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

RAY-TRACING S KNIHOVNOU IPP

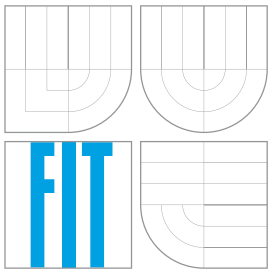
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MICHAL KUKLA

BRNO 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

RAY-TRACING S KNIHOVNOU IPP

RAY-TRACING USING IPP LIBRARY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. MICHAL KUKLA
AUTHOR

VEDOUCÍ PRÁCE — Ing. MICHAL HRADIŠ
SUPERVISOR

BRNO 2010

Abstrakt

Práce se zabývá návrhem a implementací ray-tracingu a path-tracingu s využitím knihovny IPP. Teoretická část diskutuje současné postupy při akceleraci zmíněných algoritmů a možnost paralelizace. V další části práce je popsán návrh algoritmů ray-tracingu a path-tracingu a způsob paralelizace zmiňovaných algoritmů. Tato část taktéž diskutuje možnosti implementace adaptivního vzorkování a metody importance sampling v souvislosti s metodou Monte Carlo pro urychlení algoritmu path-tracingu. Další část se zabývá postupem implementace zmíněných zobrazovacích algoritmů v kontextu knihovny IPP a také využitím knihovny Boost při tvorbě síťového rozhrání aplikace. Implementované postupy jsou v závěru práce podrobeny testům výkonnosti a kvality zobrazení pro stanovení úspešnosti zvolených postupů. Výstupem práce je serverová aplikace schopna současné obsuhy více klientů poskytující vizualizaci a klientská aplikace implementující ray-tracing a path-tracing.

Abstract

Master thesis is dealing with design and implementation of ray-tracing and path-tracing using IPP library. Theoretical part discusses current trends in acceleration of selected algorithms and also possibilities of parallelization. Design of ray-tracing and path-tracing algorithm and form of parallelization are described in proposal. This part also discusses implementation of adaptive sampling and importance sampling with Monte Carlo method to accelerate path-tracing algorithm. Next part is dealing with particular steps in implementation of selected rendering methods regarding IPP library. Implementation of network interface using Boost library is also discussed. At the end, implemented methods are subjected to performance and quality test. Final product of this thesis is server aplication capable of handling multiple connections which provides visualisation and client application which implements ray-tracing and path-tracing.

Klíčová slova

ray-tracing, path-tracing, Intel integrated performance primitives, IPP, realistické vykreslování, importance sampling, Monte Carlo, koherence paprsků, SIMD, adaptivní vzorkování, Phongův osvětlovací model, Boost, síťový rendering, paralelizace, zobrazování v reálnem čase

Keywords

ray-tracing, path-tracing, Intel integrated performance primitives, IPP, realistic rendering, importance sampling, Monte Carlo, ray coherence, SIMD, adaptive sampling, Phong shading, Boost, network rendering, parallelization, realtime rendering

Citace

Michal Kukla: Ray-tracing s knihovnou IPP, diplomová práce, Brno, FIT VUT v Brně, 2010

Ray-tracing s knihovnou IPP

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Michala Hradiše. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Michal Kukla
26. května 2010

Poděkování

Děkuji Ing. Michalovi Hradišovi za jeho odborné vedení mé práce, za rady, pomoc a cenný čas, který mi během vypracovávání práce věnoval.

Obsah

1 Úvod **3**

2 Raytracing v reálnom čase **4**
2.1 Akceleračné štruktúry . . . 4
2.1.1 Octree . . . 4
2.1.2 Kd-tree . . . 5
2.1.3 BSP tree . . . 5
2.2 Bounding volume hierarchies . . . 6
2.3 Mriežky . . . 7
2.4 Koherencia lúčov . . . 9
2.5 Paralelný ray-tracing . . . 10

3 Vyššie metódy zobrazenia **12**
3.1 BRDF . . . 12
3.2 Renderovacia rovnica . . . 15
3.3 Monte Carlo . . . 18
3.4 Path tracing . . . 21
3.5 Photon-mapping . . . 25
3.6 Paralelizácia výpočtu globálneho osvetlenia . . . 26

4 Návrh implementácie **27**
4.1 Intel Integrated Performace Primitives . . . 27
4.2 Dáta scény . . . 28
4.3 Zobrazovací reťazec ray-tracingu . . . 30
4.4 Zobrazovací reťazec path-tracingu . . . 31
4.5 Paralelizácia výpočtu . . . 34
4.6 Komunikácia uzlov . . . 36

5 Implementácia **39**
5.1 Zobrazovacia konzola . . . 39
5.2 Výpočtový uzol . . . 41
5.3 Implementácia ray-tracingu . . . 41
5.4 Implementácia path-tracingu . . . 42

6 Dosiahnuté výsledky **44**
6.1 Urýchlenie výpočtu paralelizáciou . . . 44
6.2 Kvalita výstupu . . . 46
6.3 Budúci vývoj . . . 47

7 Záver **50**

A Obsah CD **54**

B Manuál k programu **55**

C Plagát **58**

Kapitola 1

Úvod

Stúpajúci výpočtový výkon hardware v posledných rokoch posunul možnosti implementácie zobrazovacích metód od jednoduchých rasterizačných metód po v súčasnosti existujúce implementácie ray-tracingu a metód výpočtu globálneho osvetlenia v reálnom čase. V súčasnej dobe existuje viacero implementácii využívajúcich sledovanie lúča v reálnom čase. Intel prezentoval realtime ray-tracing v engine Quake wars [26], kde možno porovnať sledovanie lúča s pôvodným rasterizačným algoritmum na totožných dátach. Implementáciu zahrňujúcu interaktívne globálne osvetlenie a renderovanie rozsiahlych modelov predstavuje knižnica OpenRT [2], využívajúca rozhranie podobné OpenGL API. Prístup využívajúci technológiu SIMD je prezentovaný v implementácii Arauna realtime ray-tracing [6] resp. [7]. Spomenuté riešenia však aj napriek optimalizácii vyžadujú pre realtime zobrazenie s odpovedajúcim počtom snímkov výkonné paralelizované systémy. Odhliadnuc od zábavného priemyslu má realtime ray-tracing a metódy globálneho osvetlenia uplatnenie pri vizualizácii vo viacerých odvetviach. Architektúra má v súčasnosti možnosť pred samotnou realizáciou stavby umožniť investorom realistickú vizualizáciu presvetlenia miestností využitím metód globálneho osvetlenia v reálnom čase. Práca [36] spoluautora OpenRT sa zaoberá možnosťou využitia fyzikálneho modelu osvetlenia , či vizualizáciou rozsiahlych scén.

Z tohoto dôvodu vznikla snaha výpočty optimalizovať pre využitie možností súčasného hardware. V prípade ray-tracingu a metód výpočtu globálneho osvetlenia scény je predmetom výskumu optimalizácia dátových štruktúr, adaptácia algoritmov pre využitie technológií dnešných CPU a v neposlednom rade spôsob paralelizácie výpočtu. Zaujímavým trendom v tejto súvislosti je aj vývoj hardware. Riešenie Intelu Larrabee [31] , postavené na many-core architektúre GPU s inštrukčnou sadou a flexibilitou súčasných procesorov, možno onedlho vytvorí konkurenciu rasterizačným algoritmom.

Predmetom mojej diplomovej práce je návrh a implementácia zobrazovania sledovaním lúča v reálnom čase s využitím knižnice Intel Integrated Performace Primitives, ktorá umožňuje vyvýjať platformovo nezávislé optimalizované riešenia výpočtovo náročných úloh.

Práca ďalej diskutuje viaceré aspekty a možnosti akcelerácie dôležitých častí ray-tracingu. V ďalšej časti práce sa zaoberám diskusiou súčasných postupov výpočtu globálneho osvetlenia, vhodnosťou paralelizácie pre konkrétne metódy a možnosťou implementácie. Teoretické východiská problematiky ďalej transformujem do návrhu implementácie. V tejto časti predkladám postup riešenia s využitím knižnice IPP. Postupy a prostriedky využité pri implementácii ako aj opodstatnenie ich využitia diskutujem v kapitole 5. V závere práca hodnotí dosiahnuté výsledky a črtá možnosti ďalšieho vývoja.

Kapitola 2

Raytracing v reálnom čase

V tejto kapitole budem diskutovať súčasný stav poznatkov týkajúcich sa urýchlenia výpočtu ray-tracingu. V krátkosti uvediem používané dátové štruktúry a výpočtovú náročnosť vyhľadávania priesečníkov. Zameriam sa taktiež na technológie súčasných CPU a možnosť ich využitia pri implementácii. V závere kapitoly načrtnem východiská paralelizácie výpočtu v súvislosti s ray-tracingom a metódami výpočtu globálneho osvetlenia.

