový výkon, čo sa prejavilo vo využití GPU nie len na spracovanie obrazových dát. Začali vznikať prvé jazyky pre programovanie aplikácií implementovaných na grafických kartách. Spočiatku išlo len o vytváranie shaderov, neskôr však vznikali vysokoúrovňové jazyky s veľkou podporou rôznych nástrojov a knižníc pre jednoduchú realizáciu rozličných výpočtov.

Zdokonaľovanie výpočtovej techniky umožnilo implementovať aj zložitejšie postupy vizualizácie obrazu. Mnohé metódy realistického zobrazenia, ktoré sú výpočtovo a časovo náročné, môžu byť niekoľkonásobne urýchlené a spracované v reálnom čase. Patria sem rôzne pokročilé techniky sledovania lúča ale aj metódy pre výpočet globálneho osvetlenia.

1.1 Cieľ práce

Cieľom práce bolo zmapovať dostupné technológie pre implementáciu obecných výpočtov na grafických kartách. Jednotlivým technológiám je venovaná kapitola 2. Pozornosť je zameraná predovšetkým na architektúru CUDA a jej základné vlastnosti. Je podrobne definované jej rozhranie, hardvérové členenie a spôsob výpočtu pre dosiahnutie čo najväčšieho výkonu.

V ďalších častiach práce 3 a 4 sa nachádzajú rôzne algoritmy pre výpočet osvetlenia a zobrazenia scény. Sú tu uvedené jednoduché ale aj zložité postupy, ktoré majú za cieľ dosiahnuť realistické výsledky. Hlavná pozornosť je upriamená na metódu sledovania lúča, ktorá bola napokon implementovaná pomocou jazyka CUDA C.

Kapitola 5 sa zaoberá rôznymi možnosťami urýchlenia samotnej implementácie sledovania lúča. Ide predovšetkým o akceleračné štruktúry ako sú napr. uniformné mriežky, binárne delenie priestoru alebo KD stromy. Postupne je popísaný princíp stavby a prechádzania týchto štruktúr a taktiež sú uvedené ich výhody a nevýhody .

Šiesta kapitola 6 je venovaná samotnému návrhu algoritmu sledovania lúča. Sú tu definované postupy a riešenia problémov, ktoré boli použité pri implementácii.

Záverečná kapitola 7 obsahuje rôzne testy nad vytvorenou aplikáciou. Merania boli realizované pre rôzne scény a rozdielne natavené parametre, ako je rozlíšenie obrazu, hĺbka zanorenia, použite PBO a mnohé iné. Dosiahnuté výsledky sú okomentované a zdôvodnené.

V samotnom závere 8 sú zhrnuté prednosti a nedostatky implementácie na grafickom akcelerátore. Hodnotím tu dosiahnuté úspechy a diskutujem o ďalších možnostiach vývoja a rozšíreniach do budúcnosti.

Kapitola 2

Programovanie GPU

V posledných rokoch prešli grafické karty búrlivým vývojom a zaznamenali výrazné zmeny v oblasti počítačovej grafiky. S rastúcim výkonom grafických procesorov sa začalo uvažovať o využití grafického akcelerátora aj pre iné účely ako spracovanie trojrozmerných obrazových dát. GPU sú vďaka svojej štruktúre vhodné predovšetkým pre vysoko paralelné výpočty, čím umožňujú v porovnaní s CPU efektívnejšie spracovanie niektorých algoritmov.

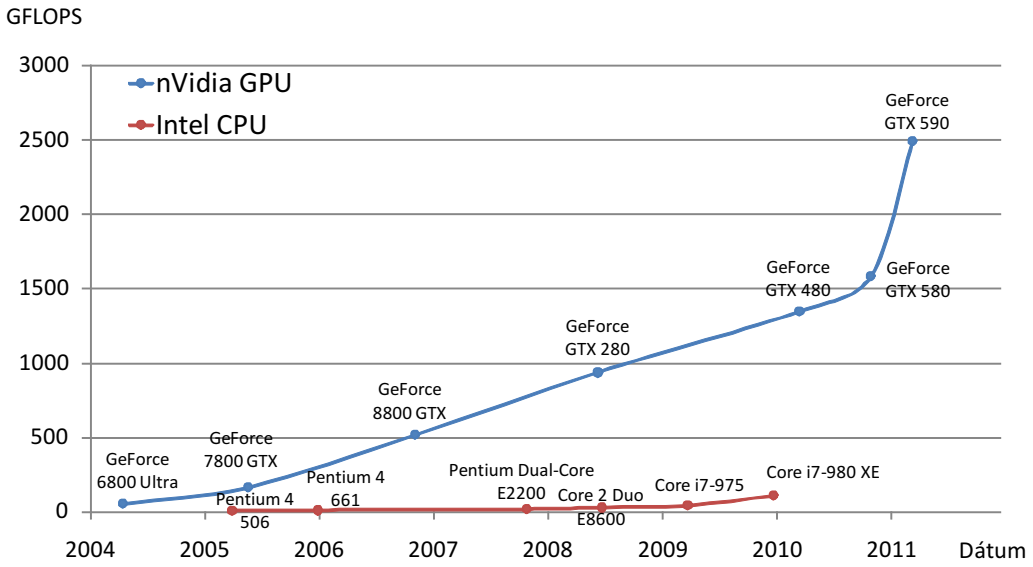
Hlavným cieľom prenosu výpočtu z CPU na GPU je zvýšenie výpočtového výkonu najrôznejších úloh, ako sú napr. náročné operácie nad veľkým množstvom dát, rôzne simulácie alebo dokonca vykonávanie náročných dotazov nad databázami. Ide predovšetkým o algoritmy z rôznych oblastí matematiky, fyziky, medicíny, ale aj počítačovej grafiky a iných oblastí. Medzi vhodné implementácie na GPU patria Fourierová transformácia, numerické riešenia diferenciálnych rovníc, výpočet zobrazovacej rovnice a mnohé iné.

GPU obsahuje množstvo výpočtových jednotiek (jadier). Je optimalizované pre aritmetické operácie nad číslami s pohyblivou rádovou čiarkou a pre prácu s rovnakým typom dát. V takýchto výpočtoch je mnohonásobne výkonnejšie ako klasické CPU (pozri obrázok 2.1). Taktiež dokáže pracovať so svojou pamäťou DRAM niekoľkonásobne vyššou rýchlosťou. Problémy má s bežnými úlohami ako je beh operačného systému, spúšťanie programov, obsluha prerušenia a pod. Cieľom GPU nie je nahradenie klasického procesora, ale jeho využitie ako určitého koprocesora (akcelerátora). Rozdielna štruktúra GPU a CPU je znázornená na obrázku 2.2.

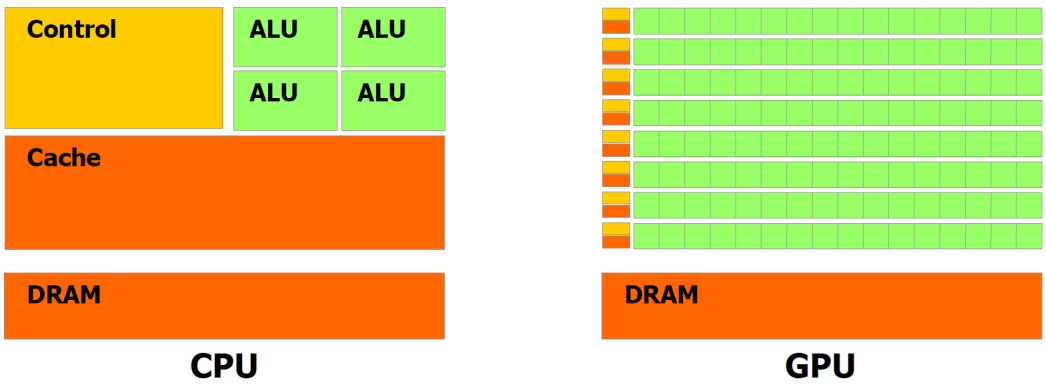
Medzi prvé jazyky umožňujúce programovanie grafických procesorov patrili tzv. shading jazyky. Sú určené predovšetkým k programovaniu shaderov ovplyvňujúce grafickú pipeline a ich využitie pre obecné výpočty je obmedzené. Vyžadujú pokročilé znalosti grafického API. Medzi hlavné jazyky pre programovanie shaderov patria GLSL (*OpenGL Shading Language*), HLSL(*High Level Shading Language*), Cg (*C for Graphics*), atď.

Pre obecné výpočty je výhodnejšie používať nástroje špeciálne na to určené. Takéto nástroje sa označujú skratkou GPGPU (*General Purpose Computation on Graphics Processing Unit*). V súčasnosti existuje celá rada technológií určených pre akceleráciu obecných výpočtov prostredníctvom grafických kariet. K najznámejším a najrozšírenejším technológiám v dnešnej dobe patria CUDA, ATI Stream, OpenCL, alebo DirectCompute.

V nasledujúcom texte popíšem niektoré z týchto nástrojov, pričom väčšiu pozornosť budem venovať architektúre CUDA, ktorú som použil vo svojej implementácii. V dobe písania tejto práce bola jej technická vyspelosť a jednoduchosť výrazne vyššia ako u ostatných technológií. Taktiež disponuje veľkým množstvom manuálov a rôznych príkladov.



Obrázok 2.1: Porovnanie rozdielu výkonu GPU a CPU pri práci s číslami v pohyblivej rádovej čiarke



Obrázok 2.2: Rozdielna štruktúra GPU a CPU (prevzaté z [2])

2.1 ATI Stream

Technológia bola pôvodne určená pre profesionálne účely na drahých grafických kartách, avšak po rozšírení CUDA bolo ATI Stream povolené prostredníctvom ovládačov aj pre bežných používateľov. Podobne, ako je CUDA späté s hardvérom od firmy nVidia, ATI Stream má podporu v grafických kartách od spoločnosti AMD (pôvodne ATI).

Technológia v rôznych obdobiach svojho vývoja reprezentovalo rozličné prístupy pre programovanie grafických kariet. Za bezprostredného predchodcu sa považuje *Data Parallel Virtual Machine*, ktorý bol neskôr označovaný ako CTM (*Close to Metal*). Jeho základný prínos spočíval v sprístupnení funkcionality stream procesorov bez použitia jazyka pre tvorbu shaderov. Nejednalo sa však o žiadny vysokoúrovňový jazyk a syntax CTM bola podobná assembleru. V súčasnosti je ATI Stream postavený na technológii BrookGPU, ktorá bola pôvodne vyvinutá na Standfordskej univerzite v roku 2004. Ide o vysokoúrovňový kód podobný jazyku C, ktorý môžeme chápať ako abstraktnú vrstvu nad klasickým programovaním shaderov.

Ati Stream nie je len jazyk pre programovanie GPU, ale predstavuje technológiu poskytujúcu mnoho nástrojov pre samotný vývoj. Jedným takýmto nástrojom je *ATI Strem Profiler*, ktorý umožňuje programátorovi analyzovať jeho kód a taktiež poskytuje prehľadné užívateľské rozhranie. Novšie grafické karty od AMD sa vyznačujú podporou technológie OpenCL.

2.2 OpenCL

OpenCL (*Open Computing Language*) je otvorený štandard pre programovanie paralelných systémov. Jeho počiatky siahajú k firme Apple, ktorá má na svedomí jeho prvotný návrh. V súčasnosti stojí za jeho vývojom konzorcium Khronos Group, ktoré je tvorené zástupcami najväčších firiem v oblasti IT (AMD, IBM, nVidia, Intel, Motorola a mnohé iné).

Základnou výhodou OpenCL je nezávislosť na operačnom systéme a konkrétnom grafickom zariadení (ako je to u CUDA alebo ATI Stream), čo mu umožňuje vykonávanie paralelného výpočtu nad množstvom rôznorodých výpočtových jednotiek. Medzi podporované typy hardvéru patria viacjadrové procesory s podporou inštrukcií SSE3, grafické procesory (nVidia, AMD), procesory typu Cell ale aj niektoré mobilné čipy. Hlavná myšlienka OpenCL spočíva vo využití niekoľkých výpočtových jednotiek rôznych typov súčasne. Veľká podpora rozličných zariadení však prináša zložitejšie API a problémy s prenositeľnosťou medzi platformami rôznych výrobcov.

2.2.1 Rozhranie

Štandard OpenCL definuje programovací jazyk, knižnice, API a runtime systém. Jazyk OpenCL vychádza z programovacieho jazyka C (konkrétne zo štandardu C99), obsahuje ale určité rozšírenia a obmedzenia. Oproti jazyku C nepodporuje rekurziu, hlavičkové súbory, ukazovatele na funkcie, atď. Na druhej strane obsahuje podporu vektorových typov, synchronizáciu, sadu rôznych vstavaných funkcií (napr. goniometrické funkcie) a mnohé iné.

2.2.2 Architektúra zariadenia

Hardvérový model je tvorený hostiteľským zariadením (zvyčajne procesor CPU) a jedným alebo viacerými OpenCL zariadeniami (*compute device*). Tie sú zložené z niekoľkých výpočtových jednotiek (*compute unit*), ktoré odpovedajú multiprocesorom v CUDA architektúre,

alebo iným výpočtovým jednotkám ako sú napr. jadrá CPU, FPGA, atď. Každá výpočtová jednotka pritom pozostáva z jedného alebo viacerých výkonných prvkov (*processing element*). Výkonné prvky si môžeme predstaviť ako samostatné jednotky (jadrá) realizujúce ucelenú časť výpočtu. V porovnaní s CUDA architektúrou sú zhodné s jednotlivými stream procesormi. Samotná aplikácia beží na hostiteľskom zariadení a predáva príkazy k spracovaniu jednotlivým výpočtovým jednotkám vo vnútri OpenCl zariadenia.

2.2.3 Pamäťový modul

OpenCL definuje štyri základné typy pamätí s rôznymi vlastnosťami a prístupovou dobou:

- **globálna pamäť** (*global memory*) – najpomalšia zo všetkých pamätí, je prístupná v rámci jedného OpenCL zariadenia, prístup k nej majú všetky výkonné prvky vo výpočtových jednotkách,
- **pamäť konštánt** (*constant memory*) – označuje časť globálnej pamäte, ktorá je počas výpočtu nemenná (určená len pre čítanie), slúži k ukladaniu konštánt a zvyčajne obsahuje vyrovnávaciu pamäť,
- **lokálna pamäť** (*local memory*) – je zdieľaná medzi všetkými výkonnými prvkami v rámci jednej výpočtovej jednotky, väčšinou býva rýchlejšia ako globálna pamäť,
- **súkromná pamäť** (*private memory*) – pamäťový priestor vyhradený pre jednotlivé výkonné prvky, ide o najrýchlejšiu pamäť predstavujúcu registre.

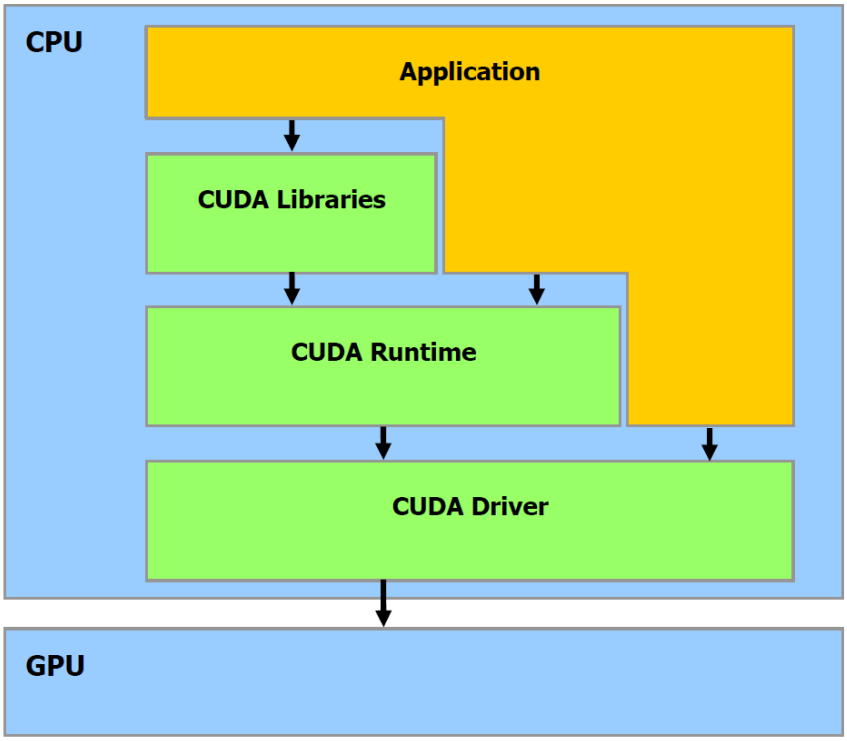
2.3 CUDA

CUDA (*Compute Unified Device Architecture*) je technológia určené pre výpočty na grafických kartách, ktorá bola vyvinutá spoločnosťou nVidia. Ide o súbor rôznych hardvérových a softvérových prostriedkov potrebných pre programovanie grafického akcelerátora. Jej história nesiaha príliš ďaleko do minulosti. Prvá verzia bola uverejnená v decembri roku 2006 a odvtedy sa neustále vyvíja.

V súčasnosti sú podporované všetky najpoužívanejšie 32 a 64-bitové operačné systémy, ako sú Windows, Linux a Mac OS X. Za nevýhodu tejto technológie možno považovať závislosť na hardvéri od firmy nVidia. Programy postavené na platforme CUDA tak nie je možné spracovávať na grafických kartách od iných výrobcov. Na druhej strane však umožňuje spoluprácu s grafickými API OpenGL a DirectX. Taktiež podporuje výpočty s jednoduchou a dvojitou presnosťou v pohyblivej rádovej čiarke. Pre tvorbu aplikácie pomocou tejto architektúry je možné využiť nízkoúrovňové a aj vysokoúrovňové programovanie, čo je spracované v ďalších častiach tejto kapitoly.

2.3.1 Rozhranie

Ako už bolo spomínané, pre implementáciu programov na GPU je možné využiť dva typy rozhrania, nízkoúrovňové (*CUDA Driver API*) alebo vysokoúrovňové (*CUDA Runtime API*). Vysokoúrovňové rozhranie je jednoduchšie a je postavené na nízkoúrovňovom. Výhoda nízkoúrovňového programovania spočíva v priamom prístupe na CUDA zariadenie, čo poskytuje viacero možností, implementácia je ale oveľa zložitejšia. Na obrázku 2.3 je vyobrazená základná štruktúra rozhrania.



Obrázok 2.3: Rozhranie CUDA (prevzaté z [2])

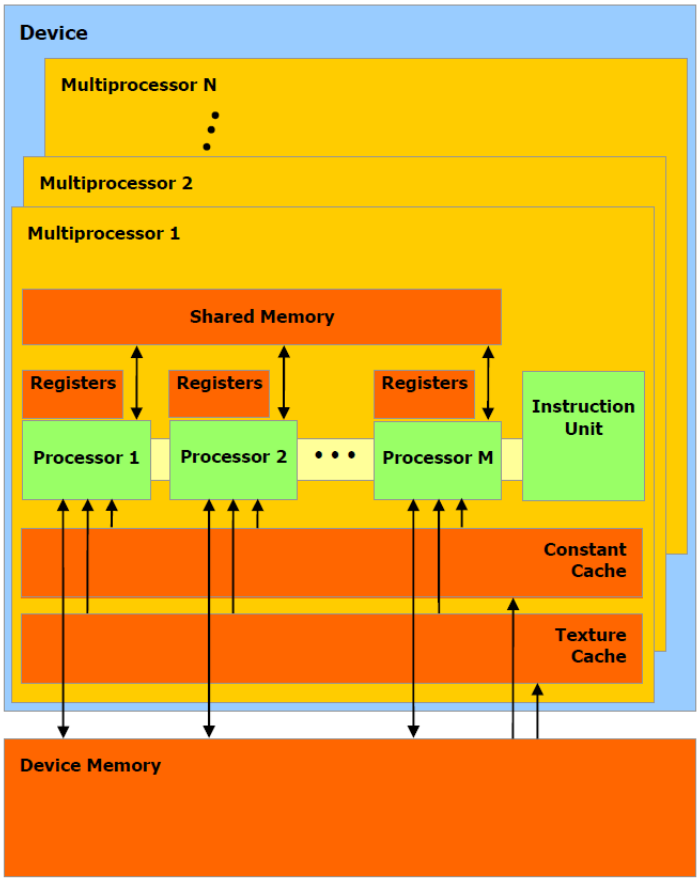
Architektúra CUDA podporuje veľké množstvo jazykov, medzi ktoré patria OpenCL alebo DirectCompute. Pre priame programovanie GPU je taktiež možné použiť aj samotné CUDA Driver API využívajúce kód PTX. Vďaka Runtime API je možné použitie jazyka CUDA C, ktorý vychádza z jazyka C, obsahuje však rôzne rozšírenia. Pre implementáciu môžu byť využité aj ďalšie vysokoúrovňové jazyky ako sú C++, Fortran, Java, Python, atď.

Pre prácu s grafickým akcelerátorom poskytuje nVidia celý balík nástrojov *Cuda Toolkit*, ktorý obsahuje kompilátor (nvcc), profiler (cudaprof), debugger (cudagdb), knižnice (CUBLAS, CUFFT), ale aj rôzne dokumentácie a zdrojové kódy.

CUDA taktiež umožňuje spustenie programu v tzv. emulovanom móde, čo dovoľuje vykonanie programov aj na počítačoch, ktoré nie sú vybavené potrebným hardvérom (grafickou kartou od spoločnosti nVidia). Celý výpočet je realizovaný na CPU, čo má za následok extrémny nárast výpočtového výkonu. Pre spustenie aplikácie v emulovanom móde je potrebná jej rekompilácia s príslušným parametrom.

2.3.2 Architektúra

Z pohľadu GPU je celé zariadenia rozdelené na niekoľko multiprocesorov (*multiprocessor*) a globálnu pamäť (*device memory*). Počet multiprocesorov je rôzny v závislosti na type grafickej karty. Každý obsahuje niekoľko výpočtových jadier (*stream processor*), inštrukčnú jednotku (*instruction unit*), sadu registrov (*registers*), zdieľanú pamäť (*shared memory*) a vyrovnávaciu pamäť konštantnej (*constant cache*) a textúrovej pamäte (*texture cache*). Štruktúra celého zariadenia je znázornené na obrázku 2.4.



Obrázok 2.4: Architektúru CUDA z pohľadu GPU (prevzaté z [2])

Jednotlivé generácie grafických kariet sa medzi sebou navzájom odlišujú výkonom, veľkosťou pamäte, počtom multiprocesorov, atď. Každá GPU je definovaná pomocou tzv. *compute capability*, pričom novšia verzia je vždy spätne kompatibilná so staršími verziami a prináša nové funkcie a rozšírenia. Aby bolo možné pre programovanie grafického akcelerátora použiť architektúru CUDA, musí grafická karta podporovať aspoň *compute capability 1.0*. Ako príklad uvediem kartu nVidia GeForce GTX 590, ktorá má podporu *compute capability 2.0* a obsahuje 1024 stream procesorov.

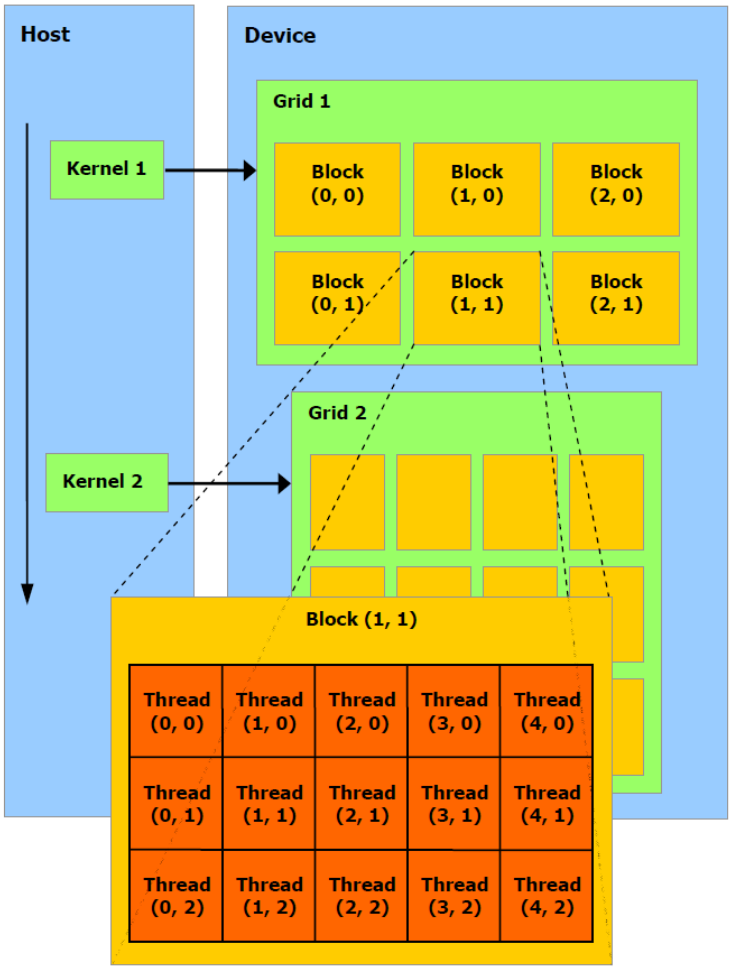
2.3.3 Hierarchia vláken

Pre správnu implementáciu paralelného programu je potrebné vedieť, ako prebieha samotný výpočet na grafickej karte. Dôležité je predovšetkým usporiadanie vláken a ich priradenie jednotlivým procesorom a multiprocesorom.