2.1 Akceleračné štruktúry

Kľúčovou súčasťou výpočtu pri ray-tracingu je hľadanie priesečníkov lúčov s prvkami scény, preto vzniklo viacero štruktúr pre uloženie dát scény, ktoré umožňujú urýchliť výpočet ray-tracingu v porovnaní s triviálnou metódou testu lúča so všetkými prvkami scény.

2.1.1 Octree

Štruktúra octree vznikne rozšírením quadtree do troch rozmerov, rozdeľuje priestor v každom kroku pomocou 2 hyperplôch na 8 zhodných podpriestorov. Z hierarchického hľadiska sú jednotlivé oktanty reprezentované stromovou štruktúrou, ktorej každý podpriestor je reprezentovaný uzlom stromu a elementárny podpriestor listom stromu. Zostavenie takejto štruktúry vzhľadom na pevne dané umiestnenie hyperplôch je pomerne rýchle. Horný odhad časovej zložitosti vyhľadávania konkrétneho listu je podľa [19]:

$$O(log_8 N) \tag{2.1}$$

Štruktúru možno linearizovať a dosiahnuť tak časovú zložitosť vyhľadania uzla podľa [19] na úrovni:

$$O(log_2 N) \tag{2.2}$$

Táto štruktúra nieje vhodná pre scény s nerovnomerným rozložením objektov, vznikajú prázdne listy, čo v konečnom dôsledku predlžuje čas vyhľadania daného objektu. V ray-tracingu je zaujímavá rýchlosť zostavenia tejto štruktúry a jej využitie pri dynamických scénach, kde je možné pristupovať do suboktantov a tieto prebudovávať na základe lokálneho pohybu objektu.

2.1.2 Kd-tree

Kd-tree je stromová štruktúra, špeciálny prípad BSP-tree. Podobne ako octree pre delenie využíva hypeplochy, avšak priestor rozdeľuje na rôzne veľké podpriestory. Štruktúru možno implementovať ako binárny strom, ktorého každý uzol, ktorý nieje list obsahuje podstrom podpriestoru daný hyperplochou rodiča, ktorá tento uzol definuje. Schému rozdeľovania vidieť v obr. 2.1.2

Vytvorenie Kd-tree je na rozdiel od octree netriviálny problem. Umiestnenie hyperplochy je v tomto prípade vzhľadom na podpriestor a jeho rozmery variabilné a možností kam umiestniť deliacu plochu je nekonečne veľa. Pre určenie polohy hyperplochy bolo vyvinutých viacero heuristík. Použitie konkrétnej závisí od účelu vybudovanej štruktúry. Triviálne určenie polohy pre umiestnenie hyperplochy je vyvážený počet primitív v oboch vzniknutých podpriestoroch. Predpoklad, že takéto rozmiestnenie je optimálne vyvracia [37]. Opiera sa o predpoklad, že pri hľadaní prisečníkov v podpriestore nemá počet obsiahnutých primitív takú dôležitosť ako ich celková plocha. Preto sa takmer výlučne používajú alternatívy SAH (Surface area heuristics), kde pre umiestnenie hyperplochy je do výpočtu zahrnutá plocha jednotlivých primitív. Heuristika vychádza z predpokladu o tom, že pri použití zväzku lúčov stúpa pravdepodobnosť detekcie priesečníka s väčšou plochou polygónu a počet testov je v rámci daného listu kd-stromu vo väčšine prípadov minimalizovaný. Nájdenie optimálnej hyperplochy sa však podľa [15] asymtoticky približuje k:

$$O(N^2) \tag{2.3}$$

pri naivnom riešení, kde N je počet primitív v danom priestore. Pokiaľ uvažujeme statické scény, je čas vybudovania štruktúry menej dôležitý ako rýchlosť a optimálnosť obsahu jednotlivých listov stromu, preto je hladanie optimálneho rozdelenia v kontexte mojej práce dôležité pri použití modifikovanej SAH heuristiky je možné optimalizovat na:

$$O(NlogN) \tag{2.4}$$

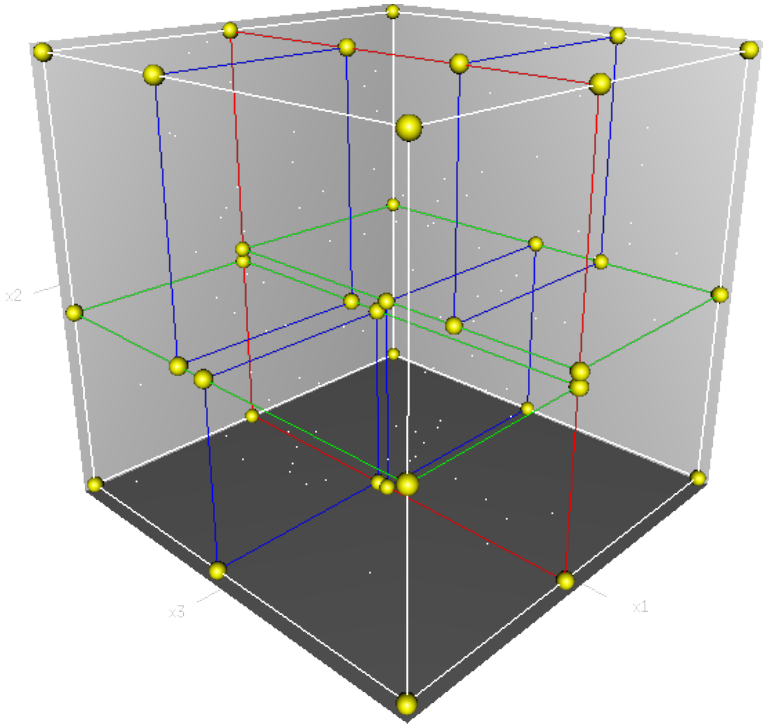
Metóda SAH je podrobne popísaná v [15] [37]. Z hľadiska povahy spracovávaných dát trpí Kd-tree nedostatkami pri veľkom počte primitív rovnobežných so súradnicovým systémom, možno to badať v scénach s architektúrou. Vznikajú tu prázdne podpriestory a naopak podpriestory tesne obklopujúce geometriu.

2.1.3 BSP tree

BSP používa v porovnaní s Kd-tree všeobecné hyperplochy a mohol by sa označiť za optimálnu metódu akcelerovania ray-tracingu. Toto tvrdenie však vyvracia [15]. Použitie obecných hyperplôch rieši problémy pretiahnutých polygónov a architektonických scén, na druhej strane však je pri určení danej hypeplochy potrebné definovať 3 body ležiace v tejto ploche. Následné rozdelenie podpriestoru v prostredí trpí zaokrúhľovacími chybami, čo môže viesť k zobrazovacím chybám resp. k zvýšeniu výpočtovej náročnosti špecifických častí scény.

Pri budovaní BSP stromu je hľadanie optimálnej hyperplochy pre rozdelenie priestoru v podstate prehľadávanie 3-rozmerného priestoru všetkých možných riešení. To vedie pri naivnom riešení na časovú zložitosť podľa [37]:

$$O(N^3) \tag{2.5}$$



Obrázek 2.1: Schéma delenia priestoru a tvorenia uzlov Kd-tree prevzaté z [39]

kde N je počet primitív v danom priestore. Zrýchleniu opätovne prispieva využitie heuristiky SAH, ktoré v ideálnom prípade zníži zložitost na úroveň subkvadratickej:

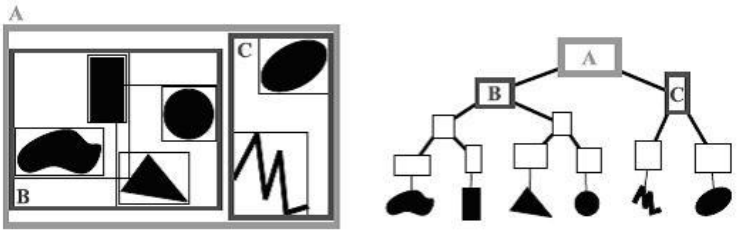
$$O(Nlog^2N) \tag{2.6}$$

BSP tree má využitie pri renderovaní rozsiahlych scén s nerovnomerným rozložením objektov v nich. Neortogonálne deliace hyperplochy sú vhodné pre architektonické modely. V [37] je navrhnutá metóda hybridného stromu spojením rýchlosti budovania Kd-stromu a výhod BSP stromu v oblasti zložitej geometrie.