Vlákno (*thread*) reprezentuje najmenšiu výpočtovú jednotku (jeden proces) na GPU. Pre využitie výkonu grafickej karty je potrebných niekoľko tisíc týchto vláken. Jednotlivé vlákna sú zoskupené do blokov (*block*), ktoré sú navzájom výpočtovo nezávislé. Každý obsahuje zdieľanú pamäť (*shared memory*), pomocou ktorej môžu vlákna v rámci bloku medzi sebou komunikovať. Vnútorná logická štruktúra bloku môže byť jednorozmerná, dvojrozmerná alebo trojrozmerná a mala by odpovedať formátu vstupných dát. Jednotlivé bloky

sú pri spracovaní výpočtu rozdistribuované medzi všetky dostupné multiprocesory, a preto by mal ich počet byť výrazne väčší.

Bloky sú združené do väčšieho celku, tzv. mriežky (*grid*). Mriežkou nazývame všetky vlákna, ktoré vzniknú spustením *kernelu*. Štruktúra mriežky môže byť taktiež jednorozmerná, dvojrozmerná alebo trojrozmerná. Hierarchia vláken, blokov a mriežok je vyobrazená na obrázku 2.5.



Obrázok 2.5: Hierarchia vláken, blokov a mriežok (prevzaté z [2])

Výpočet na GPU sa realizuje pomocou *warpov*, čo je skupina 32-och vláken. Počet aktívnych *warpov* na jednom multiprocesore závisí na množstve zdieľanej pamäte a počte registrov, ktoré *warp* potrebuje pre svoj beh. Procesory sú architektúry typu SIMT (*Single Instruction Multiple Thread*), ktorá dovoľuje nezávislé spracovanie vláken. Každé vlákno má pridelený vlastný čítač inštrukcií, čo umožňuje použitie konštrukcií vetvenia a cyklov. Vetvenie vláken v rámci *warpu* môže mať však veľký vplyv na výpočtový výkon, pretože jednotlivé vetvy sú spracované postupne (sériovo). Najlepšie využitie multiprocesoru nastáva v prípade, ak všetky vlákna spracovávajú rovnakú postupnosť inštrukcií.

Pre vzájomnú spoluprácu vláken je potrebná synchronizácia. V rámci bloku je možné synchronizovať jednotlivé vlákna pomocou príkazu `syncthreads()`. Ten zapríčiní, že vlákna budú na danom mieste pozastavené, až pokiaľ všetky nedosiahnu tohto miesta, z ktorého bola funkcia `syncthreads()` volaná. Synchronizácia je veľmi rýchla, len ak vlákna spracovávajú rovnakú postupnosť inštrukcií a nedochádza k vetveniu kódu. V prípade podmieneného vetvenia je synchronizácia prípustná, pokiaľ všetky vlákna prechádzajú rovnakou cestou, v opačnom prípade by došlo k uviaznutiu.

2.3.4 Kernel

Základom každej CUDA aplikácie je tzv. *kernel*. Ide o funkciu definovanú kľúčovým slovom `__global__`, ktorá je implementovaná na grafickej karte a pri spustení vygeneruje veľké množstvo vláken (procesov). *Kernel* sa volá pomocou príkazu ***názov_funkcie***`<<<` ***počet_blokov, počet_vlákien***`>>>(`***parametre***`)`. Ostrým zátvorkám `<<< >>>` hovoríme *execution configuration*. Prvá hodnota definuje veľkosť mriežky (počet blokov v mriežke) a druhá hodnota definuje veľkosť bloku (počet vlákien v jednotlivých blokoch). Obidve premenné sú typu `dim3` (3-zložkový vektor). Zariadenia s podporou *compute capability 2.0* alebo vyššou verziou dovoľujú spustenie viacerých *kernelov* súčasne, čo umožňuje lepšie využitie výpočtovej kapacity grafickej karty.

Každé spustené vlákno nesie informácie o veľkosti bloku (premenná `blockDim`), umiestnení bloku v mriežke (`blockIdx`) a umiestnení vlákna v rámci bloku (`threadIdx`). Vďaka týmto hodnotám je možné pre každé vlákno zistiť jeho unikátny index v rámci daného *kernelu*. Pre osu x môžeme definovať výslednú pozíciu vlákna ako `blockIdx.x * blockDim.x + threadIdx.x`. Premenná `threadIdx.x` nadobúda hodnoty od 0 až `blockDim.x` (počet vláken v bloku pre osu x), `blockIdx.x` nadobúda hodnoty 0 až `gridDim.x` (počet blokov v mriežke pre osu x). Podobne je možné vypočítať indexy vláken aj pre ďalšie osy y a z.

Okrem základnej funkcie `__global__` definuje architektúra CUDA ďalšie dva typy funkcií reprezentované kľúčovými slovami `__device__` (funkcie realizované a volané na GPU) a `__host__` (funkcie vykonávané a volané na CPU). Pokiaľ nemá funkcia žiadny identifikátor, automaticky je považovaná za *host* funkciu.

V prílohe A je popísaný jednoduchý algoritmus sčítania matíc implementovaný pomocou jazyka CUDA C.

2.3.5 Hierarchia pamätí

Grafický akcelerátor obsahuje niekoľko typov pamätí, ktoré sa od seba odlišujú rôznymi parametrami (čítanie/zápis, rýchlosť, veľkosť, kešovanie, atď.). Architektúra CUDA rozlišuje celkom šesť typov pamätí:

- globálna pamäť (*global memory*),
- pamäť konštánt (*constant memory*),
- textúrová pamäť (*texture memory*),
- zdieľaná pamäť (*shared memory*),
- lokálna pamäť (*local memory*),
- registre (*registers*).

Ich hierarchia je znázornená na obrázku 2.6. V tabuľke 2.1 sa nachádza stručná charakteristika a porovnanie týchto pamätí.

Globálna pamäť

Ide o hlavnú pamäť grafickej karty, ktorá má podobnú funkciu ako hlavná pamäť počítača. Je najväčším dátovým úložiskom v rámci GPU. Veľkosť môže dosahovať stovky MB až niekoľko GB. Slúži ako komunikačné médium pre prenos dát medzi *host* (CPU) a *device* (GPU). Jednotlivé jadrá do nej môžu zapisovať, ale aj čítať. Prístup do pamäte nie je kešovaný a trvá niekoľko stoviek cyklov, preto je aj zo všetkých pamätí na grafickej karte najpomalšia. Urýchlenie prístupu do pamäte je možné pomocou metódy združeného prístupu.

Pamäť konštánt

Používa sa pre dáta, ktorých hodnoty sú známe pred samotným spustením jadra. Na rozdiel od globálnej pamäte je kešovaná, čo umožňuje rýchlejší prístup k jednotlivým hodnotám. Čítanie z vyrovnávacej pamäte je rovnako rýchle ako čítanie z registrov, ak všetky vlákna *half-warpu* (16 vláken) čítajú z rovnakej adresy. Pamäť je schopná obslúžiť jednu požiadavku súčasne, a preto cena prístupu rastie lineárne s množstvom rôznych požadovaných adries. Ak hodnota nie je vo vyrovnávacej pamäti, je potrebný prístup do globálnej pamäte DRAM.

Textúrová pamäť

Ide o globálnu pamäť, ktorá je kešovaná a optimalizovaná na prístup k 2D poliam (tzn. zrýchľuje prístup k textúram uložených v hlavnej pamäti).

Zdieľaná pamäť

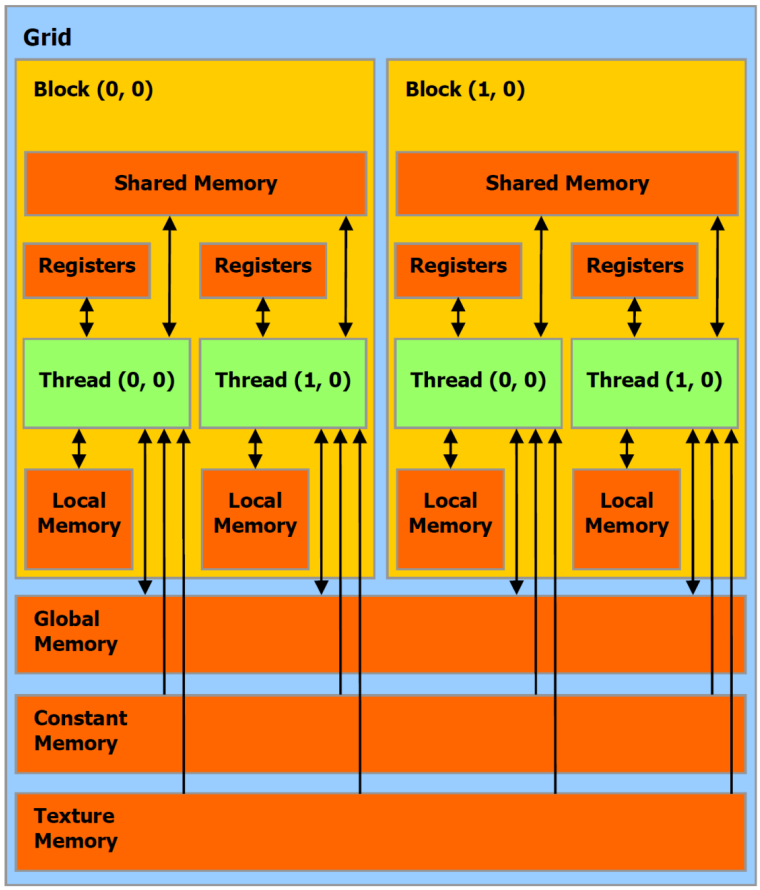
Je umiestnená na čipe a prístupná všetkým vláknam v rámci jedného bloku, čo umožňuje komunikáciu medzi vláknami. Je rozdelená na 16 nezávislých bánk. Prístup do nej je rovnako rýchly ako do registrov za predpokladu, že každé vlákno pristupuje do inej banky.

Lokálna pamäť

Každé vlákno má svoju lokálnu pamäť. Nenachádza sa priamo na čipe, preto je prístup rovnako pomalý ako do globálnej pamäte. Kompilátor do nej automaticky ukladá premenné, ktoré nemôžu byť uložené v registroch (vlákno vyčerpá všetky registre), čo má negatívny vplyv na rýchlosť aplikácie. Umiestnenie premenných do tejto pamäte nemôže programátor nijako ovplyvniť.

Registre

Registre sú podobne ako u CPU pracovným priestorom procesora. Predstavujú primárne úložisko premenných *kernelu* (kompilátor do nich ukladá premenné). Sú umiestnené priamo na čipe. Sú najrýchlejším typom pamäte, ktorú využívajú jednotlivé vlákna. Počet dostupných registrov je určený výpočtovou schopnosťou procesora (*compute capability*).



Obrázok 2.6: Hierarchia pamätí z pohľadu architektúry CUDA (prevzaté z [2])

Typ pamäte	**Prístup**	**Umiestnenie**	**Operácie**	**Kešovanie**
globálna pamäť	všetky vlákna a host	mimo čip (DRAM)	čítanie/zápis	áno*
pamäť konštánt	všetky vlákna a host	mimo čip (DRAM)	čítanie	áno
textúrová pamäť	všetky vlákna a host	mimo čip (DRAM)	čítanie	áno
zdieľaná pamäť	všetky vlákna v rámci jedného bloku	na čipe	čítanie/zápis	nie je potreba (na čipe)
lokálna pamäť	jedno vlákno	mimo čip (DRAM)	čítanie/zápis	áno*
registre	jedno vlákno	na čipe	čítanie/zápis	nie je potreba (na čipe)

* – platí pre grafické karty s podporou *compute capability 2.0* a vyššou verziou

Tabuľka 2.1: Rozdelenie GPU pamätí

Združený prístup do pamäte

Globálna pamäť nie je kešovaná, a preto je práca s ňou pomalá a neefektívna. Exituje však technika, pomocou ktorej je možné výrazne urýchliť prístup do tejto pamäte. V prípade, ak všetky vlákna *half-warpu* pristupujú do rovnakého segmentu, bude tento prístup spoločný a uskutočnený ako jedna transakcia. Ak však vlákna čítajú dáta z n rôznych častí pamäte, je realizovaných n transakcií prístupu. Možno povedať, že pokiaľ vlákna budú pristupovať náhodne do rôznych častí pamäte, dôjde k výraznému spomaleniu celého výpočtu. A naopak, ak budú čítať dáta z jedného bloku pamäte, môžu byť požiadavky zlúčené a spracované dostatočne rýchlo. Tento princíp sa nazýva združený prístup do pamäte (*coalesced memory access*).

Kapitola 3

Osvetlenie a tieňovanie

Vďaka rôznym intenzitám odrazu svetla sme schopný vnímať trojrozmerný tvar objektov. Spôsob odrazu svetla na povrchu týchto objektov popisujú rôzne osvetľovacie modely. Rovnako dôležitú úlohu pri zobrazení objektov má aj tieňovanie, ktoré je potrebné predovšetkým pre urýchlenie výpočtu osvetlenia a určenie farby zvyšných bodov objektu. Postupne popíšem rôzne metódy osvetlenia a tieňovania, ich výhody a nevýhody.

3.1 Osvetľovacie modely

Osvetľovací model je matematická funkcia, ktorá vyjadruje intenzitu rozptýleného svetla v závislosti na jeho smere a na smere, intenzite a vlnovej dĺžke dopadajúceho svetla. Osvetľovacie modely môžeme rozdeliť do dvoch základných skupín:

- **fyzikálne modely** – realistický popis vlastností svetla a jeho odrazu, výpočtovo náročné, nevhodné pre zobrazenie v reálnom čase,
- **empirické modely** – značne zjednodušené, rýchle, vhodné pre real-time vykreslenie trojrozmernej scény.

3.1.1 BRDF

BRDF (*Bidirectional Reflectance Distribution Function*) je dvojsmerná odrazová distribučná funkcia, ktorá je dôležitým prvkom pri popise povrchu objektov scény. Bola popísaná v roku 1977 [18] a patrí medzi fyzikálne modely osvetlenia. Ide o šesťrozmernú matematickú funkciu, ktorá popisuje vlastnosť materiálu odrážajúceho svetelnú energiu. Je definovaná ako pomer odrazeného a dopadajúceho žiarenia:

$$f_r(x, \vec{w_i}, \vec{w_o}) = \frac{\mathrm{d}L_r(x, \vec{w_o})}{\mathrm{d}L_i(x, \vec{w_i})}, \tag{3.1}$$

kde:

$L_r(x, \vec{w_o})$ – odrazené žiarenie v bode x,

$L_i(x, \vec{w_i})$ – ožiarenie povrchu v bode x,

$\vec{w_o}$ – smer odrazeného žiarenia (môže byť vyjadrený ako dvojica uhlov θ_o a φ_o),

$\vec{w_i}$ – smer, z ktorého žiarenie dopadá (podobne môže byť vyjadrený pomocou uhlov θ_i a φ_i),

$\vec{w_o}$ a $\vec{w_i}$ – sú definované vzhľadom k povrchovej normále v bode x,

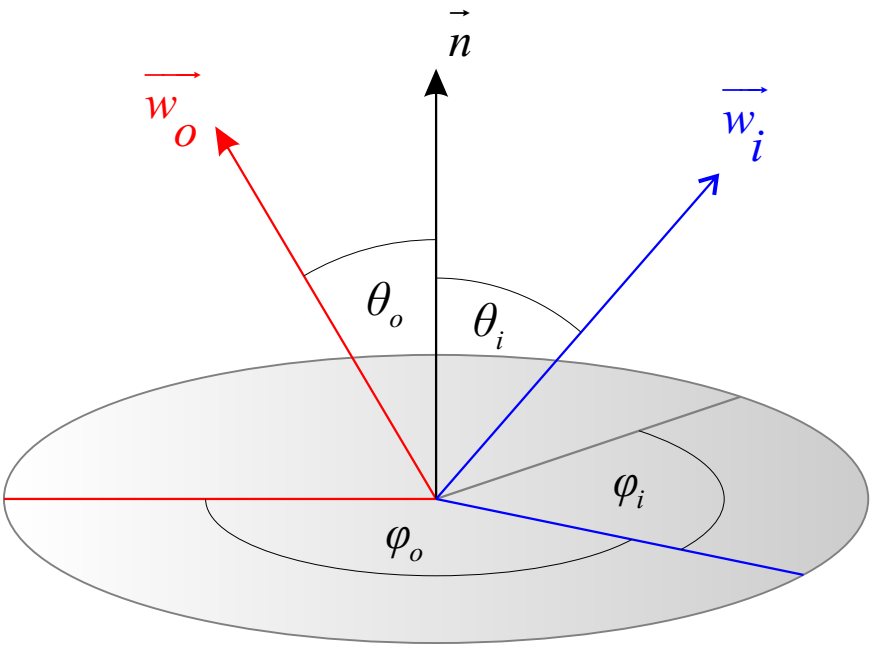
x – bod povrchu, ktorý môže byť vyjadrený pomocou dvojice súradníc.

Funkciu (3.1) možno prepísať do tvaru:

$$f_r(x, \vec{w_i}, \vec{w_o}) = \frac{\mathrm{d}L_r(x, \vec{w_o})}{L_i(x, \vec{w_i}) \cos\theta_i \, \mathrm{d}\vec{w_i}}, \tag{3.2}$$

kde θ_i reprezentuje uhol medzi normálov povrchu $\vec{n}$ v bode x a smerom dopadajúceho žiarenia $\vec{w_i}$ ($\cos\theta_i = \vec{n} \cdot \vec{w_i}$).

Výpočet BRDF je veľmi zložitý a časovo náročný. Vlastnosti niektorých povrchov nemožno vyjadriť analyticky. Takéto materiály sú vyjadrené pomocou tabuľky, ktorá pre dané parametre obsahuje pomer medzi prijatým a odrazeným svetlom. Vďaka tomu dokáže model zachytiť rôzne fyzikálne vlastnosti materiálov. Jednou z takýchto vlastností je anizotropia, ktorá vyjadruje závislosť odrazeného svetla od smeru natočenia povrchu okolo normálového vektora. Základný princíp je znázornený na obrázku 3.1.



Obrázok 3.1: Základný princíp BRDF a anizotropie

Fyzikálne založená BRDF spĺňuje niekoľko vlastností. Odhliadnuc od vlastnej emisie svetla platí zákon zachovanie energie, podľa ktorého nemôže povrch odrážať väčšie množstvo energie, ako na daný povrch dopadá. Preto platí:

$$\forall \vec{w_i}, \int_\Omega f_r(x, \vec{w_i}, \vec{w_o}) \cos\theta_o \, \mathrm{d}w_o \leq 1. \tag{3.3}$$

Taktiež je dôležité si uvedomiť, že hodnota BRDF je vždy nezáporná:

$$f_r(x, \vec{w_i}, \vec{w_o}) \geq 0. \tag{3.4}$$

Dôležitou vlastnosťou je *Helmholtzov princíp reciprocity*, podľa ktorého sa hodnota funkcie pri výmene smeru dopadajúceho a odrazeného žiarenia nemení. Táto skutočnosť nám umožňuje využitie BRDF v algoritmoch sledovania lúča od pozorovateľa, tak aj od svetelných zdrojov. Helmholtzov princíp môžeme vyjadriť vzťahom:

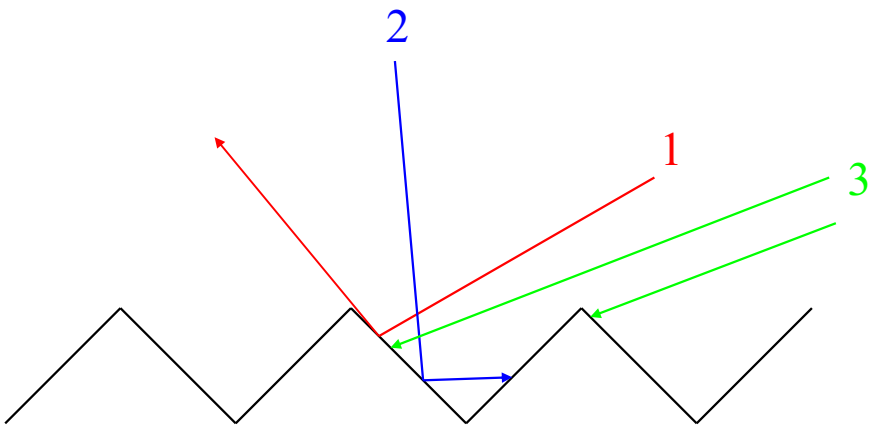
$$f_r(x, \vec{w_i}, \vec{w_o}) = f_r(x, \vec{w_o}, \vec{w_i}). \tag{3.5}$$

3.1.2 Model Torrance-Sparrow

Model vhodný pre realistické zobrazenie, ktorý bol publikovaný v práci [23]. Podstatou modelu je povrch zložený z veľkého množstva miniatúrnych plôch, ktoré sú rôzne orientované. Jednotlivé plochy sa môžu pri veľkých uhloch pohľadu pozorovateľa alebo dopadu svetla navzájom zatieňovať, čo vedie k rozdielnym intenzitám odrazu svetla v rôznych smeroch. Týmto osvetľovacím modelom je simulovaná nepravidelná štruktúra povrchu telesa.

Na obrázku 3.2 je viditeľné vzájomné tienenie plôch na povrchu:

1. lúče sa normálne odrazia,
2. časť odrazeného svetla je zachytená protiľahlou plochou (nastáva v prípade, ak sa smerový vektor pozorovateľa priblíži k rovine plochy),
3. časť dopadajúceho svetla je zatienená inou plochou (smerový vektor dopadajúceho svetla sa priblíži k rovine plochy).



Obrázok 3.2: Princíp modelu Torrance-Sparrow

3.1.3 Lambertov osvetľovací model

Jeden z najjednoduchších empirických modelov, ktorý obsahuje jedinú, difúznu zložku. Ide o časť svetla, ktorá dopadá na teleso zo svetelného zdroja a odráža sa rovnomerne do všetkých smerov (odraz od matného povrchu). Intenzita difúznej zložky je tým väčšia, čím je smer dopadajúceho svetla bližší k povrchovej normále (tzn. výsledná intenzita závisí na uhle dopadu lúča na povrch, nie je však závislá na pozícii pozorovateľa). Používa sa pre jednoduché vykreslenie objektu v scéne. Hlavnou výhodou je rýchly výpočet, osvetlenie však nezodpovedá realite.

Pre určenie intenzity odrazeného svetla sa používa Lambertovo pravidlo [3]:

$$I = Lk\cos\theta = Lk(\vec{l}\cdot\vec{n}), \tag{3.6}$$

kde:

I – výsledná intenzita odrazeného svetla,

L – intenzita dopadajúceho svetla,

k – koeficient odrazu materiálu,

θ – uhol medzi vektorom určujúceho smer svetelného zdroja a normálou povrchu,

$\vec{l}$ – vektor určujúci smer svetelného zdroja,

$\vec{n}$ – normála povrchu.

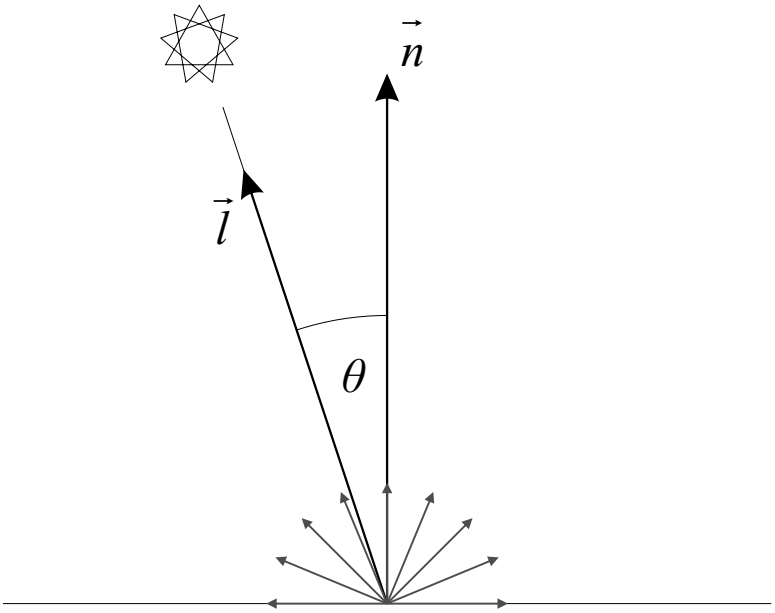
Pre uhol $\theta > \frac{\pi}{2}$ ($\cos\theta < 0$) nedochádza k odrazu, pretože svetlo dopadá na odvrátenú plochu objektu. Lambertov zákon 3.6 je preto možné upraviť na tvar:

$$I = Lk\max((\vec{l}\cdot\vec{n}), 0). \tag{3.7}$$

Na obrázku 3.3 je vykreslený princíp šírenia difúznej zložky svetla a označenie jednotlivých veličín použitých vo vzorci 3.6 pre výpočet výslednej intenzity.

3.1.4 Phongov osvetľovací model

Najpoužívanejší a najznámejší variant empirického modelu, ktorý navrhol Bui Tuong Phong vo svojej dizertačnej práci [19] a neskôr publikoval [20]. Na rozdiel od Lambertovho osvetľovacieho modelu obsahuje okrem difúznej zložky aj ambientnú a spekulárnu zložku. Snaží sa čo najviac zjednodušiť fyzikálny model a optické vlastnosti svetla tak, aby bol výpočet čo najrýchlejší a výsledný obraz čo najrealistickejší. Má podporu hardvérovej akcelerácie a je implementovaný v mnohých knižniciach (napr. v OpenGL).



Obrázok 3.3: Lambertov osvetľovací model

Ambientná zložka

Intenzita tzv. okolitého svetla (svetelný šum), ktoré vzniklo mnohonásobným odrazom od ostatných telies. Simuluje odrazené a rozptýlené svetlo v scéne. Dovoľuje, aby odvrátené povrchy od svetelného zdroja a povrchy v tieni mali zachovanú svoju farbu (neboli úplne čierne). Pre ambientnú zložku platí:

$$I_a = L_a k_a, \tag{3.8}$$

kde:

I_a – výsledná intenzita ambientnej zložky,

L_a – intenzita ambientnej zložky svetla v scéne,

k_a – koeficient ambientnej zložky materiálu.

Difúzna zložka

Je zhodná s Lambertovým osvetľovacím modelom. Ide o časť sveta, ktorá dopadá na teleso zo svetelného zdroja a odráža sa rovnomerne do všetkých smerov. Pre intenzitu difúznej zložky platí:

$$I_d = L_d k_d(\vec{l} \cdot \vec{n}), \tag{3.9}$$

kde:

I_d – výsledná intenzita difúznej zložky,

L_d – intenzita difúznej zložky dopadajúceho svetla,

k_d – koeficient difúznej zložky materiálu,

$\vec{l}$ – vektor určujúci smer svetelného zdroja,

$\vec{n}$ – normála povrchu.

Spekulárna (zrkadlová) zložka

Modeluje odraz svetla na lesklom povrchu, pričom vznikajú odlesky. To umožňuje lepšie vnímanie zakrivenosti povrchu a umiestnenie jednotlivých svetelných zdrojov. Odrazené svetlo sa sústreďuje v okolí vektoru odrazu. Výsledná intenzita závisí nielen na polohe svetelného zdroja, ale aj polohe pozorovateľa, a preto platí:

$$I_s = L_s k_s (\vec{r} \cdot \vec{v})^{\alpha}, \tag{3.10}$$

kde:

I_s – výsledná intenzita spekulárnej zložky,

L_s – intenzita spekulárnej zložky dopadajúceho svetla,

k_s – koeficient spekulárnej zložky materiálu,

$\vec{r}$ – smer dokonale zrkadlového odrazu,

$\vec{v}$ – vektor určujúci smer k pozorovateľovi,

α – koeficient zrkadlového odrazu materiálu.

Výsledná intenzita

Výsledná intenzita odrazeného svetla je súčtom všetkých troch zložiek 3.8, 3.9 a 3.10. Pre každú farebnú zložku RGB (t.j. červená, zelená a modrá) sa počíta výsledná intenzita nezávisle na ostatných zložkách. Výsledný vzťah pre intenzitu každej farebnej zložky je

$$I = I_a + I_d + I_s. \tag{3.11}$$

Tento vzťah môžeme rozpísať do konečnej podoby:

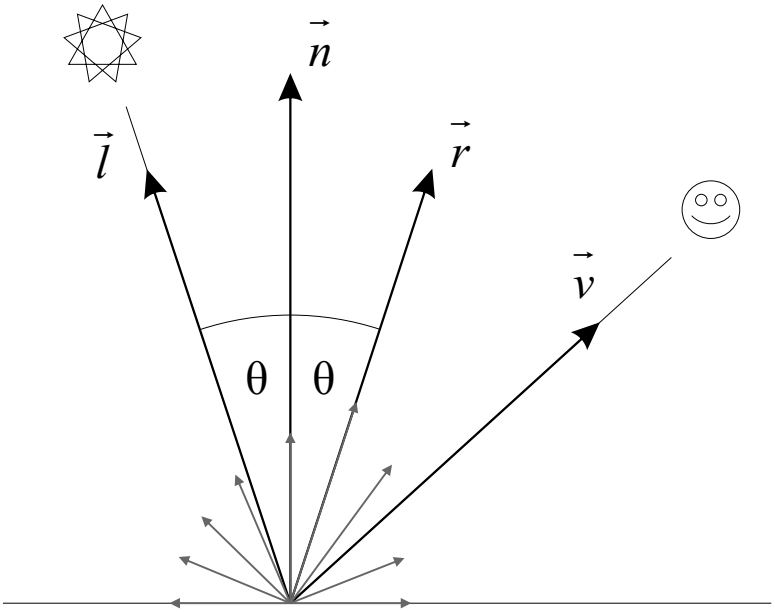
$$I = L_a k_a + \sum_{i=0}^{m} (L_d k_d (\vec{l_i} \cdot \vec{n}) + L_s k_s (\vec{r_i} \cdot \vec{v})^{\alpha}), \tag{3.12}$$

kde:

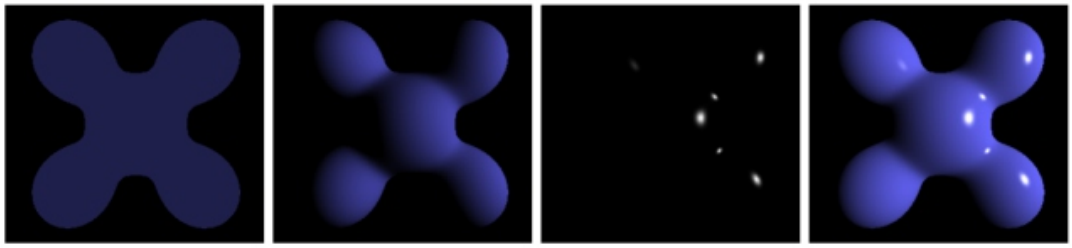
m – počet svetelných zdrojov v scéne,

i – index svetelného zdroja (určuje vlastnosti pre daný svetelný zdroj).

Na nasledujúcom obrázku 3.4 je znázornený princíp Phongovho osvetľovacieho modelu a označenie jednotlivých veličín použitých vo vzorci 3.12 pre výpočet výslednej intenzity svetla. Na obrázku 3.5 sú znázornené jednotlivé zložky svetla (ambientná, difúzna a spekulárna) a výsledná intenzita svetla vypočítaná pomocou Phongovho osvetľovacieho modelu. Obrázok bol prevzatý zo zdroja [4] a čiastočne upravený.



Obrázok 3.4: Phongov osvetľovací model



Obrázok 3.5: Jednotlivé zložky Phongovho osvetľovacieho modelu (zľava doprava: ambientná zložka + difúzna zložka + spekulárna zložka = výsledná intenzita)

3.2 Tieňovanie (*shading*)

Slúži pre lepšiu reprezentáciu (zobrazenie) trojrozmerných objektov a rýchlejšie vykresľovanie. Pre niekoľko bodov na povrchu objektu sa pomocou jedného z osvetľovacích modelov vypočíta farba, pričom ostatné body nadobúdajú farbu určenú pomocou algoritmu tieňovania. Na obrázku 3.6 prevzatého z [1] je jednoduchý objekt vykreslený pomocou troch rôznych metód tieňovania: konštantné, Gouraudovo a Phongovo. Každá zo spomenutých metód má svoje výhody a nevýhody. Nemožno povedať, ktorá je lepšia alebo horšia. Sú určené predovšetkým pre výpočet tieňovania objektov tvorených množinou rovinných plôch (polygónov).

Obrázok 3.6: Rôzne metódy tieňovania (zľava doprava: konštantné, Gouraudovo, Phongovo)

3.2.1 Konštantné tieňovanie (*flat shading*)

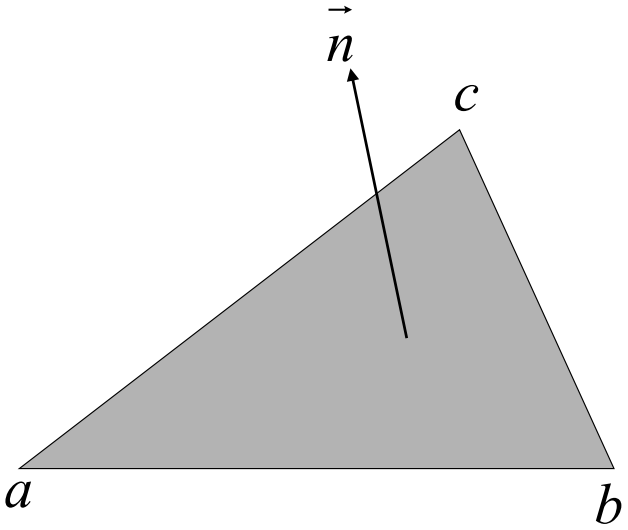
Algoritmus predpokladá, že každá plocha má práve jednu normálu, ako je znázornené na obrázku 3.7. Pomocou osvetľovacej rovnice určíme farbu jedného bodu danej plochy (polygónu). Výsledná farba celého polygónu je zhodná s farbou tohto bodu. Ide o najjednoduchšiu a najrýchlejšiu metódu, kvalita však nezodpovedá reálnemu zobrazeniu. Metóda zachováva polygonálnu štruktúru objektov.

3.2.2 Gouraudovo tieňovanie

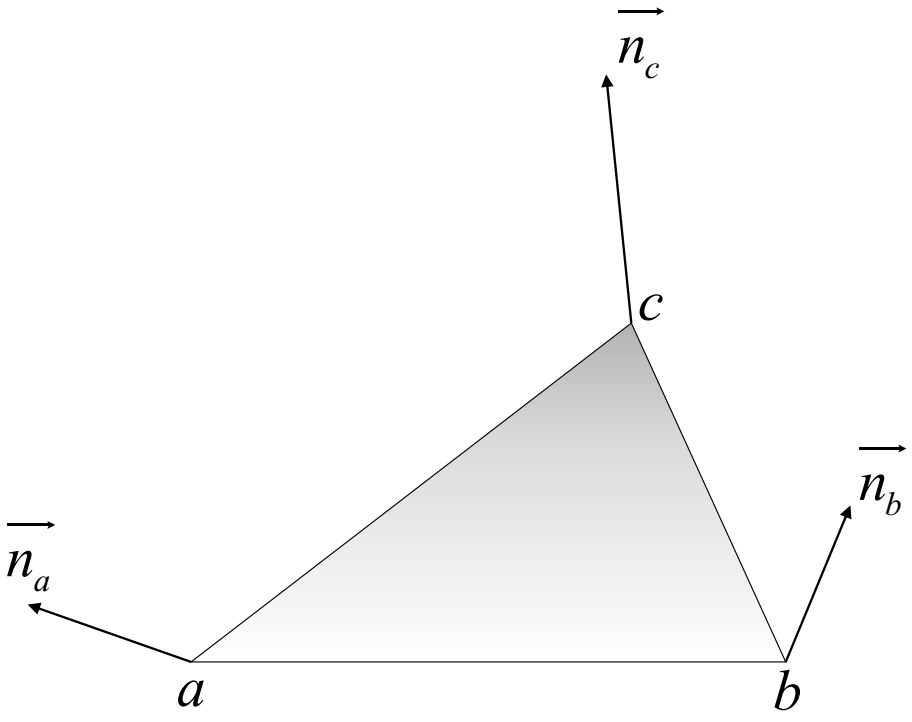
Metóda bola prvýkrát predstavená v roku 1971 H. Gouraudom [10]. Základný princíp spočíva vo vypočítaní osvetlenia vo vrcholoch polygónu a následnej interpolácie týchto hodnôt (pozri obrázok 3.8). Hlavnými výhodami tohto postupu sú rýchlosť zobrazenia, implementácia v hardvéri a plynulé tieňovanie zakrivených povrchov tvorených množinou polygónov. Nevýhodou ostáva pozorovateľná polygonálna štruktúra a vznik nerealistických odleskov.

3.2.3 Phongovo tieňovanie

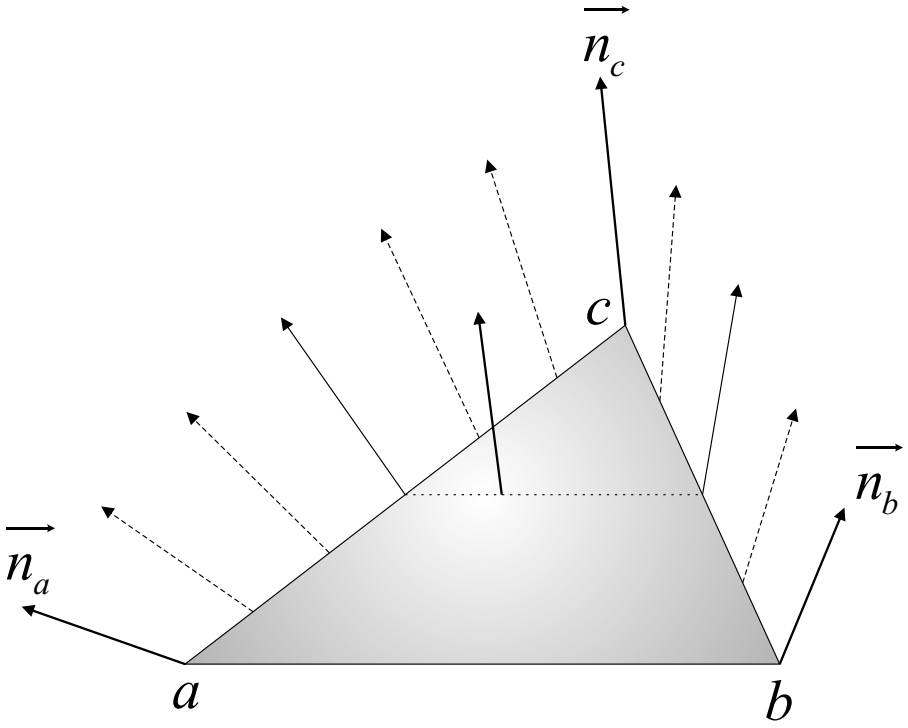
Táto implementácia bola popísaná spoločne s Phongovým osvetľovacím modelom v roku 1973 [19]. Metóda pracuje s normálami v každom vrchole daného polygónu, ale na rozdiel od Gourauda sú tieto hodnoty ďalej interpolované pre každý bod a až následne je vypočítaná farba pomocou jednej z osvetľovacích rovníc. Z uvedených troch metód je táto implementácia najzložitejšia a najnáročnejšia. Phongovo tieňovanie zvláda vykreslenie hladkých povrchov a taktiež dokáže vykresliť realistické odlesky vznikajúce na povrchu telesa. Na obrázku 3.9 je znázornený princíp tejto metódy.



Obrázok 3.7: Konštantné tieňovanie



Obrázok 3.8: Gouraudovo tieňovanie



Obrázok 3.9: Phongovo tieňovanie

Kapitola 4

Globálne zobrazovacie metódy

V nasledujúcej kapitole popíšem rovnicu zobrazenia, ktorá tvorí základ každej zobrazovacej metódy. Ďalej sa budem venovať jednotlivým výpočtom globálneho osvetlenia, budem diskutovať o ich výhodách, nevýhodách a kvalite výsledného zobrazenia.

4.1 Zobrazovacia rovnica

Ide o matematický aparát popisujúci vykreslenie obrazu ako výpočet rovnovážneho stavu prenosu svetla. Rovnica bola prvýkrát uverejnená J. T. Kajiyaou [15] a vyjadrená pomocou intenzity. V súčasnosti sa rovnica definuje pomocou radiácie. Predpokladá sa, že prenos svetla sa uskutočňuje vo vákuu (tzn. bez interakcie s prostredím), svetlo sa šíri nekonečnou rýchlosťou a rovnovážny stav prenosu svetla nastáva okamžite.

Rovnica udáva pre každý bod povrchu a pre všetky smery množstvo emitovaného žiarenia a žiarenia prichádzajúceho z ostatných častí scény, ktoré bolo odrazené. Rovnica je iným vyjadrením fyzikálneho zákona zachovania energie a má tvar:

$$L_o(x,\vec{\omega}) = L_e(x,\vec{\omega}) + L_r(x,\vec{\omega}), \tag{4.1}$$

$$L_o(x,\vec{\omega}) = L_e(x,\vec{\omega}) + \int_\Omega f_r(x,\vec{\omega}',\vec{\omega}) L_i(x,\vec{\omega}') \cos\theta \, \mathrm{d}\vec{\omega}', \tag{4.2}$$

kde:

$L_o(x,\vec{\omega})$ – celková odchádzajúca radiácia v bode x v smere $\vec{\omega}$,

$L_e(x,\vec{\omega})$ – množstvo emitovaného žiarenia v bode x v smere $\vec{\omega}$,

$L_r(x,\vec{\omega})$ – radiácia odrazená v bode x v smere $\vec{\omega}$,

$L_i(x,\vec{\omega}')$ – dopadajúca radiácia v bode x zo smere $\vec{\omega}'$,

$\vec{\omega}$ – smer odchádzajúcej radiácie,

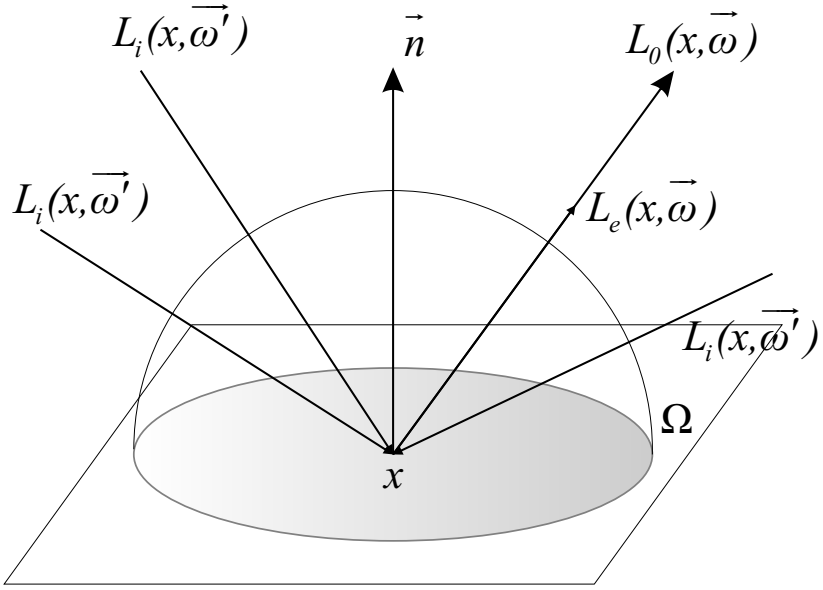
$\vec{\omega}'$ – smer dopadajúcej radiácie,

$\int_\Omega \mathrm{d}\vec{\omega}'$ – integrál cez hemisféru všetkých smerov dopadajúcej radiácie,

$f_r(x,\vec{\omega}',\vec{\omega})$ – BRDF funkcia udávajúca pre každý dopadajúci a odchádzajúci smer množstvo odrazenej radiácie,

θ – uhol medzi smerom dopadajúcej radiácie a povrchovou normálou $\vec{n}$ v bode x ($\cos\theta = (\vec{n} \cdot (-\vec{\omega}'))$).

Rovnica 4.2 patrí medzi *Fredholmove rovnice druhého rádu*. Analytické riešenie tejto rovnice je pre reálne scény nemožné a preto sa používajú rôzne aproximácie od klasických postupov riešenia zložitých rovníc až po hrubé aproximačné riešenia nahradzujúce integrál iba jednou vzorkou v smere bodového svetla. Všetky globálne zobrazovacie metódy sú čiastočným riešením tejto rovnice a ich kvalitu môžeme posudzovať podľa presnosti aproximácie. Grafický popis rovnice je vyobrazený na obrázku 4.1.



Obrázok 4.1: Grafický popis rovnice 4.2

Zobrazovaciu rovnicu môžeme definovať aj v závislosti na vzájomnej pozícii objektov v scéne. Aby sme ju však mohli vyjadriť v tomto tvare, potrebujeme najskôr definovať tzv. funkciu viditeľnosti a vzájomnú polohu dvoch bodov.

Funkcia viditeľnosti nadobúda hodnoty $\{0, 1\}$ a je definovaná vzťahom:

$$V(x,y) = \begin{cases} 1 & \text{ak je medzi bodmi } x \text{ a } y \text{ priama viditeľnosť,} \\ 0 & \text{pokiaľ medzi bodmi } x \text{ a } y \text{ nie je priama viditeľnosť,} \end{cases} \tag{4.3}$$

kde x a y sú dva rozdielne body scény. Funkcia viditeľnosti je v raytracingu známa ako tieňové lúče (*shadow rays*).

Ďalším faktorom ovplyvňujúcim množstvo radiácie dopadajúcej na povrch x z bodu y je vzájomná poloha týchto bodov. Vzájomnú polohu bodov x a y môžeme vyjadriť vzťahom:

$$G(x,y) = \frac{\cos\theta\cos\theta'}{r_{xy}^2} = \frac{(\vec{n_x} \cdot (-\vec{\omega}'))(\vec{n_y} \cdot \vec{\omega}')}{r_{xy}^2}, \tag{4.4}$$

kde:

$G(x, y)$ – člen daný relatívnou geometriou povrchov v bode x a y,

$\vec{n_x}$ – normála povrchu v bode x,

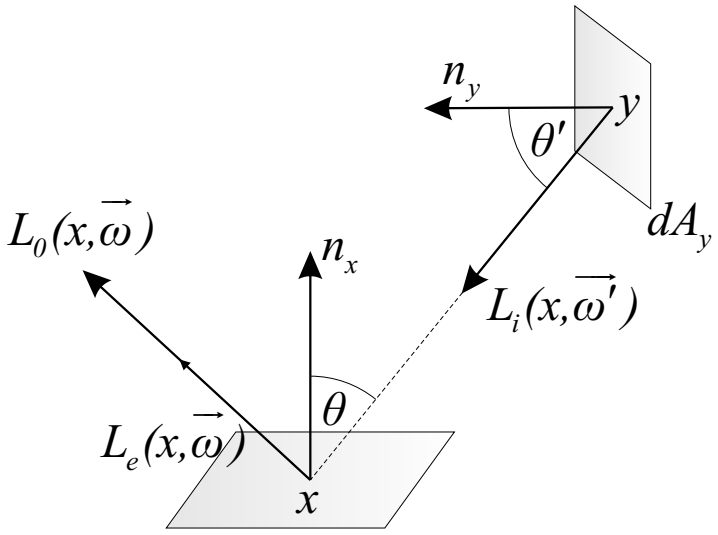
$\vec{n_y}$ – normála povrchu v bode y,

r_{xy} – vzdialenosť medzi bodmi x a y.

Zobrazovaciu rovnicu môžeme konečne definovať ako:

$$L_o(x, \vec{\omega}) = L_e(x, \vec{\omega}) + \int_A f_r(x, \vec{\omega}', \vec{\omega}) L_i(x, \vec{\omega}') V(x, y) G(x, y) \, \mathrm{d}A_y, \tag{4.5}$$

kde $\int_A \mathrm{d}A_y$ je integrácia cez plošné elementy. Tento tvar nám dáva možnosť výpočtu založeného na diferenciálnych plochách. Schéma rovnice je znázornená na obrázku 4.2.



Obrázok 4.2: Grafický popis rovnice 4.5

4.2 Raytracing

Raytracing je základná metóda na prevod priestorovej scény do dvojrozmerného obrazu, ktorú ako prvý publikoval koncom 80-tych rokov Turner Whitted [25]. Je založená na fyzikálnom princípe sledovania svetla v priestore. Vychádza z metódy vrhania lúčov (*raycasting*), ktorá bola pôvodne určená pre zobrazovanie modelov vytvorených pomocou konštruktívnej geometrie.