2.2 Bounding volume hierarchies

Bounding volume hierarchies je stromová štruktúra na množine objektov, kde sú objekty umiestnené v listoch reprezentujúcich obálky. Keďže ide o hierarchické usporiadanie, obálky sú vzájomne vnorené. Pri volení parametrov tejto štruktúry sa musíme zamerať na viacero kritérií, podľa ktorých zostavíme daný strom (na základe pamäťových obmedzení a v prípade ray-tracingu tvaru obálok). Výber tvaru obálky je pre optimálne riešenie dôležitou časťou zostavenia algoritmu tvorby a prechádzania štruktúry, na tvar obálky sú kladené viaceré požiadavky ako je minimalizovanie veľkosti dát popisujúcich tvar z hľadiska pamäťovej náročnosti, rýchle testovanie priesečníka s lúčom a rýchle určenie veľkosti a polohy pri zostavovaní.

Podľa [13], experimenty na typických scénach potvrdili, že ideálnou reprezentáciou obálky je minimálny osovo rovnobežný kváder. Ďalšie experimenty diskutujú možnosť využitia sfér, dvojice kvádrov, avšak neúspešne. Optimálnosť použitia konkrétneho útvaru je závislá od vstupných dát. Zostavenie štruktúry podľa [18] podlieha viacerým kritériám.



Obrázek 2.2: Schéma delenia priestoru a tvorenia uzlov z [13]

Dôležitým faktorom je celkový objem obálok, ktorý by mal byť minimálny spolu s ich počtom. V dynamických scénach je nutné zabezpečiť rýchle prebudovanie štruktúry v čase zanedbateľnom v porovnaní s vyhľadávaním.

Pri budovaní štruktúry je dôležité určiť počet potomkov nelistového uzla. Rôzne experimenty [18] naznačujú, že klasický model binárneho stromu je jednak implementačne nenáročný a aj v zložitých scénach prináša relatívne uspokojujúce výsledky. Naviac tento prístup možno kombinovať s linearizovaním zoznamu s prakticky okamžitým vyhľadaním príslušného listu. Pokiaľ ide o metódy zostavenia, možno použiť metódu zhora-dole, implementačne jednoduchú. Tento prístup v prvom kroku vytvorí obálku celej scény, tá je následne rozdelená pri dodržani princípov v [18] na 2 podstromy. Takto postupujeme rekurzívne pokial nevyhovieme určenej podmienke ukončenia algoritmu. Algoritmus zdola-hore má na vstupe všetky dáta, nad ktorými prevedie testovanie a podľa podmienky ukončenia algoritmu vytvorí listy, tieto následne rekurzívne zapúzdruje do obálok vyššej úrovne až sa dosiahne koreň štruktúry - algoritmus sa ukončí. Metóda bottom-top vedie v mnohých prípadoch na vyváženejšiu štruktúru a v kontexte ray-tracingu nevznikajú v scéne tzv. hotspots, miesta s výrazne vyššími časmi renderovania, preto je vhodná na predspracovanie v statických scénach.

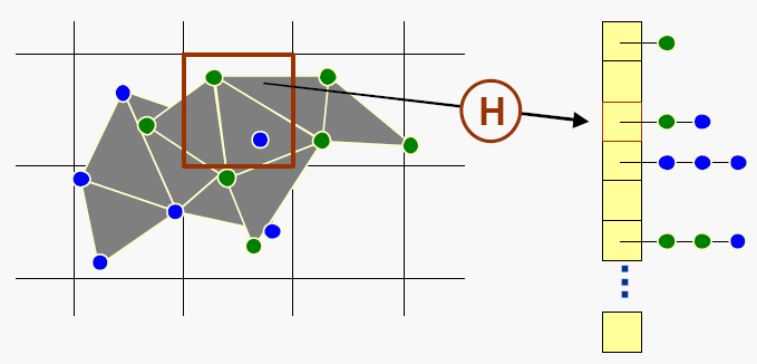
Vyhľadanie listu v štruktúre prebieha prehľadávaním od koreňa stromu, možno teda eliminovať lúče, ktoré smerujú mimo zvoleneho objemu. Podľa implementácie [18], možno pri predošlom použití zo súradnicovými osami zarovnaného kvádra dosiahnuť v najhoršom prípade časovú zložitosť:

$$O(log^2 N) \tag{2.7}$$

, kde N je počet primitív v danom liste. V praxi je však takýto scenár nedosiahnuteľný. Dôvodom je zjednodušenie obálky objektu, detekcia intersekcie môže nastať aj v prípade že samotný lúč obsiahnutú geometriu v liste minie. Dôležitým je však schopnosť štruktúry dosahovať vysoký prehľadávací výkon aj pri dynamických scénach, spolu s implementovaním sledovania zväzku lúčov optimalizovaných pre SIMD je príkladom implementácia Jacco Bikker-a [6]. V praxi taktiež nieje možné predchádzať prekrývaniu jednotlivých obálok, avšak urýchlenie hľadania priesečníkov je oproti spomaleniu vplyvom redundantného prehľadávania stromu zanedbateľné. Spolu s uspokojivým akcelerovaním tvoria BVH ideálneho kandidáta pre implementáciu a akcelerovanie ray-tracingu.

2.3 Mriežky

Predošlé metódy boli založené na vzájomnej polohe jednotlivých objektov. Pravidelné mriežky umožňujú akcelerovať hľadanie priesečníkov objektov s lúčmi rovnomerne rozmiest-

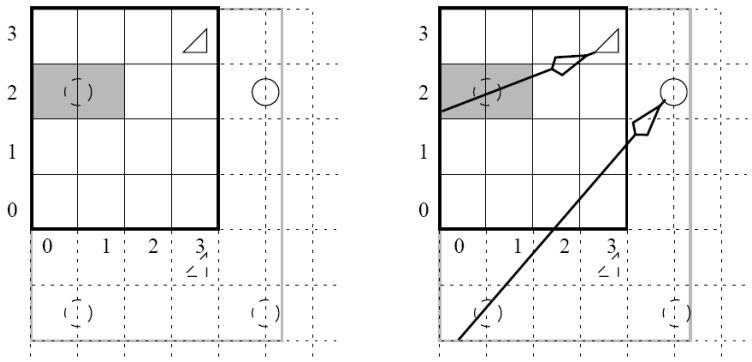


Obrázek 2.3: Schéma delenia priestoru a tvorenia uzlov pri použití jednoúrovňového gridu z [28]

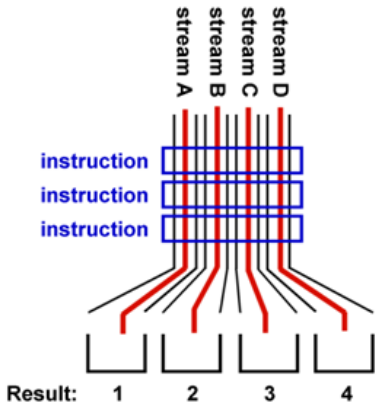
nenými priestormi, z ktorých každý obsahuje informáciu o obsiahnutej geometrii. Naviac je možné túto štruktúru vytvoriť hierarchicky a v prípade potreby je možné vytvoriť mriežku, ktorá bude mať vyššie rozlíšenie. Takáto konštrukcia vedie na úspešné eliminovanie redundantných testov a keďže je mriežka rovnobežná zo súradnicovými osami, je prechádzanie mriežkou optimalizované.

Ako možno vidieť zo schémy v obrázku 2.3 zostavenie jednoúrovňového gridu pozostáva z umiestnenia vzťažnej sústavy do stredu objemu scény. Týmto sa minimalizuje nepresnosť v určovaní polohy a náležitosti jednotlivých primitív do podpriestorov. Obsah každého voxelu je následne popísaný lineárnym zoznamom, aj tu je však možné riešenie pomocou hybridných štruktúr, tie by si však vyžadovali vytvorenie metriky pre výber z dostupných implementovaných. Tento postup by však neúmerne predĺžil predspracovanie scény. Pri väčších objemoch dát a v prípadoch "neideálnych scén"je možné vytvoriť v každom voxeli vnorený grid a takto postupovať rekurzívne, pokiaľ nieje splnená podmienka ukončenia algoritmu daná počtom primitív obsiahnutých vo voxeli resp. hĺbkou zanorenia. Podľa [28] je však využitie gridu v porovnaní s ostatnými metódami vhodné pre implementáciu dynamických scén s premenným počtom objektov. Prehľadávanie, vkladanie a odoberanie objektov je implementačne jednoduché. Metóda použitá v [28] umožňuje postihnúť i objekty, ktoré vplyvom pohybu opustia hranice spracovaného priestoru a to aj bez znovuvybudovania koreňovej obálky a následného prepočítania na vťažnú sústavu gridu. Je to dosiahnuté vytvorením tzv logickej obálky okolo pohybujúcich sa objektov a umiestnenia tejto obálky do priestoru koreňovej obálky. Pre výpočet intersekcie sa upraví poloha stredu zobrazenia ako možno vidieť na 2.4. V ľavom obrázku je sféra mimo koreňovej obálky posunutá do priestoru scény, inverzná transformácia je aplikovaná na polohu pozorovateľa. V každom testovanom prípade uvedenom v [28] bol čas vybudovania novej štruktúry v porovnaní s vytvorením logického gridu v dynamických scéna rýchlejší. Flexibilita tejto štruktúry a spomenutej metódy ju predurčuje k využitiu v dynamických scénach.