Základný princíp raytracingu spočíva vo vrhaní lúčov od pozorovateľa (kamery) cez každý pixel premietacej roviny smerom do scény. Tieto lúče sa nazývajú primárnymi. Ak nastáva interakcia primárneho lúča s niektorým objektom (polygónom) v scéne, dochádza k vytvoreniu tzv. tieňových lúčov, poprípade sekundárnych (odrazených a lomených) lúčov v závislosti na vlastnostiach materiálu. Na základe osvetľovacieho modelu je nakoniec vypočítaná farba výsledného bodu (pixelu).

Tieňové lúče

Umožňujú vytvárať ostré tiene z bodových zdrojov svetla. Sú vysielané smerom od priesečníka primárneho lúča k svetelnému zdroju. Ak sa medzi priesečníkom a svetelným zdrojom nachádza ďalší objekt, daný bod priesečníka leží v zatienenej oblasti svetla. V opačnom prípade je bod osvetlený a tieňové lúče tak modelujú priame osvetlenie objektov v scéne.

Odrazené lúče

Sú potrebné pre zobrazenie zrkadlových odrazov. Smer odrazeného lúča je určený podľa zákonov optiky. Pre jeho určenie je potrebné poznať smer dopadajúceho lúča a normálu povrchu v bode dopadu. Ideálny (dokonalý) odraz môžeme vyjadriť ako:

$$\vec{\omega_r} \quad = \quad 2(\vec{\omega} \cdot \vec{n})\vec{n} - \vec{\omega}, \tag{4.6}$$

kde:

$\vec{\omega_r}$ – smer dokonale odrazeného lúča,

$\vec{\omega}$ – smer dopadajúceho lúča,

$\vec{n}$ – normála povrchu v bode dopadu.

Vzťah 4.6 je vektorovým vyjadrením zákona o odraze, ktorý hovorí, že uhol odrazu sa rovná uhlu dopadu, pričom odrazený lúč zostáva v rovine dopadu. Odrazený lúč sa stáva primárnym, čo umožňuje modelovať vzájomné zrkadlenie lesklých telies. Ak lúč narazí na matný povrch, je vypočítaná jeho farba. Vysielanie odrazených lúčov by mohlo pokračovať do nekonečna, preto je potrebné stanoviť vrchný limit počtu odrazení. Typickým lesklým materiálom je chróm.

Lomené lúče

Vznikajú po priechode materiálu a môžeme pomocou nich modelovať priehľadné objekty. Princíp šírenia je podobný ako u odrazených lúčov, kde sa lomené lúče stávajú tiež primárnymi. Sú vypočítané podľa *Snellovho zákonu lomu*, ktorý hovorí, že ak lúč prechádza z prostredia s indexom lomu n_1 pod uhlom θ_1 do prostredia s indexom lomu n_2, zalomí sa pod uhlom θ_2:

$$\sin\theta_1 n_1 = \sin\theta_2 n_2. \tag{4.7}$$

Vzťah 4.7 môžeme upraviť a prepísať do vektorovej podoby:

$$\vec{\omega_s} = n_p\vec{\omega} + (n_p \cos\theta - \sqrt{1 - n_p^2(1 - \cos\theta^2)})\vec{n}, \tag{4.8}$$

kde:

$\vec{\omega_s}$ – smer lomeného lúča,

$\vec{\omega}$ – smer dopadajúceho lúča,

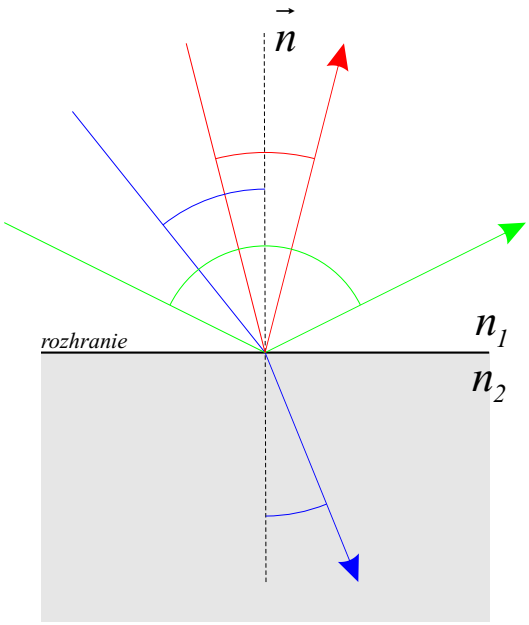
n_p – index lomu prostredia ($n_p = \frac{n_1}{n_2}$),

θ – uhol medzi dopadajúcim lúčom a povrchovou normálou ($\cos\theta = \vec{n} \cdot (-\vec{w})$),

$\vec{n}$ – normála povrchu v bode dopadu.

Ak je výraz pod odmocninou vo vzorci 4.8 záporný, uhol dopadu je väčší ako medzný uhol a dochádza k tzv. úplnému odrazu. Typickým priehľadným materiálom je sklo s indexom lomu $1,3$.

Na obrázku 4.3 je znázornený princíp šírenia svetla na rozhraní dvoch prostredí. Červenou farbou je označený odrazený lúč, modrou farbou lomený lúč a zelenou farbou lúč, ktorý vznikol pri úplnom odraze. Informácie a jednotlivé vzťahy pre odraz a lom svetla boli čerpané z článku [11].

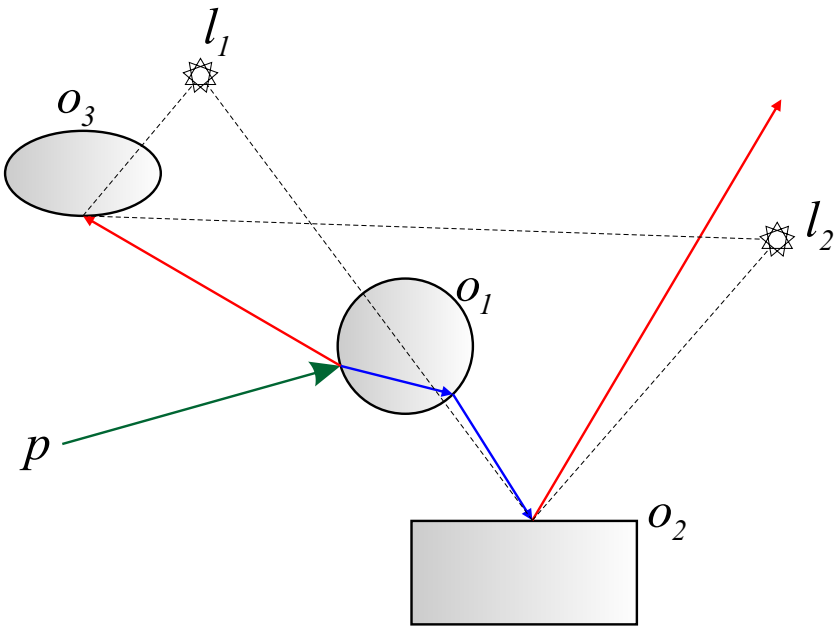


Obrázok 4.3: Odraz a lom svetla

Pre jeden primárny lúč je niekedy potrebné sledovať veľké množstvo sekundárnych lúčov, čo je výpočtovo veľmi náročné. Princíp sledovania lúčov môžeme vidieť na obrázku 4.4. Primárny lúč je znázornený zelenou farbou. Sekundárne lúče sú označené červenou (odrazené lúče) a modrou farbou (lomené lúče). Tieňové lúče smerujú vždy k svetelným zdrojom a sú modelované čiarkovanou úsečkou. Celá scéna je tvorená dvoma svetelnými zdrojmi (l_1, l_2) a troma objektmi (o_1, o_2, o_3).

Pre implementáciu raytracingu je vhodné použiť rekurziu. Ukončenie rekurzívneho volania je možné pomocou jednej z nasledujúcich podmienok:

- neexistuje priesečník lúča so žiadnym objektom (polygónom),
- priesečník lúča s čisto difúznym materiálom,
- je dosiahnutá vopred stanovená hodnota zanorenia (rekurzie),
- energia lúča klesne pod určitú hodnotu.



Obrázok 4.4: Sledovanie lúčov (raytracing)

Princíp algoritmu možno popísať nasledujúcim pseudokódom 4.1:

Rekurzívna funkcia `Raytrace(R,Z)` vracajúca farbu pixela s parametrami primárneho lúča `R` a hĺbkou zanorenia `Z`:

1. Nájdenie priesečníka `P` medzi primárnym lúčom `R` a scénou.
2. Ak priesečník `P` neexistuje, lúč prechádza scénou a funkcia vracia farbu pozadia.
3. Vyslanie tieňových lúčov z priesečníka `P` ku všetkým zdrojom svetla.
4. Vyhodnotenie farby $\texttt{I}_p$ priesečníka `P` pomocou osvetľovacieho modelu.
5. Ak nie je prekročená hĺbka rekurzie `Z`:
   (a) zavolanie funkcie $\texttt{I}_r$ `= Raytrace(`$\texttt{R}_r$`,Z+1)`, kde $\texttt{R}_r$ je odrazený lúč,
   (b) zavolanie funkcie $\texttt{I}_s$ `= Raytrace(`$\texttt{R}_s$`,Z+1)`, kde $\texttt{R}_s$ je lomený lúč.
6. Funkcia vracia výslednú farbu pixela $\texttt{I} = \texttt{I}_p + \texttt{I}_r + \texttt{I}_s$.

Algoritmus 4.1: Rekurzívna funkcia sledovania lúča

Ak má raytracing hĺbku zanorenia rovnú 1, vyhodnocujú sa iba primárne lúča a raytracing je zhodný s metódou raycasting. Hlavnou výhodou raytracingu je jeho jednoduchosť, nie je však vhodný pre realistické zobrazenie. Pracuje s bodovými zdrojmi svetla, ktoré neumožňujú vytváranie mäkkých tieňov. Šírenie svetla je možné len pomocou zrkadlových odrazov, čo má za následok nesprávne modelovanie nepriameho osvetlenia.

4.3 Distributed raytracing

V klasickom raytracingu je BRDF funkcia zjednodušená na dokonale zrkadlové odrazy a integrál v zobrazovacej rovnici je nahradený jednou vzorkou v smere bodového svetla. Tieto a ďalšie nedostatky sa snaží odstrániť metóda nazývaná *distributed raytracing* (distribuované/stochastické sledovanie lúča), ktorá bola predstavená Robertom L. Cookom, Thomasom Porterom a Lorenom Carpenterom [7] v roku 1984.

Distibuted raytracing, na rozdiel od klasického raytracingu, vysiela z jedného pixelu niekoľko lúčov, ktoré pri dopade na objekt generujú mierne odlišné sekundárne lúče. Použitie skutočnej BRDF funkcie umožňuje realistické zobrazenie lesklých povrchov, ktoré nemajú dokonalý zrkadlový odraz. Taktiež umožňuje modelovanie plošných zdrojov svetla. Tieňové lúče sú vysielané do rôznych častí svetelného zdroja, čím vznikajú kvalitné mäkké tiene.

Metódu je možné použiť pre simulovanie špeciálnych efektov, akými sú napr. hĺbka ostrosti (*depth of field*) alebo rozmazanie pohybom (*motion blur*). Nevýhodou je nesprávne vyhodnocovanie nepriameho osvetlenia. Aj napriek tomu distibuted raytracing predstavuje elegantný algoritmus, ktorý je schopný generovať takmer fotorealistické obrazy.

4.4 Pathtracing

Metóda sledovania lúča neimplementuje celú zobrazovaciu rovnicu. Nevýhoda spočíva v sledovaní priameho šírenia svetla. V skutočnosti má ale každý objekt difúzny odraz, čo znamená, že objekty sa navzájom čiastočne osvetľujú. V raytracingu je táto skutočnosť aproximovaná tzv. ambientnou zložkou v osvetľovacom modeli, čo však predstavuje veľmi hrubé zjednodušenie. Pre skutočné modelovanie nepriameho osvetlenia exituje metóda *pathtracing* (metóda sledovanie ciest) [22]. Na rozdiel od rekurzívneho sledovania lúča poskytuje kompletné riešenie zobrazovacej rovnice, avšak za cenu veľkého výpočtového výkonu.

Pathtracing využíva pre sledovanie ciest metódu *Monte Carlo* [24] a v každom priesečníku sleduje práve jeden lúč. Okrem odrazu od lesklých povrchov simuluje aj difúzne odrazy. Nevýhoda metódy spočíva v zanesení šumu do obrazu a pre kvalitné zobrazenie je potrebný veľký počet lúčov na každý pixel.

4.5 Photon mapping

Fotónové mapy (*photon maps*) predstavujú obojsmernú metódu zobrazovania vytvorenú H. Jensenem [14]. Metóda pozostáva z dvoch fáz. V prvej fáze prebieha tzv. sledovanie fotónov, ktoré sú vysielané zo svetelného zdroja. Smer, pozícia a energia emitovaných fotónov sú dané práve charakteristikou tohto zdroja. Fotóny prechádzajú scénou podobne ako lúče pri raytracingu. Pri interakcii fotónu s objektom môže dôjsť k odrazu, lomu alebo absorpcii. Ktorý z týchto troch prípadov nastane závisí na vlastnostiach materiálu. Výber môže byť uskutočnený napr. pomocou metódy *ruskej rulety* [6]. Pri interakcii fotónu s povrchom je jeho intenzita zaznamenaná do dátovej štruktúry nazývanej fotónová mapa a so zníženou intenzitou je opätovne vyslaný do scény. Fotón sa šíri scénou, pokiaľ nebol absorbovaný alebo jeho intenzita neklesla pod danú hranicu. Druhá fáza je totožná so základným raytracingom, pričom intenzita svetla v bode priesečníka je odvodená z uložených fotónov.

Fotónovú mapu je možné rozdeliť na dve nezávislé vrstvy: globálnu a kaustikovú. Do kaustikovej mapy sú ukladané iba tie fotóny, ktoré prešli zrkadlovým odrazom alebo lomom a až potom narazili na difúzny povrch. Výhoda v tomto rozdelení spočíva v kvalitnejšom

výpočte kaustiky aj pre menší počet fotónov. Prináša to však komplikácie s detekciou jednotlivých fotónov a ich uloženie do jednej z týchto máp.

Metóda umožňuje simulovať mnoho javov reálneho sveta, akými sú kaustiky alebo mäkké tiene. Taktiež poskytuje veľmi kvalitné fotorealistické obrazy. Nevýhodou je vznik nechceného šumu a potreba veľkého počtu fotónov.

Kapitola 5

Akceleračné štruktúry

Základný princíp raytracingu spočíva v hľadaní priesečníkov lúčov s prvkami scény. V mojej implementácii sú to polygonálne objekty zložené z trojuholníkov. Najjednoduchší algoritmus sledovania lúča testuje priesečník lúča so všetkými trojuholníkmi v scéne, čo je časovo a výpočtovo veľmi náročné. Zníženie počtu hľadania priesečníkov nám umožňujú akceleračné štruktúry, ktoré delia priestor na menšie časti. Lúče prechádzajúce scénou sú testované na priesečníky s malými oblasťami, a ak je niektorá oblasť zasiahnutá, potom sú testované všetky trojuholníky vyskytujúce sa v tejto oblasti. Týmto je dosiahnuté obrovské urýchlenie vykreslenia jednoduchých ale aj komplexných scén. Na druhej strane však použitie týchto akceleračných štruktúr prináša problém ich samotnej stavby, čo má zásadný vplyv pri zobrazení dynamických scén.

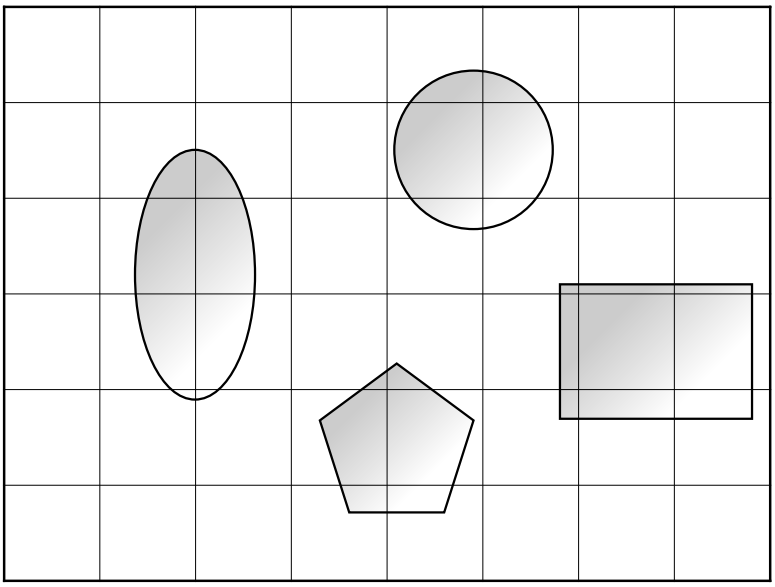
V súčasnosti existuje veľké množstvo rôznych typov akceleračných štruktúr a nie je jednoduché jednoznačne určiť, ktorá z nich je najlepšia alebo najvhodnejšia. Navzájom sa odlišujú rýchlosťou stavby a prechádzania, spôsobom organizácie objektov v scéne a ďalšími parametrami. Podľa spôsobu priradenia objektov do jednotlivých oblastí ich môžeme rozdeliť do nasledujúcich skupín:

1. štruktúry usporiadavajúce scénu do určitej hierarchie:
   - hierarchické obalové telesá (BVH),
2. rozdeľujúce priestor na menšie časti (priestorové delenie):
   - uniformná mriežka,
   - oktalový strom,
   - binárne priestorové delenie,
   - KD strom.

5.1 Uniformná mriežka

Uniformná (pravidelná) mriežka, ako už názov napovedá, rozdeľuje priestor na rovnako veľké neprekrývajúce sa časti. Ide o najjednoduchší spôsob delenia priestoru, ktorý v grafike popísal ako prvý A. Fujimoto [9]. Jednotlivé trojuholníky môžu byť umiestnené do viacerých buniek a ich počet v jednotlivých osách nemusí byť rovnaký. Na priechod lúča mriežkou sa používajú rôzne upravené DDA algoritmy. Jeden takýto algoritmus vo svojej práci publikovali J. Amanatides a A. Woo [5].

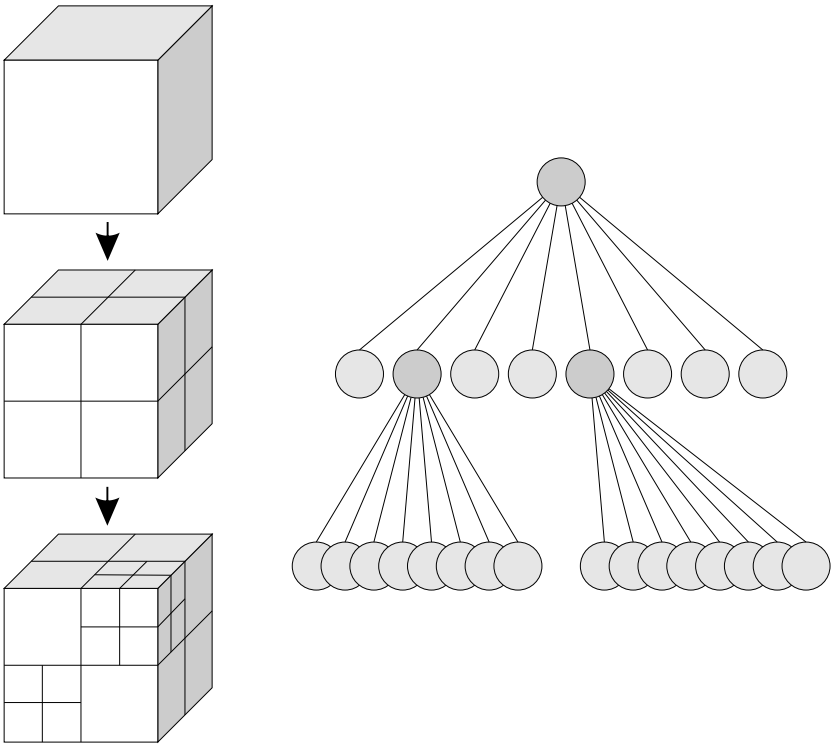
Výhoda mriežky spočíva nie len v jej jednoduchosti a rýchlosti stavby, ale aj v prechádzaní tejto štruktúry, pretože každá bunka má presne definovaných svojich susedov. Používa sa hlavne pre dynamické scény. Nevýhodou je nemožnosť meniť štruktúru mriežky podľa rozloženia objektov v scéne. Môže dochádzať k vzniku zbytočne prázdnych buniek, čo má za následok neefektívne využitie pamäte. Na druhej strane môžu vznikať bunky, ktoré obsahujú veľké množstvo trojuholníkov a ich rozdelenie by bolo oveľa účinnejšie. Uniformná mriežka najlepšie funguje v prípade, ak počet trojuholníkov bude v každej bunke približne rovnaký. Na obrázku 5.1 je znázornený princíp rozdelenia 2D priestoru pomocou uniformnej mriežky.



Obrázok 5.1: Uniformné delenie priestoru

5.2 Oktalový strom

Oktalový strom (*octree*) je hierarchická stromová štruktúra definovaná v [21]. Vznikla rozšírením *quadtree* pridaním tretieho rozmeru. Je založená na opakovanom delení priestoru na osem rovnakých buniek pomocou troch rovín kolmých na súradné osy. Každá bunka je reprezentovaná ako uzol stromu a každý jej podpriestor ako synovské uzly (pozri obrázok 5.2). Zostavenie tejto štruktúry je vzhľadom na pevné umiestnenie deliacich plôch rýchle. Každý rodičovský uzol má nula až osem potomkov v závislosti na tom, či daný synovský uzol obsahuje nejaké objekty alebo nie. Uzol bez potomkov sa nazýva listom a je to jediný uzol, ktorý obsahuje odkazy na objekty nachádzajúce sa vo vnútri daného podpriestoru. Octree sa využíva predovšetkým v dynamických scénach, kde je možné pristupovať k jednotlivým podpriestorom a tieto prebudovávať na základe lokálneho pohybu objektov.

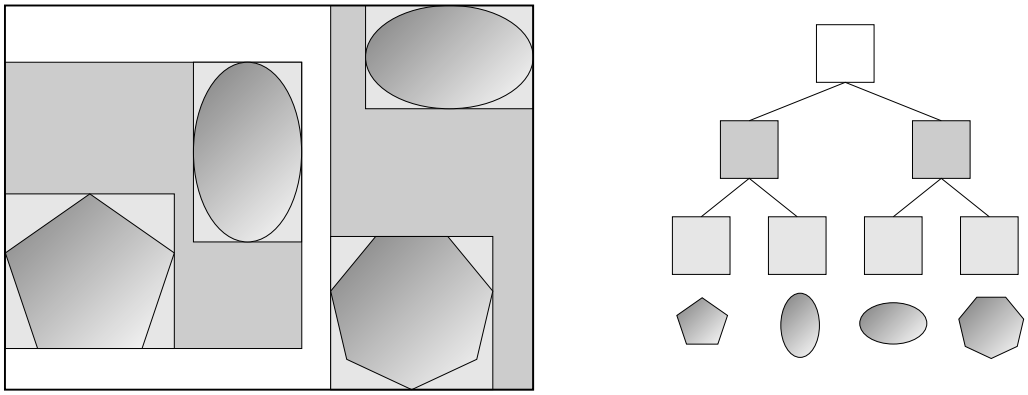


Obrázok 5.2: Rozdelenie priestoru pomocou oktalového stromu

5.3 Obalové telesá

V prípade zložitých operácií výpočtu priesečníka lúča s objektom sa často využívajú tzv. obálky objektov. Pri hľadaní priesečníka sa vždy najskôr otestuje interakcia lúča s danou obálkou a až potom sa hľadá skutočný priesečník s objektom. Existuje niekoľko druhov obálok, pričom sa od seba navzájom odlišujú svojím tvarom, zložitosťou výpočtu a presnosťou, ako daný objekt dokážu obaliť. Výber tvaru obálky je dôležitým faktorom ovplyvňujúcim výslednú rýchlosť zobrazenia a je závislý od geometrie vstupných objektov. Jednou z mnohých reprezentácií obálky je osovo rovnobežný kváder. Medzi jeho hlavnú výhodu patrí rýchly výpočet priesečníka s lúčom, nie je však schopný dobre ohraničiť jednotlivé vstupné objekty. Ako ďalšie obaľovacie tvary sa využívajú rôzne sférické telesá alebo dvojice kvádrov. Dôležitým faktorom je celkový objem a počet obálok, ktoré by mali byť minimálne.

Použitie obálok pre jednotlivé trojuholníky však nie je výhodné a používa sa hierarchia obálok (*Bounding Volume Hierarchies*). Ide o stromovú štruktúru, ktorá má najčastejšie tvar klasického binárneho stromu. V každom uzle stromu je uložená obálka obklopujúca všetky obálky objektov svojich potomkov. Obálky potomkom sa môžu navzájom prelínať a sú vždy celé obsiahnuté v obálke svojho rodičovského uzla. Samotné objekty sú uložené v listoch stromu. Pre zostavenie stromu možno použiť metódu zhora nadol, ktorá vytvorí obálku celej scény a postupne ju delí na menšie a menšie časti. Metóda zdola nahor najskôr vytvorí listy stromu a postupne ich zapuzdruje do obálok vyššej úrovne a vytvára rodičovské uzly. Pri prechádzaní hierarchie obálok testujeme najskôr priesečník lúča s obálkami a až potom so samotnými objektmi. Základný postup tvorby a scénu s odpovedajúcim stromom znázorňuje obrázok 5.3.



Obrázok 5.3: Princíp vytvárania hierarchie obálok

5.4 BSP

BSP (*Binary Space Partitioning*) je hierarchická štruktúra, ktorá funguje na princípe binárneho delenia priestoru pomocou rôznych hyperplôch. Hlavnou vlastnosťou je spojitosť s binárnym stromom. Každému uzlu je priradená deliaca plocha, ktorá rozdeľuje priestor na dve časti (podpriestory) odpovedajúce synovským uzlom. Koncové uzly (listy stromu) obsahujú zoznam objektov, ktoré sa nachádzajú v odpovedajúcich častiach priestoru. Postupne sa delí priestor až do určitej hĺbky stromu, alebo keď počet objektov prislúchajúcich danému uzlu klesne pod určitú hranicu. Deliace plochy môžu byť zvolené podľa rôznych kritérií, môžu byť vybraté náhodne alebo analyticky. Ideálnym stromom je vyvážený strom, v ktorom je počet objektov ľavej a pravej vetvy rovnaký. BSP stromy majú využitie pri renderovaní rozsiahlych scén s nerovnomerným rozložením objektov. Ich použitie v raytracingu je popísané v [13].

Špeciálnym prípadom BSP stromu je osovo súmerný BSP, ktorý delí priestor plochou kolmou na jednu zo súradných ôs, čo umožňuje výrazne zjednodušiť testovanie priesečníka s deliacou plochou. Podľa V. Havrana [12] je použitie práve osovo súmerného BSP pri sledovaní lúča výhodnejšie, ako použitie klasického BSP.

5.5 KD tree

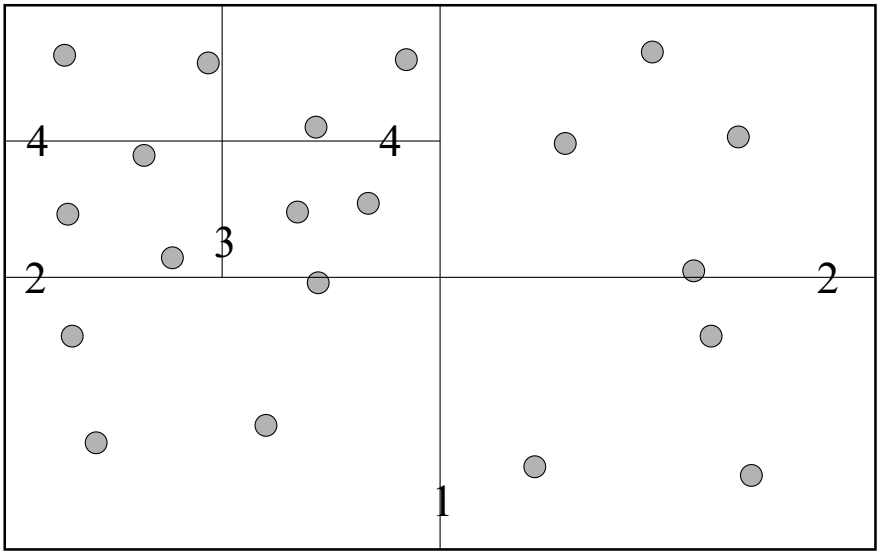
Štruktúra KD stromu (*KD tree*) je veľmi podobná štruktúre BSP, pričom hlavný rozdiel spočíva v umiestnení deliacej plochy, ktorá je vždy kolmá na niektorú osu súradného systému. Podobne ako u BSP môžu byť tieto plochy vložené ľubovoľne pozdĺž danej osy. KD stromy sú v podstate špeciálnym prípadom BSP stromov a sú totožné s osovo súmernými BSP.

Dôležitým faktorom pri stavbe KD stromu je správny výber deliacej roviny. Z práce [12] je zrejmé, že umiestňovanie deliacich plôch má veľký vplyv na výkon samotného algoritmu sledovania lúča, predovšetkým v scénach s nerovnomerným rozložením objektov. V súčasnosti existuje niekoľko metód pre výber deliacej roviny vedúcich k optimálnemu delenie priestoru vzhľadom na geometriu scény. Okrem dvoch klasických metód, priestorového a objemového mediánu, existuje aj výber deliacej plochy na základe cenového modelu,

známeho pod skratkou SAH (*Surface Area Heuristic*).

Priestorový medián

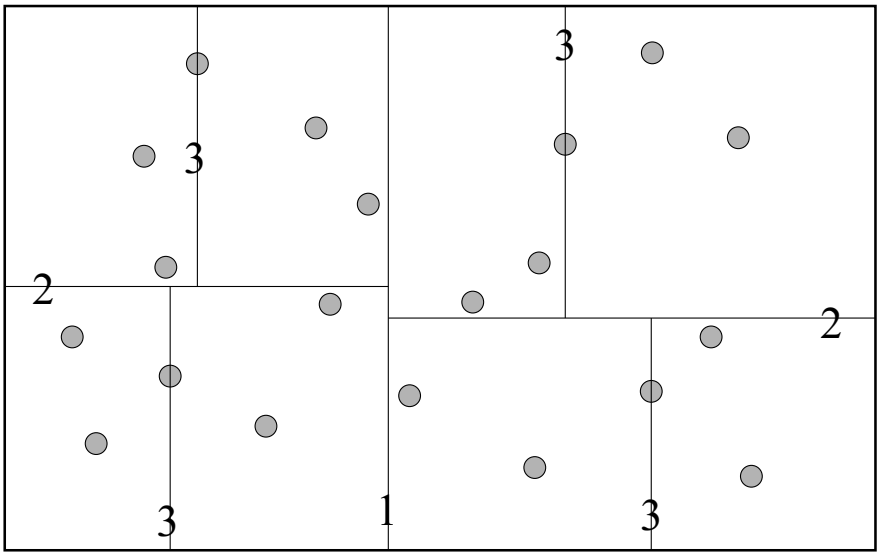
Základná metóda pre výber deliacej plochy. Rovina je umiestnená tak, aby delila priestor na dve rovnako veľké oblasti. Výber deliacej osy prebieha cyklicky alebo podľa rozmerov, keď sa delí priestor podľa najdlhšej strany. Na obrázku 5.4 je možné vidieť, ako takýto algoritmus funguje v 2D priestore. Z hľadiska konštrukcie je priestorový medián najrýchlejšou metódou. Je vhodný predovšetkým pre scény s rovnomerným rozložením objektov.



Obrázok 5.4: Metóda pre umiestnenie deliacej plochy pomocou priestorového mediánu

Objemový medián

Metóda je veľmi podobná priestorovému mediánu. Rovina rozdeľuje priestor tak, aby oblasti na oboch stranách obsahovali rovnaký počet objektov. Môžu nastať prípady, keď sú objekty priradené obom vzniknutým bunkám. Bod, ktorým bude rovina prechádzať, je zvolený ako medián bodov v danom priestore zoradených podľa súradníc v závislosti na zvolenej ose. Pre výpočet polohy je potrebné poznať stredy všetkých objektov v scéne, alebo v prípade scény zloženej z trojuholníkov iba súradnicu jedného z vrcholov. Výber deliacej osy je rovnaký ako v prípade priestorového mediánu. Na obrázku 5.5 je ilustrovaný postup delenia touto metódou.



Obrázok 5.5: Delenie priestoru pomocou objemového mediánu

Cenový model SAH

V súčasnosti najlepšou a najpoužívanejšou metódou pre výber deliacich plôch. Bola vyvinutá v roku 1989 autormi McDonald a Booth [16]. Je založená na možnosti odhadnúť dopredu cenu prechádzania lúča stromom v závislosti na pravdepodobnosti interakcie lúča s jednotlivými objektmi. Pre konvexné obálky môžeme definovať pravdepodobnosť zásahu bunky B, ktorá úplne leží v bunke A, ako:

$$p_{B|A} = \frac{SA(B)}{SA(A)} = \frac{B_w B_h + B_h B_d + B_d B_w}{A_w A_h + A_h A_d + A_d A_w}, \tag{5.1}$$

kde A_w, A_h, A_d a B_w, B_h, B_d sú jednotlivé rozmery buniek A a B. Na základe tejto pravdepodobnosti môžeme odhadnúť cenu priechodu lúča $C(p_u)$ v danom kroku delenia pre deliacu rovinu p_u ako:

$$C(p_u) = K_t + p_{V_L|V} C(V_L) + p_{V_R|V} C(V_R), \tag{5.2}$$

kde V_L a V_R sú potomkovia bunky V rozdelenej deliacou rovinou p_u. Pomocou rovnice 5.2 môžeme stanoviť cenu prechádzania pre každý strom. Množstvo rôznych stromov však rastie extrémne rýchlo s veľkosťou scény a nájsť strom so skutočne najmenšou cenou je takmer nemožné. Preto sa pre výpočet používa lokálna hladová aproximácia (*greedy algoritmus*). Predpokladá sa, že pri výpočte ceny priechodu pre konkrétnu deliacu rovinu sú vzniknutý potomkovia považovaný za listy stromu a ďalej sa už nedelia. Cenu ďalšieho priechodu v danom kroku môžeme zapísať ako:

$$C(p_u) = K_t + K_i(p_{V_L|V} N_L + p_{V_R|V} N_R), \tag{5.3}$$

$$C(p_u) = K_t + K_i(\frac{SA(V_L)}{SA(V)} N_L + \frac{SA(V_R)}{SA(V)} N_R), \tag{5.4}$$

kde parametre $K_i N_L$ a $K_i N_R$ vyjadrujú odhadnutú cenu ľavého a pravého podstromu $C(V_L)$ a $C(V_R)$. N_L a N_R predstavujú počet objektov v jednotlivých podstromoch. K_i vyjadruje cenu priesečníka lúča s objektom a K_t je cena jedného kroku priechodu scénou.

Aj keď výpočet ceny stromu neodpovedá presne skutočným hodnotám, metóda dosahuje veľmi dobrých výsledkov. V súčasnosti nie je známa iná metóda, ktorá by poskytovala na obecných scénach lepšie výsledky.

Pri prechádzaní KD stromu sa najskôr testuje interakcia lúča s potomkami uzla a až následne sa pokračuje v synovských uzloch. Prechádzanie stromu pokračuje až k samotným listom stromu, kde sa testujú priesečníky lúča s jednotlivými objektmi. Pre implementáciu môže byť použitá rekurzia alebo zásobník.

Výhoda KD stromu spočíva v jeho schopnosti sa prispôsobiť priestoru ľubovoľnej dimenzie, čo má uplatnenie v mnohých aplikáciách. Taktiež je schopný modelovať aj iné spomínané štruktúry priestorového delenia, akými sú napr. octree alebo quadtree v 2D priestore, uniformné mriežky, atď. Jeho využitie je obzvlášť vhodné pri metóde sledovania lúča, ale aj v ďalších pokročilých metódach prechodu scénou.

Kapitola 6

Implementácia

Implementovaný raytracer je vhodný pre zobrazenie scény zloženej z polygonálnych (trojuholníkových) objektov. Samotný algoritmus sledovania lúča a výpočet osvetlenia je akcelerovaný na GPU pomocou architektúry CUDA. Podrobný popis aplikácie a riešenia samotných problémov sú vysvetlené v jednotlivých častiach kapitoly.

6.1 Základné vlastnosti aplikácie

Medzi základné prvky (vlastnosti), ktoré aplikácia obsahuje, patria:

- možnosť nastavenia stupňa (hĺbky zanorenia) raytracingu,
- vytváranie dynamických scén,
- podpora odrazených a lomených lúčov (zrkadlové povrchy a priehľadné materiály),
- výpočet osvetlenia pomocou Phongovho osvetľovacieho modelu,
- podpora niekoľkých bodových zdrojov svetla,
- akcelerácia s využitím KD stromu,
- výpočet kaustiky pomocou fotónových máp,
- načítanie trojuholníkových modelov zo súboru s danou syntaxou,
- 4-bodový antialiasing.

6.2 Použité technológie

Pre vytvorenie aplikácie boli použité nasledujúce technológie, knižnice a rozhrania:

- aplikácia je vytvorená v programovacom jazyku C++,
- pre prácu s grafikou a oknami je použitá knižnica OpenGL a jej nadstavba GLUT,
- pre jednoduché nastavenie rôznych parametrov v scéne je použitá GUI knižnica AntTweakBar,
- samotný raytracer je napísaný pomocou rozhrania CUDA,
- na zobrazenie dát je použitý PBO (*Pixel Buffer Object).*

6.3 Vytvorenie a načítanie scény

Dôležitým krokom pre vykreslenie obrazu je definovanie scény, ktorá pozostáva z rôznych zložiek. Jedinou povinnou položkou ja kamera, bez ktorej nie je možné samotné zobrazenie. Medzi ďalšie prvky patria bodové zdroje svetla a polygonálne modely. Vytvorenie scény sa realizuje vo funkcii `CUTBoolean CreateScene(void)` v súbore `main.h`.

Kameru možno definovať pomocou príkazu `Camera* názov = new Camera()`. Pozíciu a parametre môžeme nastaviť podľa rôznych kritérií, ako je priame zadanie súradníc, definovanie smeru a vzdialenosti kamery od stredu zobrazenia, atď. K tomuto účelu boli vytvorené metódy `void LookAt(parametre)`, `void LookAt_2(parametre)` a `void LookAt_3 (parametre)` v triede `Camera.h`. Samotné pridanie kamery do scény sa realizuje príkazom `scene->SetCamera(názov)`.

Druhým dôležitým objektom v scéne je bodový zdroj svetla. Ako už bolo spomínané, scéna môže obsahovať ľubovoľný počet týchto objektov. Jednotlivé svetelné zdroje definujeme ako `Light* názov = new Light(parametre)`, kde *parametre* postupne určujú pozíciu zdroja a jednotlivé farebné zložky svetla (ambientú, difúznu a spekulárnu). Pridanie bodového zdroja do scény je možné pomocou zápisu `scene->AddLight(názov)`.

Posledným typom objektov sú polygonálne modely, ktoré môžeme definovať priamo v zdrojovom kóde, čo je však komplikované. Oveľa jednoduchší spôsob je načítanie jednotlivých modelov priamo z externého súboru pomocou metódy `názov->CreateFromFile (názov_súboru, orientácia_vrcholov)`. Syntax súboru pre načítanie modelov je podrobne popísaná v prílohe B. Pred samotným načítaním je potrebné definovať odkaz na tieto objekty pomocou príkazu `Object* názov = new Object(parametre)`. Pridanie modelu do scény je podobné ako v predchádzajúcich prípadoch, pomocou `scene->AddObject (názov)`. Každý model obsahuje zoznam trojuholníkov, vrcholov a normálových vektorov.

Taktiež je možné definovať farbu pozadia pomocou premennej `background_color` a hĺbku raytracingu pomocou `ray_depth`. Pri dynamických scénach je možné niektoré z vyššie spomínaných parametrov meniť priamo počas spusteného renderovania.

6.4 Interná reprezentácia scény

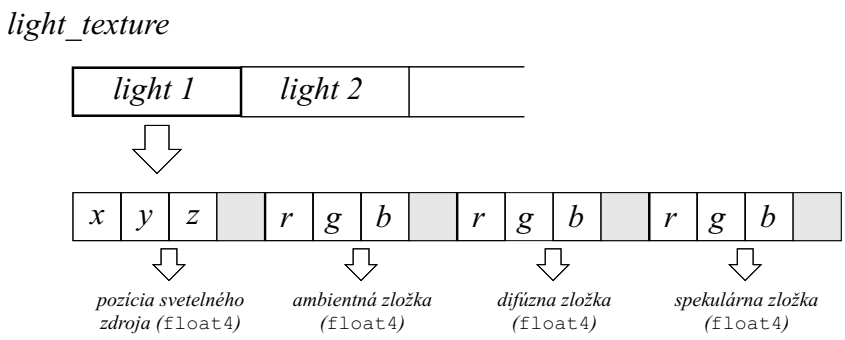
Po načítaní všetkých modelov a svetelných zdrojov sú dáta odoslané do globálnej pamäte GPU pomocou klasickej funkcie `cudaMemcpy(parametre)`. Následne sú uložené do textúrovej pamäte. Textúrová pamäť je na rozdiel od globálnej pamäte kešovaná, čo v prípade paralelného spracovania umožňuje jednotlivým vláknam rýchlejší prístup k dátam. Lúče prechádzajúce susednými pixelmi často prechádzajú aj tými istými objektmi (trojuholníkmi) a vlákna pristupujú k rovnakým informáciám.

Jednotlivé dáta sú rozdelené do viacerých textúr. Štruktúra uloženia scény je znázornená na obrázkoch 6.1 až 6.5. Každý svetelný zdroj je vyjadrený pomocou štyroch hodnôt typu `float4`. Podobne je reprezentovaný materiál objektu. Trojuholníky sú uložené ako trojica vrcholov a ku každému vrcholu je definovaný normálový vektor. Takéto uloženie trojuholníkov je však nevýhodné z hľadiska redundancie informácií. Jednotlivé trojuholníky môžu obsahovať rovnaké vrcholy, čo spôsobuje uloženie kópie a zvyšuje sa pamäťová náročnosť. Na druhej strane by však pri nezávislom uložení vrcholov museli byť pre každý trojuholník definované odkazy na tieto body. To by viedlo k vyššej časovej náročnosti, keďže pre získanie informácií o trojuholníku by sa museli načítať najskôr indexy jeho vrcholov. Mnou zvolená štruktúra je tiež vhodná pre modelovanie ostrých hrán pri Phongovom tieňovaní. Každý trojuholník má zadefinované vlastné normálové vektory vo vrcholoch, čo umožňuje

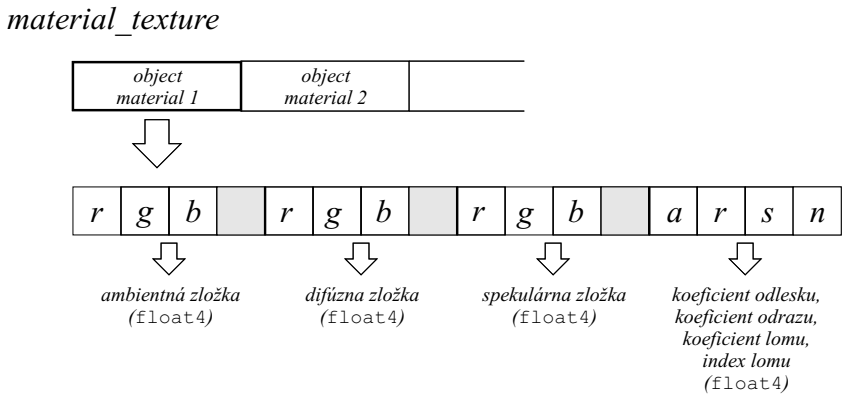
ostré prechody medzi susednými polygónmi. Keďže sú všetky trojuholníky uložené v jednej textúre, je potrebné si pamätať index prvého a posledného trojuholníka pre každý model. Posledná textúra obsahuje informácie o obalových telesách, ktorá nie je však pri implementácii KD stromu použitá, pretože potrebné informácie sú obsiahnuté priamo v akceleračnej štruktúre.

Pre rýchlejšie načítanie dát sú mnohé uložené hodnoty typu `float4`, pretože prístup k takejto hodnote je rýchlejší, ako prístup k štyrom hodnotám typu `float`. Použitie tohto dátového typu však prináša nevýhodu vzniku tzv. prázdnych miest (na obrázkoch 6.1 až 6.5 označené sivou farbou), ktoré zaberajú zbytočne miesto v pamäti.

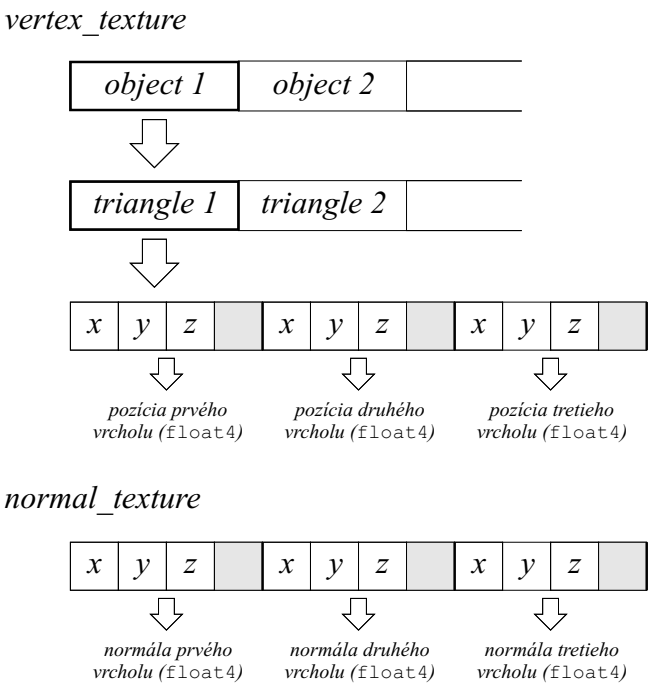
Všetky ostatné potrebné dáta, ako sú napr. parametre kamery, počet svetelných zdrojov, počet objektov, farba pozadia, veľkosť renderovacieho okna, hĺbka zanorenia raytracingu a ostatné sú predávané GPU ako parametre funkcie `__global__ void Render(parametre)`.



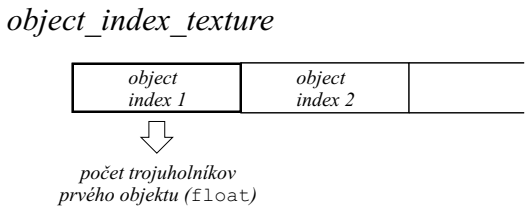
Obrázok 6.1: Štruktúra uloženia svetelených zdrojov na GPU



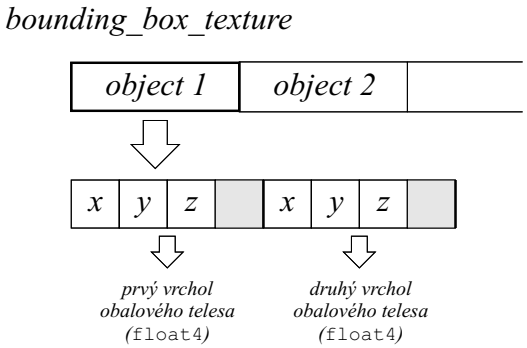
Obrázok 6.2: Štruktúra uloženia materiálov na GPU



Obrázok 6.3: Štruktúra uloženia trojuholníkov na GPU



Obrázok 6.4: Štruktúra uloženia indexov na GPU



Obrázok 6.5: Štruktúra uloženia obalových telies na GPU

6.5 Interná reprezentácia kamery

Pre výpočet primárnych lúčov je potrebné poznať pozíciu kamery a pozíciu stredu pixela, cez ktorý lúč prechádza. Umiestnenie kamery je dané jej definíciou. Pozíciu ľubovoľného pixela možno ľahko vyjadriť pomocou troch parametrov $\vec{a}$, $\vec{b}$ a $\vec{c}$. Parameter $\vec{c}$ reprezentuje počiatok projekčnej roviny (pozíciu pixela v ľavom dolnom rohu), ďalšie dva parametre $\vec{a}$ a $\vec{b}$ predstavujú vektory určujúce smer a veľkosť strán zobrazovacej roviny. Tieto parametre je možné vypočítať pomocou goniometrických funkcií a základných vlastností kamery, akými sú smer snímania, zorné pole (*field of view*) a rozmery vykresľovaného okna (pozri obrázok 6.6).