Prehľadávanie štruktúry je založené na hľadaní priesečníka s pravideľnou mriežkou tento proces je optimalizovaný a využíva skutočnosť, že v scénach s nerovnomerne rozloženými objektami neobsahuje každý voxel geometriu. V každom bode scény možno polohu bodu vzhľadom na grid určiť jednoduchým zaokrúhlením jeho hodnoty.



Obrázek 2.4: Vytvorenie logického gridu a premietnutie do priestoru scény [28]

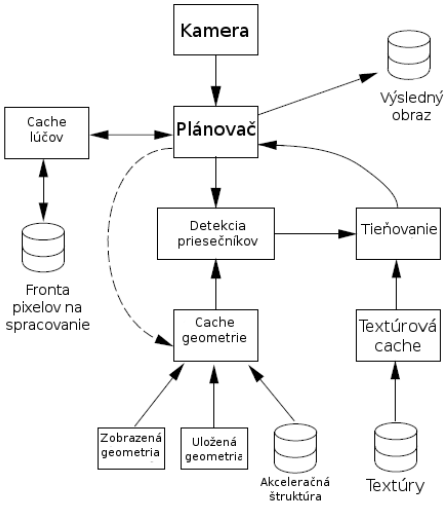


Obrázek 2.5: Spracovanie prúdu dát pomocou architektúry SIMD podľa [31]

2.4 Koherencia lúčov

Koherencia je vo všeobecnosti podobnosť resp. zhodnosť vlastností prvkov danej množiny, pri aplikácii na lúče predpokladáme, že určité atribúty sú v lokálnych zväzkoch konštantné. Naviac podľa [12] rozlišujeme viacero druhov koherencie lúčov. V súvislosti s ray tracingom možno hovoriť o koherencii hĺbky resp. o objektovej koherencii. V objektovej koherencii predpokladáme, že lokálne konštantným atribútom je objekt, v našom prípade primitív scény. Zvyčajne sa tento postup realizuje použitím zväzku lúčov. Aby bolo možné realizovať takýto postup, je potrebné aby sme dátové štruktúry prispôsobili pre hľadanie priesečníkov využitím zvazku lúčov. V súvislosti s výpočtom vačšieho množstva lúčov vo zväzku a všeobecne spracovávaním koherentných dát bolo vivinutých viacero technológií, ktoré túto vlastnosť využivajú. Jednou z najviac využivaných technológii je SIMD (Single instruction multiple data). Procesory s touto architektúrou dokážu spracovávať prudy dát o rôznej šírke (záleží od konkrétnej implementácie). V kontexte ray tracingu je toto ideálnym zdrojom pre zrýchlenie a v kombinácii s paralelným spracovaním je ideálne pre real-time nasadanie. Zjednodušená schéma architektúry SIMD je na obrázku 2.5

Architektúra SIMD má pre počítačovú grafiku a špecificky pre zobrazovanie v reálnom čase ray-tracing-om na CPU významné uplatnenie. Na tému využitia SIMD v realistickom zobrazovaní bolo vypracovaných viacero prác [4] resp. [25]. Ako možno postrehnúť architektúra je použiteľná nielen na riešenie primárnych lúčov ale po správnej úprave dát a



Obrázek 2.6: Schéma systému s použitím koherencie zväzku lúčov podľa diagramu v [25]

zapracovaniu predpokladov o koherencii do výpočtov je možná akcelerácia aj u pokročilejších metód zobrazovania. Podľa experimentov v [4] je možné dosiahnuť oproti ne-SIMD zrýchlenie výpočtu priesečníka primárnych lúčov resp. výpočtu osvetlenia až 200 - 300 %. Pre prúdové spracovanie musíme však prijať niekoľko programových opatrení pre zaručenie `koherentnosti` jednotlivých vstupov, v našom prípade lúčov. Aby bolo možné použiť prúdové spracovanie musíme o zväzku lúčov predpokladať že je koherentný. Všetky lúče v ňom majú rovnaký počiatok a väčšia časť priesečníkov leží v rovnakom liste akceleračnej štruktúry. V [4] je tento proces rozdelený do dvoch fáz, v prvej sa vylúčia lúče ktoré nemajú žiadne detekované priesečníky a v druhej sa vyhovujúce lúče zoradia podľa vzájomnej koherencie. Koherencia lúčov nám umožňuje predpokladať, že zväzok a jednotlivé jeho prvky budú pozitívne testované na priesečník s rovnakými objektami v scéne. Pre tento účel je vytvorená cache zväzku, v ktorej je uložený detekovaný priesečník predošlého lúča vo zväzku. V ďalšom kroku sa pristupuje do tejto pomocnej štruktúry, čím sa ušetrí prehľadávanie stromu. Pokial by prístup do cache nedetekoval priesečník je použitý kompletný výpočet a následná update cache. Experimenty naznačujú, že takýto postup je optimálnejší a miera zrýchlenia je konštantná aj pri členitých scénach s nerovnomerne rozmiestnenými primitívami [4]. Systém využívajúci tento postup možno vidieť na obrázku 2.6 , ktorý je prevzatý z [25]. V spomínanej práci je rozpracovaný model prístupu do pamäte nielen pomocou cache lúčov, ale aj pomocou prechodného cachovania geometrie pri rozsiahlych scénach.

2.5 Paralelný ray-tracing

Napriek využitiu vyspelých akceleračných štruktúr a architektúry SIMD je algoritmus ray-tracingu stále výpočtovo náročný, preto je vhodné hľadať ďalšie spôsoby urýchlenia. Ako práce [5], [8] resp. [21] naznačujú, riešením je využitie paralelného spracovania algoritmu ray-tracingu, ktorý je pri svojej povahe vhodným kandidátom na takýto postup, čo jednoznačne potvrdzujú výsledky implementačnných experimentov.

Podľa [21] je možné implementácie využívajúce paralelný ray-tracing rozdeliť na základe oblasti, ktorú daný výpočet využíva. Dátovo orientované paralelné implementácie sú založené na rozdelení dátovej akceleračnej štruktúry na viacero procesorov. Využivajú

vzájomnú komunikáciu pri výpočte priesečníkov lúčov s primitívami scény. Pri tomto postupe sa využíva zloženie a topológia dátovej štruktúry. Výpočet je paralelný a každá výpočtová jednotka je zodpovedná za určitý priestor daný časťou akceleračnej dátovej štruktúry. Urýchlenie výpočtu sa v tomto prípade dosahuje paralelným prehľadávaním podpriestorov daných uzlami dátovej štruktúry, ktoré su pridelené jednotlivým výpočtovým jednotkám. Výhody dátového paralelizmu možno vidieť v znížení pamäťovej náročnosti prerozdelením dát na jednotlivé výpočtové uzly systému. Nevýhodou tohto prístupu je podľa [21] distribuovanie jednotlivých častí výpočtu na výpočtové uzly systému a možná nevyvážená záťaž jednotlivých častí systému v určitých scénach (diskutované v 2.1.1, 2.1.2, 2.1.3, 2.2, 2.3). Dátový paralelizmus v prípade ray tracingu by vyžadoval inicializáciu spojenú s prenosom častí jednotlivých uzlov štruktúry na jednotlivé výpočtové uzly systému. V prípade detekcie priesečníka s primitívom obsiahnutým v jednom z uzlov dátovej štruktúry by príslušný výpočtový uzol zaslal výsledok výpočtu pre ďalšie spracovanie (napr. akumuláciu výsledkov a výsledné vygenerovanie obrazu). Takýto prístup je podľa [8] neefektívny z dôvodu veľkého objemu komunikácie medzi uzlami. Preto možno vzájomnú komunikáciu dátových uzlov spolu s rovnomerným rozdelením výpočtu označiť za úzke hrdlo paralelného výpočtu založeného na dátach scény.

Na druhej strane paralelná implementácia zameraná na rozdelenie daného objemu pixelov na jednotlivé výpočtové uzly je z hľadiska nevýhod spojených s predošlou metódou lepším riešením. Implementácie založené na rozdelení obrazu a oddelenom spracovávaní jednotlivých lúčov sú rozpracované v [5] resp. [21]. V ich implementácii je využitý princíp dynamického prideľovania jednotlivých lúčov výpočtovým uzlom na základe viacerých kritérii. Opäť je úzkym hrdlom celého paralelného výpočtu vzájomná komunikácia medzi jednotlivými výpočtovými uzlami. V [21] je implementácia úzko hardwarovo špecifická a využíva zdielanú pamäť ako prostriedok komunikácie medzi procesormi, avšak niektoré postupy sú pre mna z hľadiska mojej implementácie podstatné. Zaujímavé je heterogénne využitie procesorov v tejto architektúre. Pre paralelné výpočty sú určené 4 procesory, ktorým jednotlivé časti obrazu prideľuje hlavný procesor. Pre komunikáciu tu slúži zdielaná pamäť resp. virtuálna zdielaná pamäť v prípade medziprocesorovej komunikácie. Ako vyplýva z výsledkov experimentov v [21], prišlo k nárastu výkonu ray-tracera použitím paralelného výpočtu dynamickým prideľovaním pixelov obrazu na 303 - 389 % pôvodného výkonu využitím 4 výpočtových uzlov. Čo svedčí o takmer lineárnom náraste výkonu vzhľadom na počet výpočtových uzlov. Možné zlepšenie tohto postupu by som videl v zmene spôsobu pridelovania jednotlivých výpočtov uzlom, nakoľko v mojej implementácii budem komunikáciu realizovať v iných podmienkach s iným komunikačným médiom ako v [21], bude pre mňa prvoradé obmedziť objem komunikácie medzi jednotlivými výpočtovými uzlami. Taktiež nepredpokladám dramatický nárast výkonu pri aplikácii primárnych lúčov, preto budem uvažovať aj o implementácii vyššej metódy zobrazenia, využívajúcej ray-tracing.

Kapitola 3

Vyššie metódy zobrazenia

V tejto kapitole nadviažem na základné princípy ray-tracingu a sústredím sa na metódy výpočtu globálneho osvetlenia. Budem diskutovať vhodnosť jednotlivých postupov s ohľadom na vhodnosť pre zobrazovanie v reálnom čase, ako aj paralelizáciu spomenutých algoritmov.

3.1 BRDF

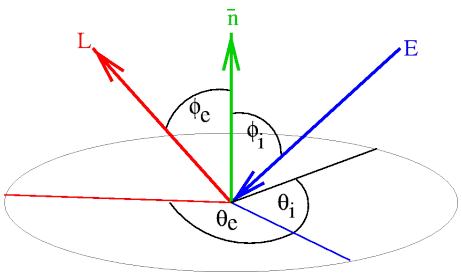
Bidirectional reflectance distribution function je dvojsmerná odrazová distribučná funkcia. Je dôležitým prvkom popisu povrchu objektov scény. Proces vykreslovania objektov v scéne s objektami a svetelnými zdrojmi vyžaduje pre algoritmizáciu tejto úlohy určitý formálny zápis vlastností povrchov objektov. BRDF je definovaná nasledovne:

$$f_r(x, \Psi \longrightarrow \Theta) = \frac{dL(x \longrightarrow \Theta)}{dE(x \longleftarrow \Psi)} \tag{3.1}$$

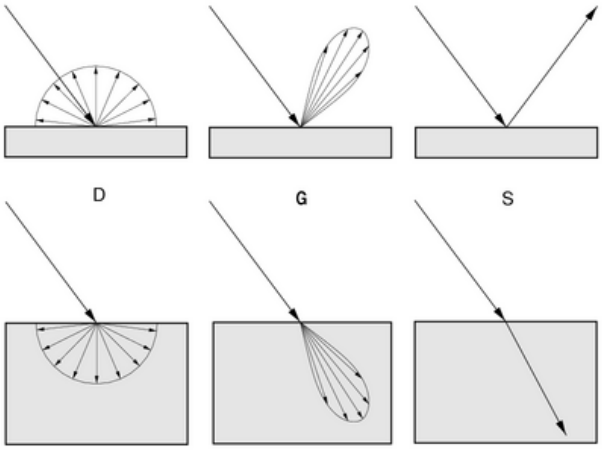
funkcia v bode povrchu x je daná pomerom diferenciálnej žiare odrazenej povrchom v bode x v smere Θ a diferenciálneho ožiarenia v bode x zo smeru Ψ. Udáva nám veľkosť časti svetelnej energie pricházajúcej zo smeru Ψ, ktorá sa odrazí do smeru Θ, ako možno vidieť na obr 3.1. Funkciu možno zapísať aj vo sférickom tvare, kde je do úvahy vzatý skalárny súčin normály povrchu v bode x N_x, takto možno prepísať BRDF do tvaru:

$$f_r(x, \Psi \longrightarrow \Theta) = \frac{dL(x \longrightarrow \Theta)}{E(x \longleftarrow \Psi) cos(N_x, \Psi) d\omega_\Psi} \tag{3.2}$$

Tento zápis vyplýva z geometrie sféry obklopujúcej daný bod a priestorového uhla, cez ktorý prichádza energia do bodu x. BRDF je definovaná cez celú sféru obklopujúcu daný bod. Možno ňou definovať aj zložité povrchy, refrakciu na rozhraní 2 prostredí a ďalšie



Obrázek 3.1: Incidenčné ožiarenie E resp. odrazená žiar L od povrchu, prevzaté z [10]



Obrázek 3.2: Modely povrchov (zľava zhora): difúzny glossy spekulárny odraz, difúzny glosy spekulárny lom, prevzaté z [1]

vlastnosti povrchu v interakcii so svetlom. Vlastnosti niektorých povrchov však nemožno jednoducho vyjadriť analyticky. Takéto materiály sú vyjadrené tabuľkou, ktorá pre dané parametre obsahuje pomer medzi prijatým a odrazeným svetlom. Pre zaobstaranie tohto typu údajov je potrebný špeciálny prístroj Gonioreflektometer. Tento prístroj obsahuje kalibrovaný zdroj svetla a kalibrovanú kameru, ktorá sníma bod pri rôznych uhloch zdroja svetla resp. pri zmene vlastnej polohy. Alternatívne možno BRDF zmerať pomocou reflektometra a postupu ktorý navrhol Ward. V tomto prípade sa cez polopriesvitnú reflexnú hemisféru prepúšťa lúč svetla, ktorý dopadá na objekt v ohnisku hemisféry. Odraz vo všetkých bodoch je možné zosnímať naraz. Viac o spôsobe obstarávania BDRF v [32].

V kontexte počítačovej grafiky však existujú empirické modely povrchov, ktoré možno vyjadriť analyticky. Vo všeobecnosti rozoznávame dokonale difúzne povrchy, rýdzo spekulárne povrchy a kombinácie týchto povrchov. Pre definíciu difúzneho povrchu možno použiť vzťah:

$$f_r(x, \Psi \longrightarrow \Theta) = \frac{\rho_d}{\pi} \tag{3.3}$$

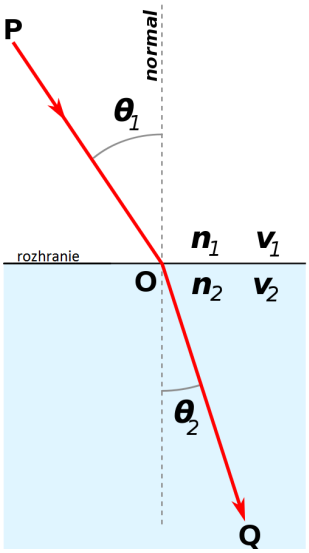
,kde odrazivosť povrchu je daná faktorom difúzneho odrazu daného povrchu ρ_d. Difúzny povrch ako možno vidieť z rovnice 3.3 je pri pozorovaní z akéhokoľvek smeru Θ konštantný pri konštantných podmienkach osvetlenia a polohy skúmaného bodu, schéma v obr. 3.2 vľavo hore. Ďalším modelom povrchu je dokonalý spekulárny povrch, ktorého BDRF definuje odrazy na základe Snellovho zákona. Dokonalý spekulárny odraz vektora Ψ možno zapísať ako:

$$2(N \cdot \Psi)N - \Psi \tag{3.4}$$

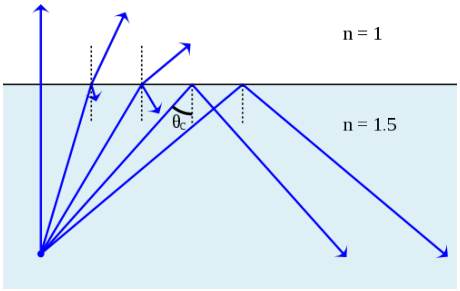
BDRF je potom definovaná nasledovne:

$$f_r(x, \Psi \longrightarrow \Theta) = \begin{cases} 1 & \text{,ak } \Theta = 2(N \cdot \Psi)N - \Psi \\ 0 & \text{,inak} \end{cases} \tag{3.5}$$