Pre pozíciu $\vec{p}$ stredu pixela platí:

$$\vec{p} = \vec{c} + x_f\vec{a} + y_f\vec{b} = \vec{c} + \frac{x - 0.5}{\text{window width}}\vec{a} + \frac{y - 05}{\text{window height}}\vec{b}, \tag{6.1}$$

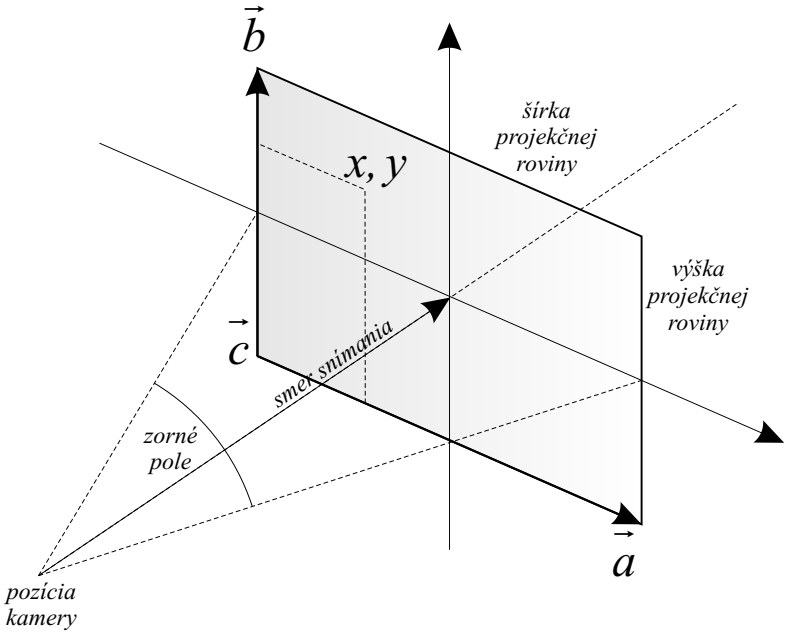
kde:

$\vec{p}$ – pozícia stredu pixela (v priestore),

x, y – pozícia pixela v rámci projekčnej roviny ($x \in \{1, ..., \text{window width}\}$, $y \in \{1, ..., \text{window height}\}$),

window width, window height – veľkosť zobrazovacej roviny (v pixeloch),

$\vec{a}$, $\vec{b}$, $\vec{c}$ – parametre projekčnej roviny.



Obrázok 6.6: Parametre projekčnej roviny

6.6 Priesečník lúča s trojuholníkom

Pre výpočet priesečníka bola použitá metóda *Möller-Trumbore* [17]. Hlavná výhoda tejto metódy spočíva v malých nárokoch na pamäťový priestor (nie je potrebné počítať rovnicu roviny a tým pádom ju ani ukladať).

Lúč R s počiatkom v bode O a normalizovaným smerovým vektorom D môžeme definovať rovnicou:

$$R(t) = O + tD. \tag{6.2}$$

Trojuholník T definovaný troma vrcholmi V_0, V_1 a V_2 môžeme taktiež popísať rovnicou:

$$T(u, v) = (1 - u - v)V_0 + uV_1 + vV_2, \tag{6.3}$$

kde (u, v) sú barycentrické súradnice splňujúce podmienky $u \geq 0$, $v \geq 0$ a $u + v \leq 1$. Parametre (u, v) môžu byť použité pre mapovanie textúry, interpoláciu normál alebo farby.

Výpočet priesečníka lúča R a trojuholníka T je ekvivalentný rovnici

$$R(t) = T(u, v), \tag{6.4}$$

ktorú môžeme rozpísať na:

$$O + tD = (1 - u - v)V_0 + uV_1 + vV_2. \tag{6.5}$$

Rovnicu 6.5 môžeme upraviť na tvar:

$$[-D, V_1 - V_0, V_2 - V_0] \begin{bmatrix} t \\ u \\ v \end{bmatrix} = O - V_0. \tag{6.6}$$

Z geometrického hľadiska úprava spočíva v posunutí vrcholu trojuholníka V_0 do počiatku súradnej sústavy a transformovanie trojuholníka T na jednotkový trojuholník so smerom lúča R rovnobežným s osou x. Z rovnice 6.6 vyplýva, že súradnice (u, v) a vzdialenosť t (od počiatku lúča k bodu priesečníka) môžu byť vypočítané pomocou sústavy lineárnych rovníc. Na riešenie tejto rovnice môžeme použiť *Cramerovo pravidlo*:

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{\begin{vmatrix} -D & E_1 & E_2 \end{vmatrix}} \begin{bmatrix} \begin{vmatrix} T & E_1 & E_2 \end{vmatrix} \\ \begin{vmatrix} -D & T & E_2 \end{vmatrix} \\ \begin{vmatrix} -D & E_1 & T \end{vmatrix} \end{bmatrix}. \tag{6.7}$$

kde $E_1 = V_1 - V_0$, $E_2 = V_2 - V_0$ a $T = O - V_0$. Z lineárnej algebry však vieme, že $|A, B, C| = -(A \times C) \cdot B = -(C \times B) \cdot A$. Rovnicu 6.7 môžeme tak prepísať do konečnej podoby

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{(D \times E_2) \cdot E_1} \begin{bmatrix} (T \times E_1) \cdot E_2 \\ (D \times E_2) \cdot T \\ (T \times E_1) \cdot D \end{bmatrix} = \frac{1}{P \cdot E_1} \begin{bmatrix} Q \cdot E_2 \\ P \cdot T \\ Q \cdot D \end{bmatrix}. \tag{6.8}$$

kde $P = (D \times E_2)$ a $Q = (T \times E_1)$. Táto rovnica bola použitá pri implementácii riešenia.

6.7 Algoritmus pre výpočet raytracingu

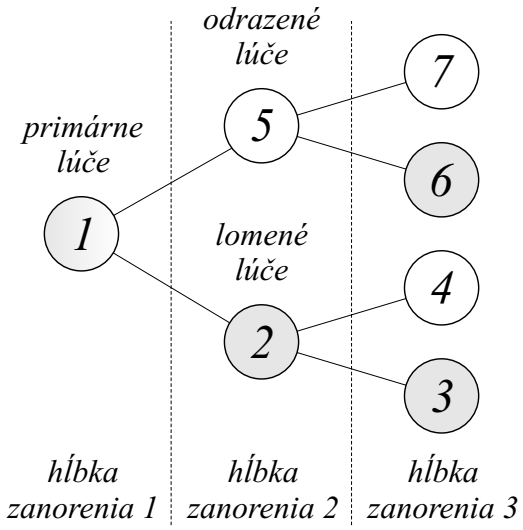
Pri implementácii raytracingu pomocou architektúry CUDA bolo potrebné upraviť algoritmus do nerekurzívnej formy, pretože CUDA nepodporuje rekurzívne volanie funkcií. Úprava spočíva vo vytvorení dvoch polí `currRays` a `newRays`, v ktorých sa postupne ukladajú primárne a sekundárne lúče. Princíp algoritmu reprezentuje pseudokód 6.1:

1. Inicializácia algoritmu spočíva vo vytvorení pola `currRays` obsahujúci primárny lúč `R` a prázdneho pola `newRays`. Výsledná farba `I` je nastavaná na čiernu farbu.
2. Pokiaľ pole `currRays` obsahuje aspoň jeden lúč a súčasne nebola prekročená hĺbka zanorenia `Z`:
   (a) Pre každý lúč `R` pola `currRays`:
      i. Zavolá sa nerekurzívna funkcia $\texttt{I}_p$ `= Raytrace(R,` $\texttt{R}_r$`,` R_s`)`, ktorá vracia farbu $\texttt{I}_p$ na základe osvetľovacieho modelu.
      ii. K výslednej farbe bodu (pixelu) `I` sa pripočíta farba $\texttt{I}_p$ (`I+=`$\texttt{I}_p$).
      iii. Sekundárne lúče $\texttt{R}_r$ a $\texttt{R}_s$ sú uložené do pola `newRays` (lúče boli získané pri volaní funkcie `Raytrace(R,` $\texttt{R}_r$`,` $\texttt{R}_s$`)`.
   (b) Presunutie všetkých lúčov z pola `newRays` do pola `currRays` (sekundárne lúče sa stanú primárnymi).
   (c) Zvýšenie hĺbky zanorenia (`Z++`).

Algoritmus 6.1: Nerekurzívna funkcia sledovania lúča

Nevýhoda algoritmu spočíva v neefektívnom sledovaní sekundárnych lúčov. Každý primárny lúč môže po dopade na povrch objektu vytvoriť odrazený a/alebo lomený lúč, alebo v prípade čisto difúzneho povrchu ukončiť svoju cestu. Takto môžu primárne lúče počas priechodu scénou v závislosti na hĺbke zanorenia vygenerovať rôzny počet sekundárnych lúčov. To má za následok nevyužitie úplného výpočtového výkonu vláken vo warpe, pretože jednotlivé vlákna sú spracovávané rovnako dlho, ako vlákno reprezentujúce primárny lúč s najdlhším prechodom scénou.

Pre čiastočné odstránenie tohto nedostatku bol vytvorený algoritmus, v ktorom by jednotlivé vlákna spracovávali len primárne lúče a odrazené a lomené lúče by ukladali do globálnej pamäte. Následne by boli tieto lúče opätovne spracované rovnakým spôsobom novou skupinou vláken. Pre minimalizovanie využitia pamäte je vždy spracovaná „aktuálna“ skupina sekundárnych lúčov, ktorá vznikla pri poslednom zanorení. Princíp je podobný zásobníkovej štruktúre, kde prvkami zásobníka sú množiny lúčov. Pri prvom priechode sú spracované všetky primárne lúče, pričom vzniknú dve skupiny sekundárnych lúčov (odrazené a lomené). Ako prvé sa vždy spracujú odrazené lúče, ktoré vznikli pri poslednom zanorení. Tie vždy vygenerujú novú skupinu odrazených a lomených lúčov. V prípade, ak bola dosiahnutá maximálna hĺbka zanorenia alebo neboli vygenerované žiadne sekundárne lúče, je spracovaná posledná uložená skupina (ktorá vznikla pri najväčšom zanorení). Postupne algoritmus spracuje všetky sekundárne lúče vygenerované pôvodnými primárnymi. Aj napriek optimalizácii spracovania je celková pamäť pre ukladanie sekundárnych lúčov obrovská, pretože pre každý stupeň zanorenia a pre každý pixel je potrebné si ukladať jeden sekundárny lúč spolu s jeho parametrami (druhý sekundárny lúč je automaticky spracovaný hneď v ďalšom kroku výpočtu). Princíp algoritmu je znázornený na obrázku 6.7, kde sú skupiny lúčov očíslované v poradí, v akom by boli spracované.



Obrázok 6.7: Princíp spracovania sekundárnych lúčov na GPU

Výhoda algoritmu sa predpokladá v scénach s obrovským počtom objektov rôznych povrchov (materiálov), kde majú primárne lúče rôzne priechody a generujú rozdielny počet sekundárnych lúčov.

6.8 PBO

Pixel buffer object (PBO) reprezentuje dáta uložené vo vnútornej pamäti grafickej karty. Architektúra CUDA dovoľuje priamy zápis do PBO rozhrania OpenGL, čo umožňuje rýchle zobrazenie výsledku bez potreby kopírovania dát do hlavnej pamäte počítača a naspäť na GPU. Výrazné urýchlenie pritom nastáva pri vykresľovaní dynamických scén v reálnom čase, keď je potrebné niekoľkokrát v krátkom intervale zobraziť výsledok renderovania.

6.9 Akceleračné štruktúry

Tvoria podstatnú časť pri urýchlení výpočtu priesečníka s trojuholníkmi a samotnom vykreslení obrazu. Moja implementácia stavia na vhodnej kombinácii obalových telies a KD stromu. Základnou podstatou sú obalové telesá, ktoré sú vytvorené nad jednotlivými modelmi v scéne. Každá obálka zapuzdruje práve jeden objekt scény. Obalové telesá sú v tvare kvádra súmerného so súradnými osami, čo umožňuje použiť jednoduchý a rýchly algoritmus na určenie priesečníka lúča s týmto telesom.

Ďalším krokom je delenie obalového telesa pomocou KD stromu na menšie oblasti, ktoré majú taktiež tvar kvádra. Pre výber deliacich rovín bolo použité priestorové delenie. Každý uzol stromu zastupuje určitú časť obalového telesa a listy obsahujú samotné trojuholníky nachádzajúce sa v týchto oblastiach. Koreňový uzol reprezentuje obálku nad celým modelom. Výsledný strom tak predstavuje v konečnom štádiu určitú hierarchiu obálok.

Pri stavbe stromu boli použité obálky trojuholníkov. Pri určení príslušnosti trojuholníka k danej oblasti sa určuje príslušnosť obalového telesa tohto polygónu. V prípade, že

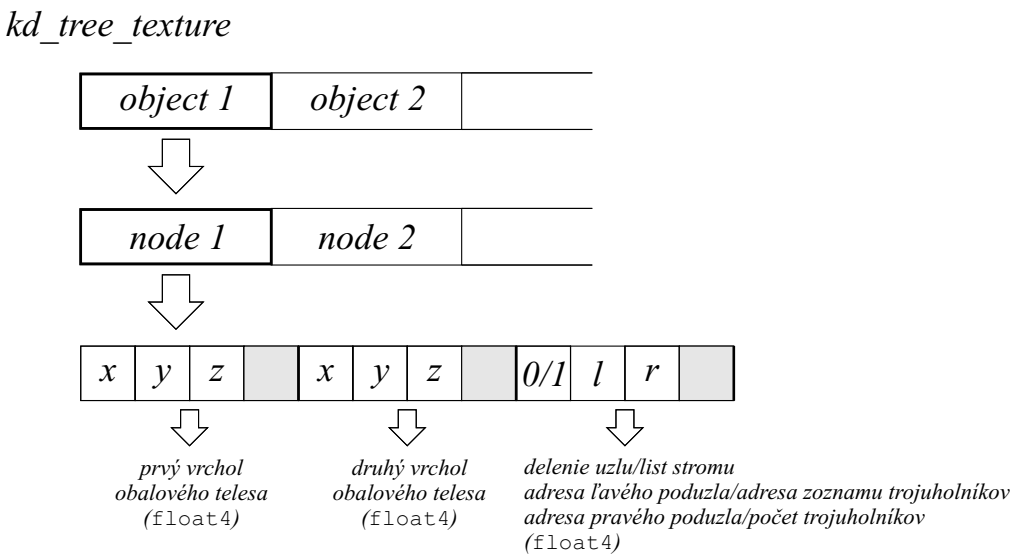
deliaca rovina delí trojuholník na dve časti (patrí do obidvoch synovských uzlov), tak je vhodne prepočítaná jeho obálka, aby obsahovala len tú časť trojuholníka patriacu do danej oblasti. Tvorba celého stromu je implementovaná na CPU, existujú však práce venujúce sa urýchleniu stavby tejto štruktúry pomocou GPU [8], [26].

Výhoda tejto reprezentácie objektov spočíva v dynamických scénach. Pri pohybe jednotlivých modelov nedochádza k zmene obalového telesa a ani prepočítanie KD stromu, ale je odoslaná informácia o vektore pohybu do grafickej karty. Pri výpočte priesečníku lúča s týmto modelom dochádza k posunu lúča v opačnom smere, pričom samotný objekt v skutočnosti nemení svoju polohu. Po nájdení priesečníka je potrebné jeho pozíciu a taktiež smer jednotlivých sekundárnych lúčov vhodne prepočítať. Pri výpočte tieňových lúčov je taktiež potrebné vždy brať do úvahy posun tohto lúča v závislosti na teoretickom posune ostatných objektoch scény.

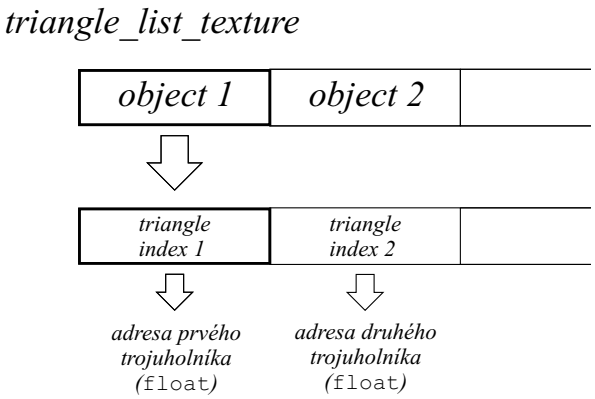
Analogicky môžeme postupovať pri rotácii jednotlivých objektov scény. Nedochádza k skutočnej rotácii týchto objektov, a teda nie je potrebné prepočítavať ich obálku a ani KD strom. Taktiež pri hľadaní priesečníka lúča s modelom je vhodne zmenená jeho orientácia a následne upravená pozícia priesečníka a smer sekundárnych lúčov.

Pri zmene vnútornej štruktúry objektu (zmena trojuholníkov) stačí prepočítať KD strom len pre aktuálny model a dáta znova odoslať na GPU. Pri lokálnej zmene by stačilo prestavať len podstrom, v ktorom došlo zmene.

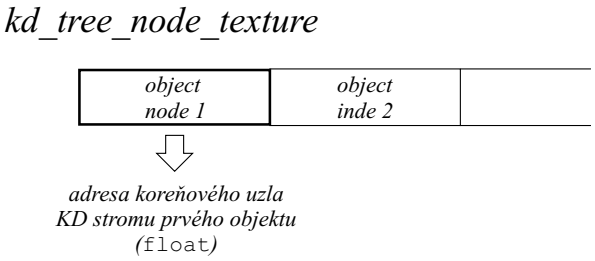
Celé delenie scény je na grafickej karte uložená vo forme textúr, ako je znázornené na nasledujúcich obrázkoch 6.8, 6.9 a 6.10. Prvá textúra obsahuje KD stromy pre jednotlivé objekty usporiadané za sebou. Každý uzol obsahuje informácie o svojom obalovom telese a odkazy na svoje synovské uzle. V prípade koncového uzla (listu) obsahuje index prvého a posledného trojuholníka, ktorý sa nachádza v jeho časti. Ďalšie textúry obsahujú indexy na koreňové uzly KD stromov a zoznam trojuholníkov pre jednotlivé listy.



Obrázok 6.8: Uloženie KD stomov na GPU



Obrázok 6.9: Zoznam trojuholníkov

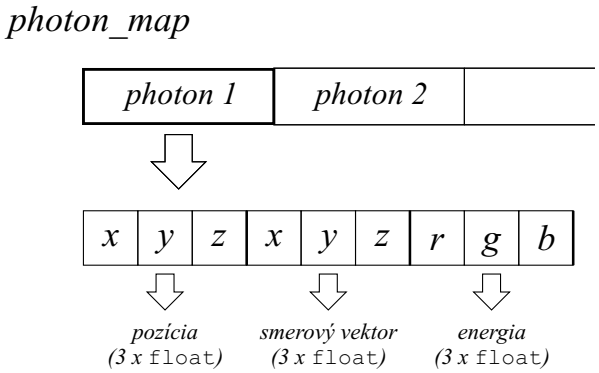


Obrázok 6.10: Adresy koreňových uzlov

6.10 Fotónová mapa

Pre realistickejšie renderovanie objektov bola implementovaná fotónová mapa reprezentujúca kaustiky. Samotné sledovanie a výpočet fotónov je realizované na grafickej karte. Prenos dát na GPU nie je nutný, pretože všetky potrebné informácie (ako sú trojuholníky, normálové vektory, svetelné zdroje, atď.) sú uložené v textúrovej pamäti grafickej karty.

Samotné zobrazenie fotónov je súčasťou algoritmu pre výpočet osvetlenia, a preto je taktiež implementované na grafickom akcelerátori. Pre prechádzanie fotónovej mapy však nie je použitá žiadna akceleračná štruktúra, čo má výrazný vplyv na samotnú rýchlosť zobrazenia scény. Fotónová mapa, podobne ako všetky ostatné dáta, je uložená do textúrovej pamäte. Jej štruktúra je znázornená na obrázku 6.11.



Obrázok 6.11: Štruktúra fotónovej mapy

Kapitola 7

Výsledky a testovanie

Testovanie aplikácie sa realizovalo na niekoľkých scénach líšiacich sa počtom a zložitosťou objektov. Niektoré modely boli prevzaté z depozitára Stanfordskej univerzity. Boli vytvorené skenovaním skutočných trojrozmerných objektov a následne prevedené do polygonálnej reprezentácie. Konkrétne som použil modely Stanford Bunny a Stanford Dragon. Medzi ďalšie použité objekty patril model átria paláca Sponza, ktorého autorom je M. Dabrović. Všetky ostané objekty nachádzajúce sa v scénach boli vlastnoručne vytvorené pomocou rozličných programov pre tvorbu trojrozmernej grafiky.

Prvá séria testov bolo zameraná na meranie výkonnosti akceleračnej štruktúry. Merania boli uskutočnené nad rôznymi modelmi, ktoré sa navzájom odlišovali tvarom a počtom trojuholníkov. Scény boli taktiež použité pre určenie závislosti rozlíšenia obrazu na výslednej dobe zobrazenia scény.

Druhá séria testov obsahovala komplexnejšie scény zložené z väčšieho počtu modelov rôzneho materiálu. Jednotlivé scény boli vytvorené predovšetkým pre porovnanie rýchlosti vykreslenia obrazu pri statických a dynamických objektoch. V tejto sérii testov boli taktiež uskutočnené merania potvrdzujúce výhody použitia PBO.

Pre ďalšiu sériu meraní bola vytvorená scéna obsahujúca veľké množstvo lesklých a priehľadných objektov. Scéna bola špeciálne určená pre testovanie dvoch rôznych prístupov spracovania lúčov na GPU.

Ďalšia skupina testov bola realizované nad modelom Sponza Atrium. Tieto merania slúžia predovšetkým pre porovnanie s inými implementovanými raytracermi.

Posledné merania boli zamerané na použitie fotónových máp. Pre testovanie bola použitá scéna obsahujúca malý počet priehľadných objektov.

Výsledným faktorom všetkých meraní bola rýchlosť vytvorenia scény, odoslania dát na grafickú kartu a samotná doba renderovania (zobrazenia). Pri každom testovaní bolo uskutočnených 100 meraní a do tabuľky sa zaznamenala vždy priemerná hodnota. Scény boli renderované pri rozlíšení 640×480 pixelov (pokiaľ nebolo uvedené inak). Pre testovanie bola použitá grafická karta nVidia GeForce GTX460. Podrobný popis testovacej zostavy a konfigurácie grafickej karty sú uvedené v prílohe C.

7.1 Prvé meranie – statické scény

Prvá séria testov bola realizovaná nad tzv. statickými scénami, kde je pevne zvolená pozícia kamery a pozícia svetla a nie je nijako možné meniť žiadne parametre počas spusteného renderovania. Testy boli prevedené nad štyrmi objektmi obsahujúce rôzny počet vrcholov

a trojuholníkov. Medzi testované objekty patria:

- mnohosten,
- obrys tváre,
- Stanford Bunny,
- Stanford Dragon.

Podrobné informácie o týchto scénach a ich vyobrazenie je v prílohe D.

Použitie akceleračných štruktúr

Prvé meranie je zamerané na porovnanie rýchlosti vytvorenia scény (prípadne KD stromu), čas prenosu dát na GPU a čas samotného zobrazenia. Taktiež bolo testované množstvo využitej pamäte grafickej karty pri jednotlivých meraniach. Keďže šlo o porovnanie akceleračnej štruktúry, boli použité len primárne lúče. To nám umožňuje vhodne porovnať jednotlivé merania, pretože počet primárnych a sekundárnych lúčov bol v scénach vždy rovnaký. Ostatné parametre ako je použitie PBO, antialiasing alebo výsledné rozlíšenie obrazu sú pre všetky scény nastavené na rovnaké hodnoty.

Výsledky testovania sú zhrnuté do tabuľky 7.1. Ako je možné vidieť, renderovanie bez použitia KD stromu viedlo k rýchlemu vytvoreniu scény. K výraznému zníženiu doby prenosu dát do globálnej pamäte grafickej karty však nedošlo a urýchlenie predstavuje len niekoľko milisekúnd. Na druhej strane doba zobrazenie objektu pre väčší počet trojuholníkov exponenciálne rástla a pre scény s vysokým počtom polygónov sa pohybovala rádovo v jednotkách sekúnd.

V prípade použitia KD stromu sa celkové vykreslenie scény značne urýchlilo predovšetkým pre scény so zložitými objektmi. Tvorba stromu však priniesla spomalenie pri vytváraní scény. Taktiež narástlo využitie pamäte, hlavne v scénach s veľkým počtom polygónov. Je to dané predovšetkým štruktúrou KD stromu, kde pre zložité objekty veľkosť stromu rastie a pre každý uzol stromu je potrebné si pamätať obalové teleso a odkaz na svojich potomkov. Pre listy stromu je potrebný zoznam trojuholníkov, ktoré sa nachádzajú v danej oblasti priestoru.

Ideálna maximálna hĺbka stromu a minimálny počet trojuholníkov v listoch je pre jednotlivé scény rozdielny. Pre rôzne nastavenia týchto parametrov sú aj rozdielne výsledky merania. Pre dostatočne zložité objekty platí pravidlo, že čím väčšia povolená hĺbka stromu a menší počet trojuholníkov v listoch, tak tým dlhšie trvá vytvorenie daného stromu a zaberá väčšie množstvo pamäte. Na druhej strane však nemusí dochádzať k zvyšovaniu rýchlosti samotného zobrazenia (prechádzania stromom), ale naopak môže dôjsť aj k jeho zníženiu.

Doba vytvorenia KD stromu môže pre zložité scény dosahovať rádovo niekoľko jednotiek až desiatok sekúnd, čo je spôsobené implementáciou na CPU. Ich použitie je vhodné predovšetkým pre dynamické scény, kde k jeho tvorbe dochádza len jeden krát pred samotným zobrazením a následne je pri zmene štruktúry objektu prepočítaná len časť tohto stromu. Pre statické scény závisí jeho použitie na celkovom čase vytvorenia stromu, prenosu dát a doby zobrazenia. V niektorých scénach môže byť tento celkový čas menší, v iných scénach môže byť zasa použitie KD stromu nevýhodné.

Označenie scény	Akceleračná štruktúra (hĺbka stromu/min. počet trojuholníkov)	Vytvorenie scény [ms]	Prenos dát [ms]	Renderovanie [ms]	Využitie pamäte [Mb]
1A	len obalové telesá	2,2	139,1	5,9	47
	10/10	6,3	142,7	4,5	49
	20/20	13,4	141,2	4,3	48
	50/10	26,3	142,2	4,5	49
	50/20	19,6	141,9	4,5	49
	50/50	6,3	141,2	5,0	48
1B	len obalové telesá	564,9	138,6	1736,5	49
	10/10	2241,2	139,6	113,3	50
	20/20	6334,7	143,1	48,1	51
	50/10	12325,2	142,2	48,1	52
	50/20	6579,2	146,2	48,4	53
	50/50	3806,3	139,9	60,5	50
1C	len obalové telesá	783,4	149,9	5361,2	54
	10/10	4341,7	150,5	144,8	54
	20/20	11659,4	150,4	59,3	55
	50/10	20357,0	155,6	55,1	67
	50/20	11674,3	151,9	59,6	58
	50/50	7251,9	151,3	77,0	57
1D	len obalové telesá	842,6	146,2	9298,3	56
	10/10	5894,7	147,9	209,8	57
	20/20	17774,7	159,1	70,3	58
	50/10	36554,1	167,2	66,2	69
	50/20	19531,2	155,6	69,2	62
	50/50	11803,9	147,5	89,6	57

Tabuľka 7.1: Použitie rôznych akceleračných štruktúr

Rozlíšenie obrazu a antialiasing

Cieľom testovania bolo určenie závislosti doby renderovania na rozlíšení obrazu. Snažil som sa porovnať celkové množstvo využitia pamäte a samotný čas zobrazenia scény (bez predspracovania, ako je vytvorenie KD stromu a poslanie dát na GPU) v závislosti na rozlíšení zobrazovacej plochy a použití antialiasingu. Pre jednotlivé objekty boli použité vždy rovnaké parametre KD stromu, ako aj iné parametre scény, aby nedochádzalo k narušeniu merania hodnôt rôznymi faktormi. Pre každú scénu boli merané hodnoty pre štyri rozlíšenia:

- 480 x 360 pixelov (spolu 172 800),
- 640 x 480 pixelov (spolu 307 200),
- 1024 x 768 pixelov (spolu 786 432),
- 1280 x 960 pixelov (spolu 1 228 800),

a jednotlivé merania boli uskutočnené so zapnutým a vypnutým antialiasingom. Výsledky testovania sú uverejnené v tabuľka 7.2.

Označenie scény	Rozlíšenie [px]	Využitie pamäte (bez/s antialiasingom) [Mb]	Renderovanie (bez/s antialiasingom) [ms]
1A	480 x 360	43/51	3,59/9,32
	640 x 480	49/54	4,35/14,61
	1024 x 768	54/62	8,3/34,10
	1280 x 960	69/78	12,49/53,85
1B	480 x 360	49/54	33,42/157,20
	640 x 480	53/60	47,76/232,43
	1024 x 768	62/71	77,40/399,21
	1280 x 960	72/83	106,66/532,14
1C	480 x 360	52/57	40,94/196,31
	640 x 480	58/70	59,82/292,09
	1024 x 768	64/78	108,03/548,75
	1280 x 960	76/84	147,82/745,92
1D	480 x 360	56/73	47,30/232,43
	640 x 480	59/68	69,61/343,36
	1024 x 768	65/83	128,21/653,02
	1280 x 960	80/94	173,34/880,73

Tabuľka 7.2: Rôzne rozlíšenie a použitie antialiasingu

Výsledky merania odpovedajú predpokladu a vo všetkých meraniach rástla doba výpočtu so zvyšujúcim sa rozlíšením. Každým pixelom prechádza jeden primárny lúč a pri zväčšení počtu pixelov logicky dochádza k zvýšeniu počtu lúčov v scéne (či už primárnych, ale aj tieňových a sekundárnych). Pri zvýšení rozlíšenia taktiež dochádza k zväčšeniu množstva použitej pamäte. Je to dané tým, že pre každý pixel je v pamäti GPU vyhradená hodnota typu **int**, ktorá reprezentuje jeho farbu. Táto položka však nie je výrazná a činí rádovo len stovky KB. Výrazný nárast použitej pamäte spôsobil samotný algoritmus prechádzania lúča, ktorý si uchováva pri výpočte rôzne hodnoty potrebné pre spracovanie.

Pri použití antialiasingu dochádza k rapídnemu zväčšeniu doby zobrazenia scény pre akékoľvek rozlíšenie. Je to spôsobené základnou implementáciou antialiasingu, ktorý je štvorbodový a pre každý pixel vysiela štyri primárne lúče. Ide o základný a najjednoduchší typ antialiasingu, ktorý nie je nijak optimalizovaný a nezávisí na geometrii scény. Jeho použitie má výrazný vplyv na výslednú rýchlosť zobrazenia scény, a preto pri ostatných meraniach nie je používaný.

7.2 Druhé meranie – komplexné scény

Pre ďalšie testovanie boli vytvorené tzv. dynamické scény, v ktorým je možné meniť pozíciu kamery a jednotlivých objektov, rôzne parametre svetelných zdrojov, hĺbku zanorenia raytracingu, atď. Poloha kamery mala však veľký vplyv na výsledný čas renderovania, a preto bola pri testovaní nemenná a jej pozícia bola nastavená tak, aby vždy zobrazovala celú scénu s čo najväčším počtom neprekrývajúcich sa objektov.

Jednotlivé scény sú zložené z väčšieho počtu modelov rôznej zložitosti a materiálu, nachádzajú sa tu lesklé a priehľadné objekty, viacero svetelných zdrojov, atď. Vyobrazenie

scén a ich podrobný popis sa nachádza v prílohe E.

Dynamické scény

Testovanie bolo zamerané predovšetkým na scény s pohyblivými svetelnými zdrojmi a jednotlivými objektmi. Z výsledkov testovania (pozri tabuľka 7.3) je viditeľné, že aj napriek plne dynamickej scéne nedochádza k výraznému spomaleniu zobrazenia. Výsledný počet zobrazených snímkov za sekundu je takmer rovnaký ako pre statické aj dynamické scény. Veľký význam na to mala vhodne zvolané štruktúra scény popísaná v časti 6.9, kde pri pohybe objektov nedochádza k prepočítaniu obalových telies a KD stromu, ale len k odoslaniu údajov do GPU.

Označenie scény	Rozlíšenie [px]	Počet obrázkov za sekundu][fps]	
		statická scéna	dynamická scéna
2A	480 x 360	206,08	192,21
	640 x 480	124,20	119,4
	1024 x 768	60,73	57,98
	1280 x 960	41,33	39,17
2B	480 x 360	20,03	19,11
	640 x 480	15,88	15,54
	1024 x 768	8,59	8,32
	1280 x 960	6,35	6,48
2C	480 x 360	88,01	87,49
	640 x 480	60,43	60,02
	1024 x 768	32,02	31,88
	1280 x 960	21,89	21,53
2D	480 x 360	17,72	17,94
	640 x 480	12,38	12,63
	1024 x 768	6,31	6,49
	1280 x 960	4,42	4,48

Tabuľka 7.3: Porovnanie statických a dynamických scén

Použitie PBO

Pri testovaní bola meraná rýchlosť zobrazenia scény so zapnutým a vypnutým PBO. Merania boli uskutočnené pre rôzne rozlíšenia zobrazovacej roviny. PBO umožňuje priame zobrazenie výsledku bez nutnosti jeho kopírovania z grafickej karty na CPU a spätne, čo potvrdili aj namerané výsledky (pozri tabuľku 7.4).

Z výsledkov je viditeľné, že použitie PBO je vhodné predovšetkým v dynamických scénach, v ktorých dochádza k vykresleniu veľkého počtu zobrazení za krátku dobu. Pri každom zobrazení dochádza k čiastočnému urýchleniu a v prípade scén s vysokým počtom zobrazených snímkov za sekundu je celkové urýchlenie značne vysoké.

Označenie scény	Rozlíšenie [px]	Počet obrázkov za sekundu [fps]		Urýchlenie [%]
		bez použita PBO	s použitím PBO	
2A	480 x 360	140,80	206,08	1,46
	640 x 480	93,62	124,20	1,33
	1024 x 768	40,08	60,73	1,51
	1280 x 960	28,06	41,33	1,47
2B	480 x 360	19,46	20,03	1,03
	640 x 480	14,96	15,88	1,06
	1024 x 768	7,90	8,59	1,09
	1280 x 960	5,83	6,35	1,09
2C	480 x 360	72,15	88,01	1,22
	640 x 480	52,25	60,43	1,16
	1024 x 768	26,01	32,02	1,23
	1280 x 960	17,51	21,89	1,25
2D	480 x 360	17,16	17,72	1,03
	640 x 480	11,92	12,38	1,04
	1024 x 768	6,01	6,31	1,05
	1280 x 960	4,21	4,42	1,05

Tabuľka 7.4: Použitie PBO

7.3 Tretie meranie – rôzne prístupy sledovania lúča

Scéna (pozri prílohu F) bola špeciálne vytvorená pre porovnanie rýchlosti vykreslenia pomocou dvoch rôznych prístupov sledovania lúča popísaných v podkapitole 6.7. Prvý prístup reprezentuje klasický nerekurzívny raytracing, kde každé vlákno spracuje celú cestu lúča scénou spolu s odrazenými a lomenými lúčmi. V druhom prípade dochádza k uloženiu odrazených a lomených lúčov do globálnej pamäte a znovu spustenie kernelu pre spracovanie týchto lúčov.

Celkovo bolo uskutočnených niekoľko meraní pre rôzne rozlíšenie výsledného obrazu a rôznu úroveň zanorenia raytracingu. Pri hĺbke raytracingu 1 majú všetky objekty čisto difúzny povrch a nedochádza tak k vytvoreniu sekundárnych lúčov. Jednotlivé výstupy merania sú popísané v tabuľke 7.5.

Výsledky testovania ukázali lepšiu použiteľnosť pomocou prvého prístupu, hoci pri veľkej koherencii lúčov a rôznemu priechodu scénou bolo predpokladané urýchlenie pomocou druhej metódy. Pri použití primárnych lúčov je rýchlosť zobrazenia približne rovnaká pre obe implementácie, ale s rastúcou hĺbou zanorenia dochádza v 2-hej metóde k výraznému spomaleniu. Jej nevýhodou je predovšetkým práca s globálnou pamäťou, ktorá predstavuje časovo náročné operácie. Z hľadiska uloženia veľkého počtu dát však nebolo možné použitie iného typu pamäte.

Rozlíšenie [px]	Hĺbka zanorenia	Počet obrázkov za sekundu [fps]	
		1. metóda	2.metóda
480 x 360	1	41,13	42,07
	5	9,81	3,63
	9	8,74	1,74
640 x 480	1	27,55	28,16
	5	6,46	2,51
	9	5,73	1,29
1024 x 768	1	13,31	13,53
	5	3,27	1,27
	9	2,87	0,75
1280 x 960	1	9,07	9,19
	5	2,62	0,89
	9	2,02	0,58

Tabuľka 7.5: Rôzne prístupy sledovania lúča

7.4 Štvrté meranie – model Sponza Atrium

V tejto sérii testovania bolo merané množstvo zobrazených obrázkov za sekundu pre model Sponza Atrium. Celkovo boli uskutočnené tri meranie pre rôzne rozlíšenie zobrazovacej roviny a parametre kamery. Jednotlivé vyrenderované obrázky a popis scény je uvedený v prílohe G. Výsledky merania sú uvedené v tabuľke 7.6. Rýchlosť zobrazenia scény je pre rôzne nastavenia pozície a smeru snímania kamery odlišná, pohybuje sa však rádovo v jednotkách fps.

Označenie svény	Rozlíšenie [px]	Počet obrázkov za sekundu [fps]
4A	480 x 360	5,288
	640 x 480	3,603
	1024 x 768	1,911
	1280 x 960	1,415
4B	480 x 360	9,762
	640 x 480	6,831
	1024 x 768	3,536
	1280 x 960	2,583
4C	480 x 360	4,834
	640 x 480	3,358
	1024 x 768	1,801
	1280 x 960	1,345

Tabuľka 7.6: Výsledky merania pre model Sponza Atrium

7.5 Piate meranie – fotónové mapy

Testovanie bolo realizované nad scénou popísanou v prílohe H. Pre vytvorenie fotónovej mapy bolo použitých 200 000 fotónov. Samotné vytvorenie mapy trvalo 130,58ms a zobrazenie 81027,91ms. Z nameraných hodnôt je viditeľné rapídne zvýšenie renderovania scény, ktoré je spôsobené samotným prechádzaním fotónovej mapy. Tá nie je nijak optimalizovaná a pre každý priesečník lúča s objektom je prechádzaná celá mapa.

Kapitola 8

Záver

V rámci diplomovej práce boli spracované rôzne kapitoly z oblasti počítačovej grafiky a vizualizácie scény. Predovšetkým to bola zobrazovacia rovnica a rozličné metódy vykreslenia a osvetlenia. Taktiež nechýbajú rôzne akceleračné štruktúry pre urýchlenie algoritmov zobrazujúcich scénu. Ďalej sú tu popísané základné technológie pre implementáciu na grafickej karte, pričom väčšia pozornosť je venovaná architektúre CUDA.

Dôležitá časť práce je venovaná samotnému návrhu a implementácii algoritmu sledovania lúčov, v ktorom sú detailne popísané riešenia jednotlivých problémov. Algoritmus využíva možnosti paralelného spracovania a je implementovaný pomocou architektúry CUDA. Na grafickú kartu sa podarilo preniesť všetky výpočty spojené so zobrazením scény a tým zabrániť zbytočnému prenosu dát medzi hostiteľom a GPU. Pre priechod scénou boli použité dva spôsoby. V prvej metóde išlo o klasický nerekurzívny algoritmus sledovania lúča, kde každé vlákno spracovávalo celú cestu lúča scénou. Druhý spôsob využíval globálnu pamäť pre ukladanie sekundárnych lúčov, ktoré boli neskôr spracované novou skupinou vláken (novým kernelom). Merania ukázali nevýhodu využitia globálnej pamäte v prípade druhej implementácie, čo malo za následok zníženie rýchlosti vykreslenia obrazu.

Pre overenie vlastností výslednej aplikácie bolo uskutočnených niekoľko testov. Jednotlivé merania boli realizované nad rôznymi scénami líšiacich sa počtom a zložitosťou objektov, rôznymi materiálmi, atď. Štruktúra delenia scény a aj samotná implementácia bola zvolená vzhľadom na renderovanie dynamických objektov. Pre každý objekt bolo vytvorené obalové teleso, ktoré bolo delené na menšie časti použitím KD stromu. Merania preukázali výhodu použitia tejto akceleračnej štruktúry, ktorá výrazne urýchlila samotné renderovanie scény. Pri pohybe objektov nie je potrebné prepočítanie obalových telies a ani KD stromu, ale len vhodne upraviť parametre lúčov v scéne. Pre lepšiu výslednú kvalitu zobrazenia boli implementované metódy ako antialiasing a fotónové mapy pre výpočet kaustiky. Tieto metódy nie sú nijak optimalizované a preto výrazne spomaľujú beh celej aplikácie. Výsledný raytracer je schopný zaujímavých výstupov, pričom v dynamických scénach umožňuje užívateľovi meniť smer pohľadu, hĺbku zanorenia sledovania lúčov, pozíciu objektov, atď.

Ukázalo sa, že súčasná generácia grafických akcelerátorov nie je schopná zatiaľ implementovať pokročilé metódy zobrazenia v reálnom čase pre zložitejšie scény. Dochádza však k neustálemu zvyšovaniu výpočtového výkonu a nie je vylúčené, že v budúcnosti budú práve metódy sledovania lúčov základnými prostriedkami pre zobrazovanie scén. Nevýhodou mnohých architektúr (v rátane CUDA) je ich spätosť s konkrétnym hardvérovým zariadením.

Výsledná aplikácia umožňuje jednoduchú úpravu a poskytuje vhodný základ pre implementovanie pokročilejších metód zobrazenia. Ako základné rozšírenia možno spomenúť podporu textúr, implementáciu CSG a ďalších primitív, GUI a mnohé iné. Architektúra

CUDA poskytuje ďalšie možnosti urýchlenia v podobe tvorby KD stromu s využitím grafického akcelerátora. Za dlhodobé ciele možno považovať výpočet globálneho osvetlenia, podporu plošných a iných zdrojov svetla, atď.

Literatúra

[1] Bielefeld Graphics & Geometry Group, Bielefeld University, Germany, [online], 2011.
URL http://graphics.uni-bielefeld.de/research/points/

[2] NVIDIA CUDA – Programming Guide [online]. 2007-11-29 [cit. 2011-04-21], verzia 1.1.
URL http://developer.download.nvidia.com/compute/cuda/1_1/NVIDIA_CUDA_Programming_Guide_1.1.pdf

[3] Wikipedia – The Free Encyclopedia: Lambert's cosine law [online]. 2011-04-16 [cit. 2011-04-21].
URL http://en.wikipedia.org/w/index.php?title=Lambert%27s_cosine_law&oldid=424337297

[4] Wikipedia – The Free Encyclopedia: Phong shading [online]. 2011-04-20 [cit. 2011-04-21].
URL
http://en.wikipedia.org/w/index.php?title=Phong_shading&oldid=424980971

[5] Amanatides, J.; Woo, A.: A Fast Voxel Traversal Algorithm for Ray Tracing. In *Eurographics*, 1987, s. 3–10.

[6] Arvo, J.; Kirk, D.: Particle transport and image synthesis. *SIGGRAPH Computer Graphics*, ročník 24, č. 4, september 1990: s. 137–145, ISSN 0097-8930.

[7] Cook, R. L.; Porter, T.; Carpenter, L.: Distributed ray tracing. *SIGGRAPH Computer Graphics*, ročník 18, č. 3, január 1984: s. 137–145, ISSN 0097-8930.

[8] Danilewski, P.: *Binned kd-tree construction with SAH on the GPU*. Diplomová práca, Universität des Saarlandes, Germany, november 2009.

[9] Fujimoto, A.; Tanaka, T.; Iwata, K.: ARTS: Accelerated Ray-tracing System. In *Tutorial: computer graphics*, New York, NY, USA: Computer Science Press, Inc., 1988, ISBN 0818688544, s. 148–159.

[10] Gouraud, H.: Continuous Shading of Curved Surfaces. *IEEE Transactions on Computers*, ročník 20, č. 6, jún 1971: s. 623–629, ISSN 0018-9340.

[11] Greve, B.: Reflections and Refractions in Ray Tracing [online]. november 2006.
URL http://graphics.stanford.edu/courses/cs148-10-summer/docs/2006--degreve--reflection_refraction.pdf

[12] Havran, V.: *Heuristic Ray Shooting Algorithms*. Dizertačná práca, České vysoké učení technické v Praze – Fakulta elektrotechnická, 2001.

[13] Ize, T.; Wald, I.; Parker, S. G.: Ray Tracing with the BSP Tree [online]. 2008, [cit. 2011-04-21].
URL
http://visual-computing.intel-research.net/publications/BSPRT08.pdf

[14] Jensen, H. W.: *Realistic Image Synthesis Using Photon Mapping*. AK Peters, druhé vydanie, marec 2005, ISBN 1568811470.

[15] Kajiya, J. T.: The rendering equation. *SIGGRAPH Computer Graphics*, ročník 20, č. 4, august 1986: s. 143–150, ISSN 0097-8930.

[16] MacDonald, D. J.; Booth, K. S.: Heuristics for ray tracing using space subdivision. *The Visual Computer*, ročník 6, č. 3, máj 1990: s. 153–166, ISSN 0178-2789.

[17] Möller, T.; Trumbore, B.: Fast, minimum storage ray/triangle intersection. In *ACM SIGGRAPH 2005 Courses*, New York, NY, USA: ACM, 2005.

[18] Nicodemus, F. E.; Richmond, J. C.; Hsia, J. J.: *Geometrical considerations and nomenclature for reflectance*. Washington D.C., USA: U.S. Department of Commerce, október 1977.

[19] Phong, B. T.: *Illumination for computer-generated images*. Dizertačná práca, The University of Utah, USA, júl 1973.

[20] Phong, B. T.: Illumination for computer generated pictures. *Communications of the ACM*, ročník 18, č. 6, jún 1975: s. 311–317, ISSN 0001-0782.

[21] Samet, H.: *Applications of spatial data structures: Computer graphics, image processing, and GIS*. Boston, MA, USA: Addison-Wesley Longman Publishing, Inc., 1990, ISBN 978-0201503005.

[22] Shirley, P.; Morley, R. K.: *Realistic Ray Tracing*. AK Peters/CRC Press, druhé vydanie, júl 2003, ISBN 978-1568811987.

[23] Torrance, K. E.; Sparrow, E. M.: Theory for Off-Specular Reflection From Roughened Surfaces. *Journal of the Optical Society of America*, ročník 57, č. 9, september 1967: s. 1105–1112, ISSN 0030-3941.

[24] Weinzierl, S.: *Introduction to Monte Carlo methods*. Research School Subatomic Physics, Amsterdam, jún 2000.

[25] Whitted, T.: An improved illumination model for shaded display. *Communications of the ACM*, ročník 23, č. 6, jún 1980: s. 143–150, ISSN 0001-0782.

[26] Zhou, K.; Hou, Q.; Wang, R.: Real-time KD-tree construction on graphics hardware. *ACM Trans. Graph.*, ročník 27, č. 5, december 2008: s. 126–137, ISSN 0730-0301.

Dodatok A

Funckia implementovaná pomocou jazyka CUDA C

Na nasledujúcich riadkoch je znázornený príklad implementácie sčínania dvoch matíc s rozmermi $N \times N$ pomocou architektúry CUDA.

```
// definovanie kernelu
__global__ void matAdd (float A[N][N], float B[N][N], float C[N][N])
{
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   int j = blockIdx.y * blockDim.y + threadIdx.y;
   if (i < N && j < N)
      C[i][j] = A[i][j] + B[i][j];
}

int main()
{
   // volanie kernelu
   dim3 dimBlock(16, 16);
   dim3 dimGrid( (N + dimBlock.x - 1) / dimBlock.x,
                 (N + dimBlock.y - 1) / dimBlock.y);
   matAdd<<<dimGrid, dimBlock>>>(A, B, C);
}
```

Algoritmus A.1: Sčítanie dvoch matíc pomocou jazyka CUDA C

Dodatok B

Syntax súboru pre načítanie objektov

Syntax súboru vychádza z formátu *wavefront*, a preto je pre načítanie modelov možné použiť súbory typu *.obj*. Aplikácia umožňuje načítanie vrcholov, normálových vektorov a polygónov, ostatné parametre nachádzajúce sa v súboroch nie sú podporované. Každý parametre je definovaný na samostatnom riadku:

- `v` *`x y z`* – vrchol trojuholníka
  (*`x`*, *`y`*, *`z`* určujú pozíciu vrcholu a sú typu `float`),
- `vn` *`x y z`* – normálový vektor
  (*`x`*, *`y`*, *`z`* definujú smer normálového vektora, sú typu `float`),
- `f` *`v1 v1 v3`* – trojuholník
  (*`v1`*, *`v2`*, *`v3`* reprezentujú indexy vrcholov, sú typu `unsigned int`),
- `f` *`v1//n1 v2//n2 v3//n3`* – trojuholník so zadanými normálovými vektormi
  (*`v1`*, *`v2`*, *`v3`* definujú indexy vrcholov a *`n1`*, *`n2`*, *`n3`* určujú indexy normálových vektorov, všetky sú typu `unsigned int`).