Zo vzťahu vyplýva, že energia z daného smeru Ψ je odrazená od povrchu v úzkom zväzku lúčov, vzniká dokonalý odraz, schéma v obr. 3.2. Pri transparentných objektoch však musíme uvažovať o okolitom médiu, z ktorého prichádza svetelná energia a o materiále daného transparentného telesa. Na základe Snellovho zákona prichádza pri chode svetelného



Obrázek 3.3: Schéma chodu lúčov pri lome, prevzaté z [41]



Obrázek 3.4: Schéma chodu lúčov pri lome, prevzaté z [41]

lúča rozhraním prostredí s rôznou hustotou k lomu. Tento jav je spôsobený zmenou fázovej rýchlosti svetelnej vlny pri prechode do prostredia s inou hustotou. Predpokladajme teraz, že lúč sa pohybuje v prostredí medzi objektami rýchlosťou c - rýchlosťou svetla. Pri prechode do objektu sa jeho rýchlosť zmení, označme ju v_p, môžeme takto vyjadriť pomer rýchlostí - index lomu.

$$\eta = \frac{c}{v_p} \tag{3.6}$$

Zo Snellovho zákona ďalej vyplýva, že pre daný incidenčný lúč Ψ a index lomu prostredia η_1, z ktorého pod uhlom θ_1 prichádza platí:

$$\eta_1 \sin(\theta_1) = \eta_2 \sin(\theta_2) \tag{3.7}$$

kde θ_1 resp. θ_2 je uhol medzi medzi normálou N a Ψ resp. Θ v obr. 3.3 , rovnicu možno zapísať vo vektorovom tvare:

$$\Theta = -\frac{\eta_1}{\eta_2}\Psi + N(\frac{\eta_1}{\eta_2}(N \cdot \Theta) - \sqrt{1 - \frac{\eta_1}{\eta_2}^2(1 - (N \cdot \Psi)^2)}) \tag{3.8}$$

výraz nieje definovaný pre $1 - \frac{\eta_1}{\eta_2}^2(1 - (N \cdot \Psi)^2)) < 0$, v obr. 3.4, v tomto prípade sa jedná o tzv. totálny interný odraz, keď incidenčný lúč vstupuje z hustejšieho prostredia do redšieho prostredia pod medzným uhlom, odrazený lúč sa odráža späť do telesa. Tento uhol sa nazýva Brewsterov uhol.

Povaha BRDF skutočných materiálov je však komplexná, analytické vyjadrenie častokrát neexistuje a jedinou možnosťou je použitie merania a aproximácia, počítačová grafika preto pri výpočtoch častokrát pracuje s priblíženiami skutočných materiálov, s osvetľovacími modelmi. BRDF týchto modelov sú dané analytickým vzťahom. Difúzne povrchy môžeme aproximovať Lambertovým modelom, pre ktorý platí vzťah:

$$f_r(x, \Psi \longrightarrow \Theta) = k_d = \frac{\rho_d}{\pi} \tag{3.9}$$

,kde ρ_d je difúzna odrazivosť povrchu, známym zdrojom hardwarových implementácii je Phongov model, tento model obsahuje difúznu(Lambertovskú zložku) a spekulárnu zložku. Vzťah pre Phongov model:

$$f_r(x, \Psi \longrightarrow \Theta) = k_d + k_s \frac{(R \cdot \Theta)^n}{N \cdot \Psi} \tag{3.10}$$

spekulárna časť výrazu obsahuje skalárny súčin vektora R, daného vzťahom 3.5 a incidenčného vektora. Hodnota spekulárnej zložky je teda závislá od uhla pozorovateľa a zdroja svetelnej energie, schéma v obr. 3.2 hore v strede. Spekulárny exponent n určuje strmosť poklesu reflektivity pri odklone incidenčného lúča od vektoru dokonalého odrazu. Vizuálne možno pozorovať veľkosť spekulárneho odlesku na povrchu objektu renderovaného Phongovým modelom. Dôležitým faktorom je aj vzájomná poloha incidenčného vektoru Ψ a normály N, ktorý simuluje Frenelov odraz, ktrého intenzita závisí od uhlu pozorovania bodu na povrchu. Alternatívou k Phongovmu osvetlovaciemu modelu tvorí Blinn-Phong model. Tento využíva tzv. half-vector, os súmernosti vektorov Ψ a Θ, platí vzťah:

$$f_r(x, \Psi \longrightarrow \Theta) = k_d + k_s \frac{(N \cdot H)^n}{N \cdot \Psi} \tag{3.11}$$

Pre dosiahnutie realistických výstupov renderovacích algoritmov však musíme brať do úvahy fyzikálne vlastnosti svetla. Pri zostavovaní BRDF materiálu je dôležité si uvedomiť, že odhliadnuc od vlastnej emisie svetla materiálom musí spĺňať zákon zachovania energie, preto $f_r(x, \Psi \longrightarrow \Theta) \leq 1$. Pre výpočet globálneho osvetlenia je taktiež dôležitá reciprocita BRDF vzhľadom na Ψ a Θ, ktorá hovorí o tom, že hodnota funkcie zostane zachovaná pokiaľ zameníme Ψ a Θ. Toto možno efektívne využiť ako pri metódach vychádzajúcich od pozorovateľa tak aj pri metódach vychádzajúcich od svetelných zdrojov.

3.2 Renderovacia rovnica

Renderovacia rovnica je integrálna rovnica ,ktorej riešením je stabilný stav distribúcie svetelnej energie v scéne. V pôvodnom stave bola predstavená v roku 1986 [17] a generalizovala spektrum renderovacích algoritmov do uceleného vzťahu. Pôvodný tvar rovnice prezentovaný J.T. Kajiyom podľa [17]:

$$I(x, x') = g(x, x')\left[\epsilon(x, x') + \int_S \rho(x, x', x'')I(x', x'')dx''\right] \tag{3.12}$$

, kde $I(x, x')$ sa vzťahuje k prenosu svetelnej energie z bodu x' do bodu x, g(x,x') je geometrický člen, $\epsilon(x, x')$ sa vzťahuje emitovanej svetelnej energii z bodu x' do bodu x a

$\rho(x, x', x'')$ sa vzťahuje k energii prenesenej z bodu x'' do bodu x cez bod x'. Rovnica v tomto tvare hovorí, o transporte svetelnej energie z bodu x' scény do iného bodu x scény ako o súčte emitovanej energie bodu x' a energie prijatej so všetkých okolitých bodov x''. Integrál na pravej strane je definovaný nad doménou S, čo je povrch všetkých objektov v scéne. Geometrický člen možno odvodiť od vzájomnej polohy dvoch bodov v priestore a možno ho vyjadriť vzťahom:

$$g(x, x') = \begin{cases} \frac{1}{r^2} & \text{, ak } x \text{ a } x'' \text{ sú vzájomne viditeľné} \\ 0 & \text{, inak} \end{cases} \tag{3.13}$$

Transport emitovanej svetelnej energie z bodu x' do bodu x predstavuje člen $\epsilon(x, x')$, ktorý nám hovorí o transporte emitovanej žiare. Ožiarenie povrchu v bode x možno potom na základe definície ožiarenia podľa [17] vyjadriť ako :

$$dE = g(x, x')\epsilon(x, x')dtdxdx' \tag{3.14}$$

, kde diferenciálne ožiarenie dané integrálom plochy zdroja (x') a plochy (x) cieľa. Pre vyhodnotenie prenosu svetelnej energie z bodu x" cez bod x' do bodu x je potrebné opätovne vyhodnotit renderovaciu rovnicu pre transfer energie z bodu x" do bodu x' a prenosovú funkciu $\rho(x, x', x'')$, čím sa dostávame k opätovnému riešeniu gemetrického člena a transportu svetelnej energie z bodu x" do bodu x'. Definujme smer od povrchu x k povrchu x' ako Θ, opačný Θ'. Označme uhol medzi normálou v bode x povrchu a smerom Θ θ, uhol medzi normálou v bode x' povrchu a smerom Θ' θ'. Vzdialenosť bodu x k x' r. Prenosová funkcia $\rho(x, x', x'')$ je podľa [17] daná vzťahom:

$$\rho(x, x', x'') = \rho(\theta', \phi', \psi', \sigma') \cos\theta \cos\theta' \tag{3.15}$$

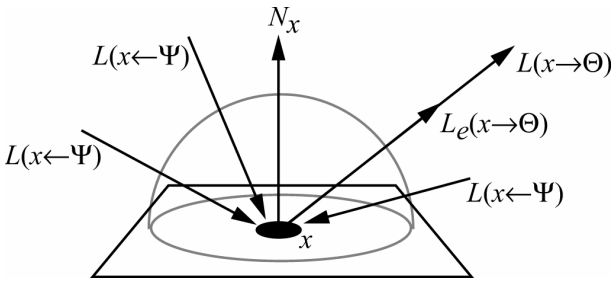
,kde σ' je uhol medzi rovinou odrazeného luča a normály povrchu v bode x' od povrchu x' k povrchu x. Renderovacia rovnica sumarizuje a formalizuje postup pri výpočte v renderovacích algoritmoch. Možno ju rozdeliť na časť emitovanej energie a časť odrazenej energie. Od pôvodnej verzie, ktorá je definovaná ako integrál cez všetky body scény môžeme renderovaciu rovnicu podľa [9] previesť z tvaru integrujúceho povrchy do tvaru integrujúceho hemisféru. Pre túto rovnicu si dovolím zmeniť notáciu. Skúmame bod povrchu, označme ho x, povrch v tomto bode je ožiarený transferom energie z objektov ktoré ho obklopujú. Tvar tejto rovnice môžeme zapísať ako:

$$L(x \longrightarrow \Theta) = L_e(x \longrightarrow \Theta) + L_r(x \longrightarrow \Theta) \tag{3.16}$$

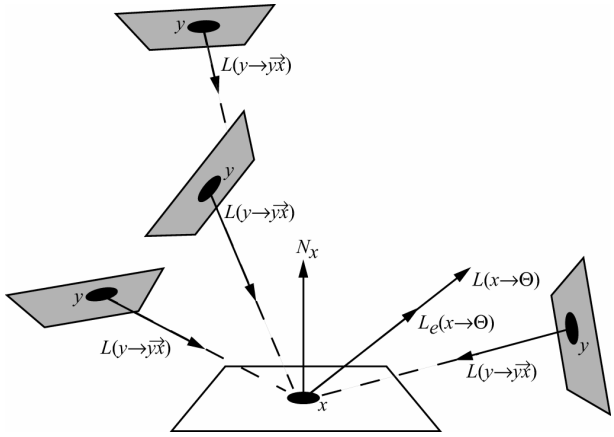
Rovnicu interpretujeme tak, že celková žiar opúšťajúca bod x v oblasti danej priestorovým uhlom Θ je daná vlastnou emisiou v tomto smere a priestorovom uhle a odrazenou žiarou v tomto smere a priestorovom uhle. Pre zahrnutie vlastností povrchu vychádzajme teraz zo vzťahu 3.1, ktorý dosadíme do rovnice 3.16 , takto dostávame vzťah pre renderovaciu rovnicu definovanú integrálom všetkých diferenciálnych priestorových uhlov na hemisfére obklopujúcej bod x :

$$L(x \longrightarrow \Theta) = L_e(x \longrightarrow \Theta) + \int_{\Omega_x} f_r(x, \Psi \longrightarrow \Theta) L(x \longleftarrow \Psi) \cos(N_x, \Psi) d_{\omega_\psi} \tag{3.17}$$

, tento tvar rovnice nám hovorí, že celková žiara prichádzajúca z bodu x priestorovým uhlom Θ je rovná súčtu emisii žiare v bode x do priestorového uhla Θ a integrálu žiare



Obrázek 3.5: Integrácia hemisféry v renderovacej rovnici, podľa [9]



Obrázek 3.6: Integrovanie plôch v renderovacej rovnici, podľa [9]

prichádzajúcej do bodu x z priestorového uhla Ψ, schéma v obr. 3.5. Dôležité je poznamenať, že integrál je definovaný cez celú sféru obklopujúcu bod x, pri modifikácii BRDF pre integrovanie celej sféry možno definovať BSSRDF, čo je dvojsmerná podpovrchová rozptylová funkcia. Túto možno využiť pri simulovaní priesvitných materiálov.

Pre výpočet osvetlenia jednotlivých bodov scény svetelnými zdrojmi je praktické redefinovať si pôvodnú zobrazovaciu rovnicu do tvaru použiteľného pre integrovanie povrchov. Tento postup je nutný pre správny návrh algoritmu výpočtu osvetlenia daného bodu v scéne, pri použití plošných zdrojov svetla. V prvom kroku definujme funkciu podobnú geometrickému členu s pôvodnej rovnice Kajiyu 3.13. Definujme dva body x a y v scéne. Funkcia $V(x, y)$ určuje pre všetky body priestoru vzájomnú viditelnost a nadobúda hodnoty 0,1 definovaná vzťahom:

$$V(x,y) = \begin{cases} 1 & \text{, ak } x \text{ a } y \text{ sú vzájomne viditeľné} \\ 0 & \text{, inak} \end{cases} \tag{3.18}$$

V kontexte emisie žiare z bodu y do bodu x možno funkciu interpretovať ako umiestnenie bodu x v tieni y. Ďalším faktorom ovplyvňujúcim množstvo žiare dopadajúcej na povrch x z bodu y je vzájomná poloha bodov. Podľa [9] je geometrický člen G(x,y) daný relatívnou geometriou povrchov v bodoch x a y, možno ho vyjadriť vzťahom:

$$G(x,y) = \frac{\cos(N_x, \Psi)\cos(N_y, \Psi)}{r_{xy}^2} \tag{3.19}$$

Renderovacia rovnicu potom možno prepísať do tvaru podobného pôvodnému tvaru

rovnice Kajiyu:

$$L(x \longrightarrow \Theta) = L_e(x \longrightarrow \Theta) + \int_A f_r(x, \Psi \longrightarrow \Theta) L(y \longrightarrow -\Psi) V(x,y) G(x,y) dA_y \quad (3.20)$$

Tento tvar nám dáva možnosť výpočtu založenom na diferenciálnych plochách. Zvlášť ideálnym je tento postup pri výpočte osvetlenia povrchu plošnými zdrojmi. Schéma v obr. 3.6

3.3 Monte Carlo

Monte Carlo je metóda numerickej integácie funkcie. Pre vyhodnotenie integrálu funkcií možno použiť aj symbolické integrovanie alebo niektorú inú numerickú napr. obdĺžnikovú metódu. V počítačovej grafike musíme pri riešení globálneho osvetlenia vyhodnocovať viacrozmerné integrály. Nevýhodou použitia obdĺžnikovej metódy je v tomto prípade potreba vyhodnotiť N^d funkčných hodnôt pre d-rozmerný definičný obor. V tomto smere poskytuje metóda Monte Carlo univerzálny nástroj pre numerické integrovanie viac-dimenzionálnych funkcii.

Metóda Monte Carlo v jej čistej forme je podľa [38] založená na náhodnom výbere hodnoty s definičného oboru integrovanej funkcie a jej následného vyhodnotenia, z ktorého odhadujeme numerickú hodnotu integrálu pomocou odhadu:

$$\langle I \rangle = \frac{1}{N} \sum_{n=1}^{N} f(x_n) \quad (3.21)$$

, kde N je počet vyhodnotení funkčnej hodnoty danej funkcie. Na základe teorému zákona velkých čísel, ktorý hovorí, že pri dostatočne velkom počte nezávyslých pokusov sa hodnota početnosti jednotlivých hodnôt bude blížiť teoretickej hodnote pravdepodobnosti, môžeme tvrdiť:

$$\lim_{N \longrightarrow \infty} \frac{1}{N} \sum_{n=1}^{N} f(x_n) = I \quad (3.22)$$

,čo znamená, že pre dostatočne veľký počet vyhodnotených funkčných hodnôt sa odhad $\langle I \rangle$ približuje skutočnej hodnote I. Dôležitým údajom v kontexte tejto metódy je aj odhad chyby $\langle I \rangle$. Definujme si teraz rozptyl σ funkcie $f(x)$:

$$\sigma^2(f) = \int (f(x) - I)^2 dx \quad (3.23)$$

pri použití metódy Monte Carlo možno tento vzťah upraviť do podoby odhadu rozptylu funkcie f(x):

$$\sigma^2(f) = \frac{1}{N-1} \int (f(x) - I)^2 dx \quad (3.24)$$

využitím centrálnej limitnej vety na predošlý výraz, s predpokladom, že dané funkčné hodnoty v odhade sú pri dostatočne veľkom N normálne rozložené so strednou hodnotou hladaného integrálu I:

$$\lim_{N \longrightarrow \infty} P\left(-a\frac{\sigma(f)}{\sqrt{N}} \leq \frac{1}{N}\sum_{n=1}^{N} f(x_n) - I \leq b\frac{\sigma(f)}{\sqrt{N}}\right) = \frac{1}{\sqrt{2\pi}}\int_{-a}^{b} e^{-\frac{t^2}{2}}dt \tag{3.25}$$