Na príklade B.1 je znázornená syntax reprezentujúca jednoduchý polygonálny objekt – kocku.

```
v 25.0 25.0 25.0
v -25.0 25.0 25.0
v -25.0 -25.0 25.0
v 25.0 25.0 -25.0
v 25.0 -25.0 25.0
v -25.0 25.0 -25.0
v 25.0 -25.0 -25.0
v -25.0 -25.0 -25.0

vn 0.0 0.0 1.0
vn 1.0 0.0 0.0
vn 0.0 0.0 -1.0
vn -1.0 0.0 0.0
vn 0.0 1.0 0.0
vn 0.0 -1.0 0.0

f 1//1 2//1 3//1
f 4//2 1//2 5//2
f 6//3 4//3 7//3
f 2//4 6//4 8//4
f 4//5 6//5 2//5
f 5//6 3//6 8//6
f 5//1 1//1 3//1
f 7//2 4//2 5//2
f 8//3 6//3 7//3
f 3//4 2//4 8//4
f 1//5 4//5 2//5
f 7//6 5//6 8//6
```

Príklad B.1: Syntax súboru pre načítanie kocky

Dodatok C

Testovacia zostava

Pre testovanie bola použitá zostava:

Procesor: Intel Core 2 Duo E7200 (2.53 GHz)

Grafická karta: Asus ENGTX460 TOP

Operačná pamäť: 2GB DDR2 (800 Mhz)

Operačný systém: Microsoft Windows 7

Základné vlastnosti grafickej karty:

Jadro:

nVidia GeForce GTX 460 (GF104)
Výrobná technológia: 40 nm
Počet stream procesorov: 336

Pamäť:

Kapacita: 768 MB
Typ: GDDR5
Zbernica: 192-bitová

Frekvencia:

Jadro: 700 Mhz
Shadery: 1400 Mhz
Pamäť: 3600 Mhz

Technológia:

Compute capability 2.1

Výkon:

907,2 GFLOPs

Dodatok D

Popis scény 1A, 1B, 1C a 1D

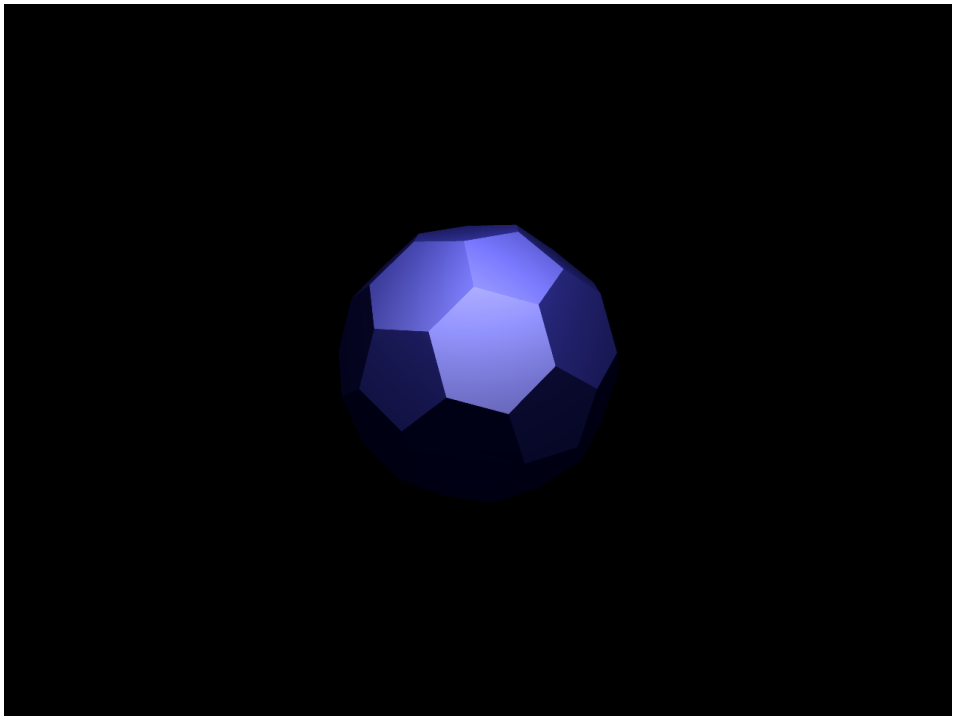
D.1 Základné informácie

Scény sú reprezentované jediným modelom a obsahujú jeden svetelný zdroj. Celkový počet vrcholov a trojuholníkov je uvedený v tabuľke D.1.

Označenie scény	Počet vrcholov	Počet trojuholníkov
1A	60	116
1B	17324	34048
1C	34835	69666
1D	50000	100000

Tabuľka D.1: Informácie o základnej štruktúre jednotlivých scén

D.2 Obrazová príloha



Obrázok D.1: Scéna 1A

Obrázok D.2: Scéna 1B

Obrázok D.3: Scéna 1C

Obrázok D.4: Scéna 1D

Dodatok E

Popis scény 2A, 2B, 2C a 2D

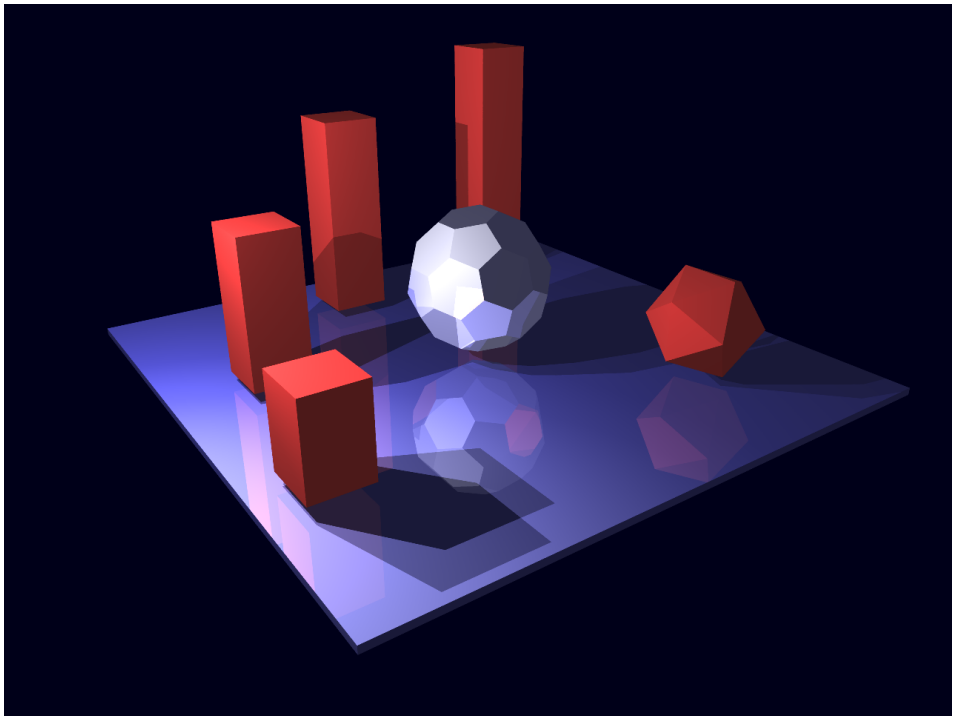
E.1 Základné informácie

Informácie o základnej štruktúre jednotlivých scén sú uvedené v nasledujúcej tabuľke E.1.

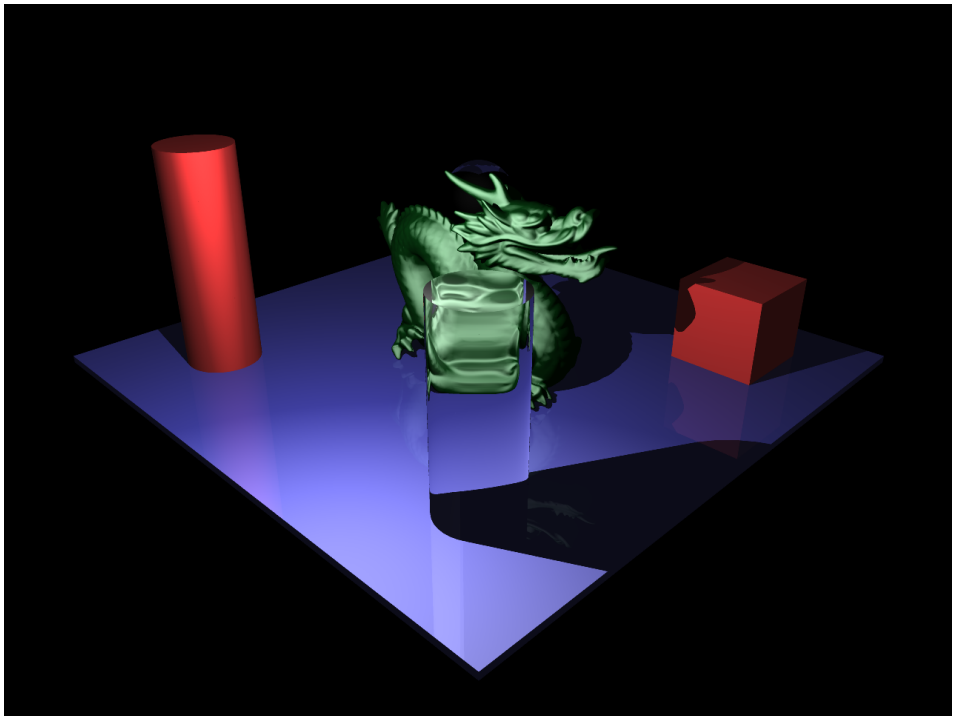
Označenie scény	Počet svetelných zdrojov	Počet objektov	Počet vrcholov	Počet trojuholníkov
2A	2	7	112	196
2B	1	6	54310	108312
2C	1	7	329	486
2D	1	8	54839	108906

Tabuľka E.1: Informácie o základnej štruktúre jednotlivých scén

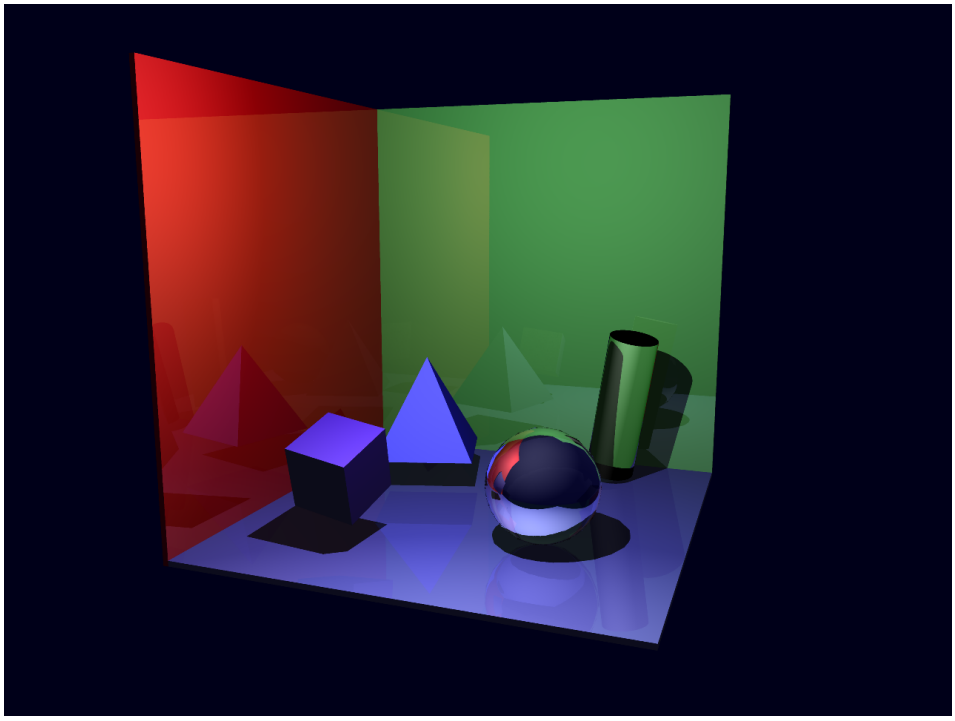
E.2 Obrazová príloha



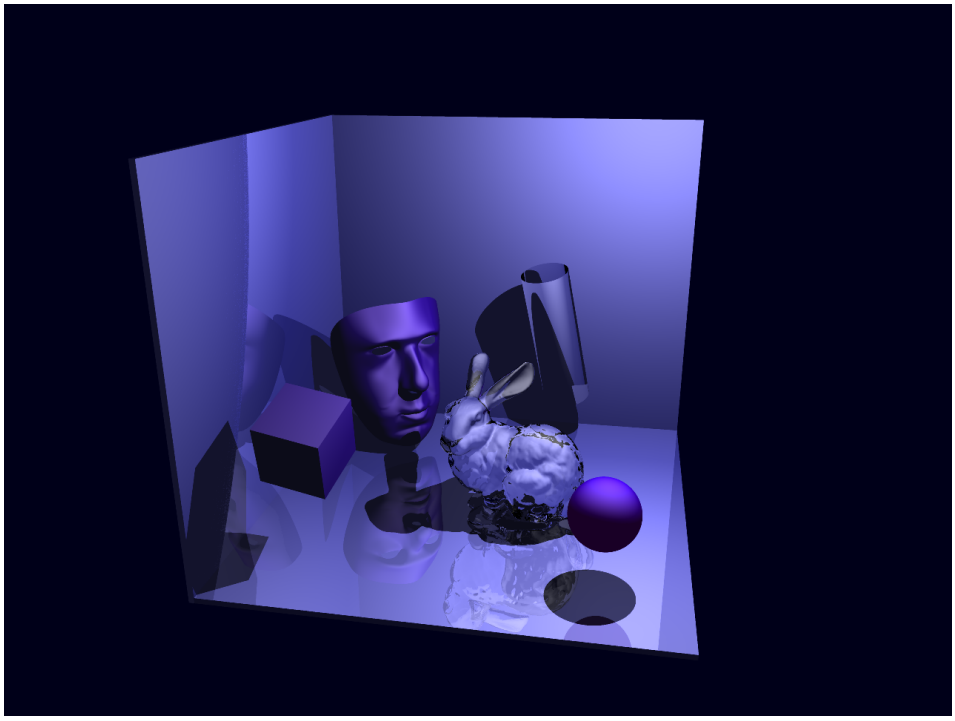
Obrázok E.1: Scéna 2A



Obrázok E.2: Scéna 2B



Obrázok E.3: Scéna 2C



Obrázok E.4: Scéna 2D

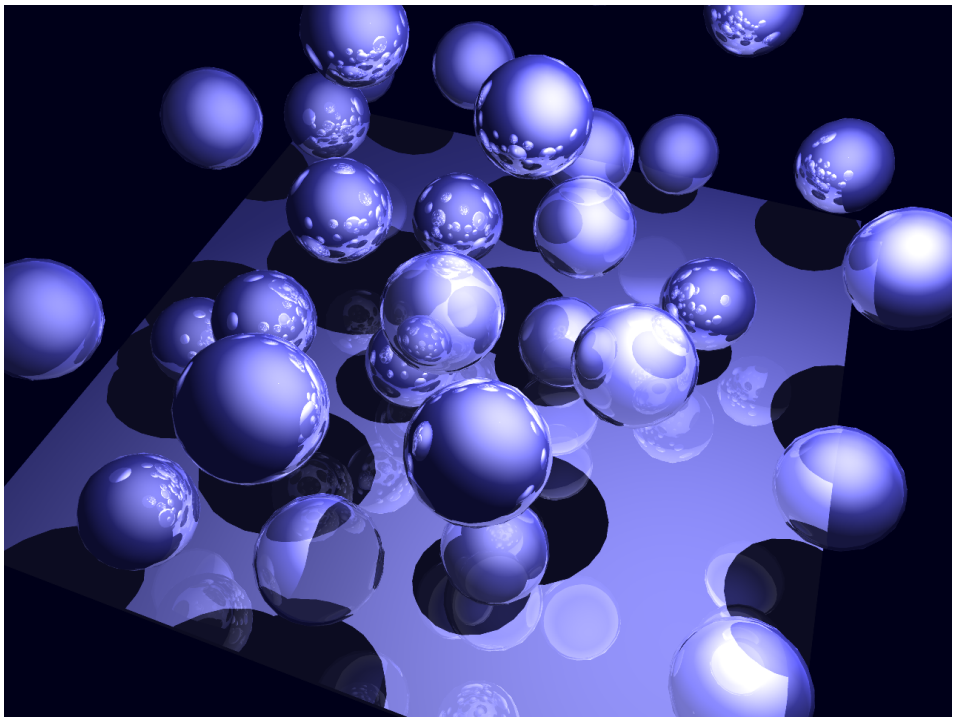
Dodatok F

Popis scény 3

F.1 Základné informácie

- počet svetelených zdrojov: 1
- počet objektov: 31
- počet vrcholov: 4388
- počet trojuholníkov: 8652

F.2 Obrazová príloha



Obrázok F.1: Scéna 3

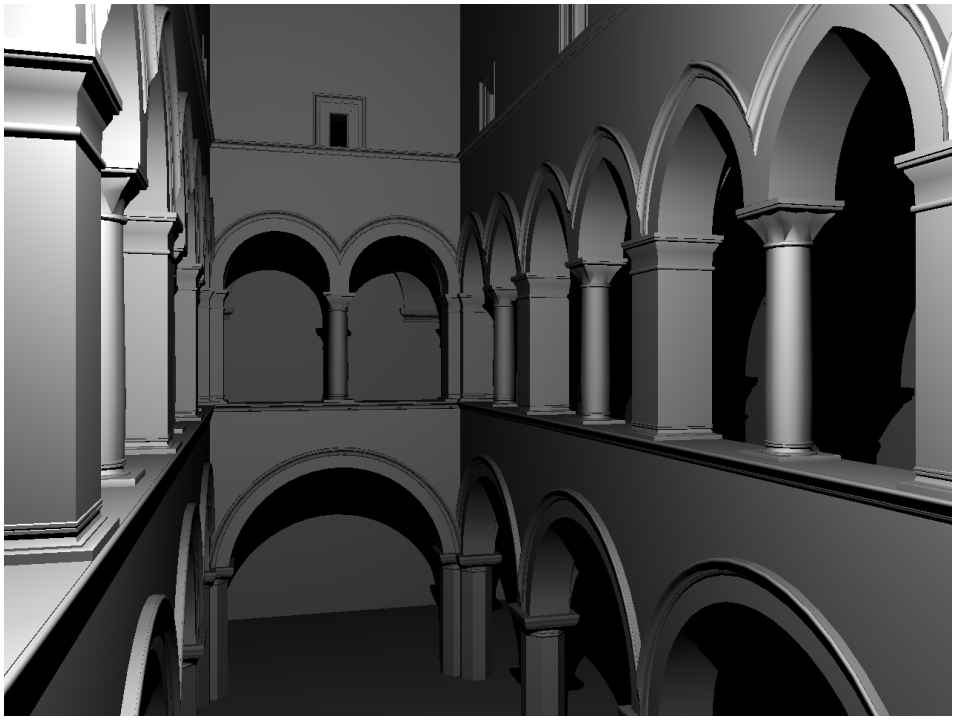
Dodatok G

Popis scény 4A, 4B a 4C

G.1 Základné informácie

Všetky scény obsahujú model *Sponza Atrium*, ktorý je zložený z 39742 vrcholov a 76154 trojuholníkov. Pre osvetlenie bol použitý jediný svetelný zdroj. Scény sa navzájom líšia umiestnením kamery pre zachytenie výsledného obrazu.

G.2 Obrazová príloha



Obrázok G.1: Scéna 4A

Obrázok G.2: Scéna 4B

Obrázok G.3: Scéna 4C

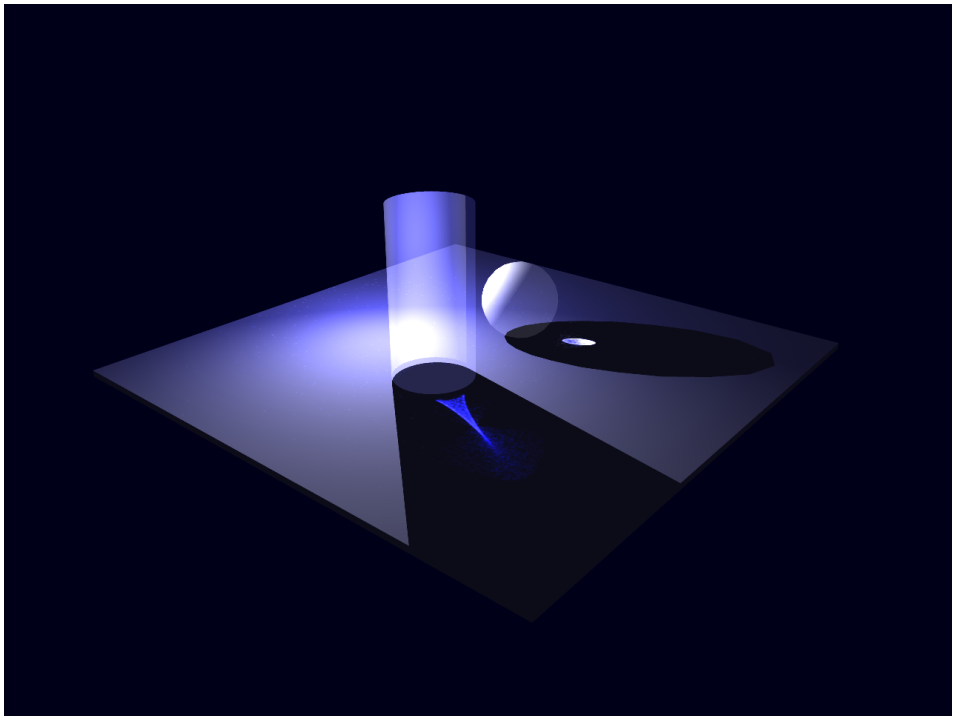
Dodatok H

Popis scény 5

H.1 Základné informácie

Scéna obsahuje jeden svetelný zdroj a tri jednoduché objekty zložené z 316 vrcholov a 476 trojuholníkov.

H.2 Obrazová príloha



Obrázok H.1: Scéna 5

Dodatok I

Obsah priloženého DVD

Na DVD sa nachádzajú nasledujúce zložky a súbory:

- `bin/` – spúšťacie súbory aplikácie (rôzne scény) a potrebné knižnice,
- `doc/` – adresár obsahujúci textovú časť diplomovej práce vo formáte *.pdf*,
- `doc_src/` – zdrojové kódy textovej časti diplomovej práce spolu s použitými obrázkami,
- `manuals/` – zložka obsahujúca rôzne manuály pre skompilovanie zdrojových súborov aplikácie,
- `outputs/` – adresár obsahujúci výstupné obrázky a videá,
- `poster/` – zložka obsahujúca plagát v rôznych formátoch,
- `src/` – zdrojové kódy aplikácie,
- `README.txt` – stručný popis aplikácie, obsah DVD, atď.

Dodatok J

Plagát

Raytracing na GPU

budúcnosť zobrazovania

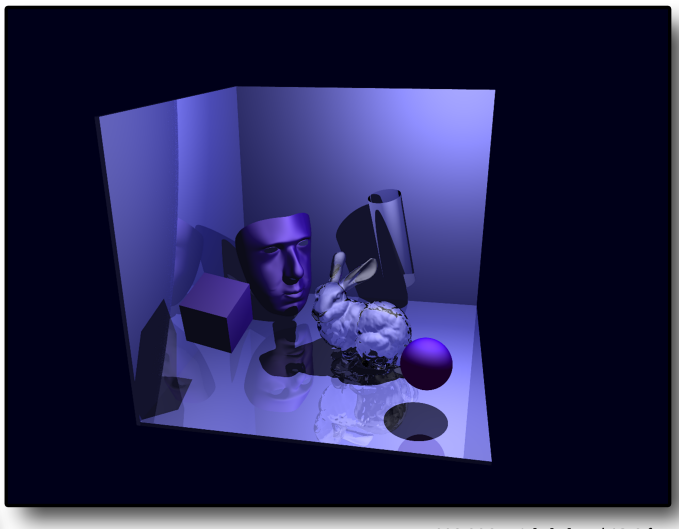
GPGPU

- *paralelné spracovanie*
- *vysoký výpočtový výkon*
- *neustály vývoj*
- *nové možnosti*

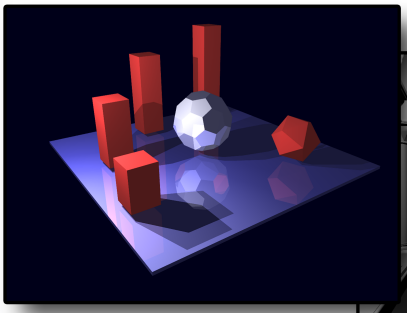
Implementovaný raytracer

- *odrazené a lomené lúče*
- *Phongov osvetľovací model*
- *bodové zdroje svetla*
- *antialiasing*
- *akceleračná štruktúra KD tree*
- *kaustiky*
- *pixel buffer object*

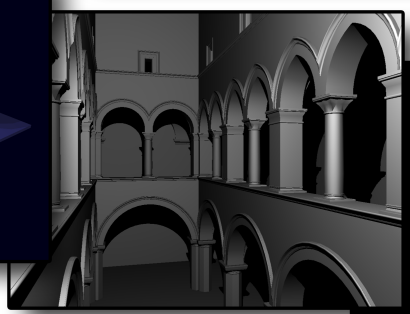
640 x 480 px



108 906 trojuholníkov / 12,6 fps



196 trojuholníkov / 124,2 fps



76 154 trojuholníkov / 3,6 fps

34 048 trojuholníkov / 47,8 fps

NVIDIA GeForce with CUDA

Marek Straňák, plagát vznikol ako súčasť diplomovej práce na FIT VUT v Brne, 2011