Ako možno vidieť zo vzťahu 3.3, pravdepodobnosť, že hodnota odhadu leží v medziach $I - a\frac{\sigma(f)}{\sqrt{N}}$ resp. $I + b\frac{\sigma(f)}{\sqrt{N}}$ sa zvyšuje pri súčasnom zmenšovaní intervalu, čo znamená že hodnota odhadu konverguje k skutočnej hodnote integrálu skúmanej funkcie. Toto tvrdenie možno aplikovať pri dostatočne veľkom N. Pre odhad chyby Metódy Monte Carlo možno podľa [9] použiť Čebyševovu nerovnosť, ktorá hovorí, že pravdepodobnosť že odhad sa odlišuje od riešenia o hodnotu väčšiu ako je $\sqrt{\frac{\sigma^2}{\delta}}$, kde δ kladné číslo menšie ako 1 je menšia ako δ, zapíšeme vzťahom:

$$P\left[|\langle I\rangle - I| \geq \sqrt{\frac{\sigma_I^2}{\delta}}\right] \leq \delta \tag{3.26}$$

Zo vzťahov 3.23 resp. 3.24 teda vyplýva, že štandardná odchylka nášho odhadu klesá proporcionálne k $\frac{1}{\sqrt{N}}$. Rýchlosť konvergencie pôvodnej metódy Monte Carlo, aj keď ide o elegantné riešenie je relatívne pomalá. Existujú však postupy pre zníženie rozptylu odhadu a tým aj pre zrýchlenie konvergencie, čo z praktického hľadiska znamená pri rovnakej kvalite zobrazovanej scény menej vzorkov a tým pádom aj rýchlejšie vykreslený frame.

Efektívnou metódou zníženia variancie je importance sampling. V predošlom texte sme hodnotu z definičného oboru vyberali náhodne z rovnomerného rozdelenia na definičnom obore tak, že pravdepodobnosť zvolenia hodnoty z definičného oboru bola konštantná a nezávisela na funkčnej hodnote funkcie, ktorej integrál sme odhadovali. Importance sampling generuje hodnoty x_i na základe funkcie hustoty pravdepodobnosti. Existuje viacero postupov ako spočítať hustotu pravdepodobnosti definovanú v doméne integrácie funkcie, jedným z nich je technika nájdenia inverznej distribučnej funkcie pravdepodobnosti na základe funkcie ktorej integrál metódou Monte Carlo počítame. Zaveďme teda modifikáciu odhadu metódy Monte Carlo zo vzťahu:

$$\langle I\rangle = \frac{1}{N}\sum_{n=1}^{N}\frac{f(x_i)}{p(x_i)} \tag{3.27}$$

, kde $p(x_i)$ je funkcia hustoty pravdepodobnosti náhodnej veličiny X, teda hodnôt pre ktoré vyhodnocujeme integrál funkcie. Zmyslom metódy importance sampling je distribuovať náhodnú veličinu X na základe distribučnej funkcie pravdepodobnosti ktorú možno odvodiť od hustoty pravdepodobnosti funkcie ktorej integrál hľadáme. Podľa [38] je p(x) vhodné zvoliť na základe tvaru funkcie f(x), ktorej integrál chceme vypočítať. Potom pre vygenerovanie náhodnej veličiny X na základe funkcie hustoty pravdepodobnosti p(x) náhodnej veličiny vypočítame distribučnú funkciu F(x):

$$F(x) = P(X \leq x) = \int_{-\infty}^{X} p(x)dx \tag{3.28}$$

zvoľme náhodnú veličinu u s rovnomerným rozdelením v intervale $[0, 1)$, ďalej zaveďme náhodnú veličinu Y tak, že:

$$P(y \leq Y) = \int_{-\infty}^{Y} p(x)dx \tag{3.29}$$

zároveň platí, že $P[u \leq X] = X$ a zároveň $y = F^{-1}(u)$, potom platí, že $P[F^{-}1(u) \leq F^{-}1(X)] = X$. Vzorky y generujeme na základe $y = F^{-}1(u)$. Keďže tento postup vyžaduje integráciu samotnej funkcie $f(x)$ a následne výpočet $F^{-}1$ je vhodný pre analyticky definované funkcie, v opačnom prípade je možné použiť pre odhad tvaru funkcie predošlé vyhodnotené funkčné hodnoty a aproximovať funkciu hustoty dynamicky počas výpočtu odhadu. Ďalšou metódou znižovania rozptylu odhadu je stratified sampling. Podstatou tejto metódy je rozdelenie domény integrácie na viacero častí. Subdoménu nazývame strata. Stratégia rozdelovania domény je založená na funkcii hustoty pravdepodobnosti. Formálne zapísané:

$$\int_{a_0}^{a_m} f(x)dx = \int_{a_0}^{a_1} f(x)dx + \int_{a_1}^{a_2} f(x)dx + \cdots + \int_{a_{m-1}}^{a_m} f(x)dx \tag{3.30}$$

, kde sme doménu rozdelili na m subdomén v ktorých vyhodnocujeme čiastkový integrál. Rozptyl na základe vzťahu 3.30 a 3.24 možno zapísať ako sumu jednotlivých rozptylov v subdoménach:

$$\sigma^2 = \sum_{j=1}^{m} \frac{a_j - a_{j-1}}{n_j} \int_{a_{j-1}}^{a_j} f(x)^2 dx - \sum_{j=1}^{m} \frac{1}{n_j} (\int_{a_{j-1}}^{a_j} f(x)dx)^2 \tag{3.31}$$

Pri generovaní jednej funkčnej hodnoty na jedno stratum a v rámci neho pomocou rovnomerného rozloženia možno napísať

$$\sigma^2 = \frac{1}{N} \int_{a_0}^{a_m} f(x)^2 dx - \sum_{m=1}^{N} (\int_{a_{m-1}}^{a_j} f(x)dx) \tag{3.32}$$

, ako možno vidieť celkový rozptyl odhadu sa v tomto naivnom prípade (rovnako veľké subdomény bez ohľadu na priebeh funkcie) bude vždy menší ako neinformovaná metóda Monte Carlo. Ďalším predpokladom pre zníženie rozptylu je voľba takého rozdelenia na disjunktné subdomény, aby ich rozptyl bol zhodný resp. podobný. Opačným postupom je generovanie proporcionálneho množstva funkčných hodnôt v subdoméne k veľkosti rozptylu v danej subdoméne. Oba postupy zaručujú, že celkový rozptyl odhadu metódy Monte Carlo bude stabilný pri prispôsobení veľkosti jednotlivých subdomén tvaru funkcie. Podľa [9] je možné stratified sampling použiť v odhade integrálov typicky menších dimenzií. Dôvodom je (pri použití jednéj funkčnej hodnoty na stratum) nutnosť vygenerovať N^d subdomén. Kľúčovou nevýhodou tejto metódy je použitie pri v predstihu neznámom počte vzorkov. Príkladom môže byť adaptívne vzorkovanie, ktoré vzorkuje hodnotu pixelu s premenlivým počtom vzoriek. Tento postup umožňuje urýchliť výpočet obmedzením vzorkovania na potrebné časti výpočtu resp. generovať v danom čase kvalitnejšie výstupy. Adaptívnym vzorkovaním sa zaoberalo viacero prác. V [27] je navrhnutá štatistická metóda, ktorá určuje kritérium pre dostatočný počet vzoriek na pixel na základe intervalov spoľahlivosti. Pixel je vzorkovaný pokiaľ úroveň spoľahlivosti nedosiahne požadovanú hodnotu pre zvolený interval resp. pre zadanú úroveň spolahlivosti klesne interval spolahlivosti pod požadovanú úroveň, čo možno zapísať vzťahom podľa [27] :

$$P\big[L \in [\langle L \rangle - d, \langle L \rangle + d]\big] = 1 - \alpha \tag{3.33}$$

, skutočná hodnota L sa vo zvolenom intervale nachádza s pravdepodobnosťou danou úrovňou spolahlivosti. V [33] je vo vzťahu uvažovaný aj operátor prevádzajúci vypočítanú hodnotu žiare na zobrazovanú hodnotu. Podmienkou ukončenia vzorkovania je tu interval spolahlivosti odvodený od schopnosti zobrazovacieho zariadenia resp. od operátora prevádzajúceho rozsah vypočítaných hodnôt na zobrazované hodnoty. Predpoklad vychádza zo skutočnosti, že metódou Monte Carlo odhadujeme skutočnú hodnotu žiare pixelu, kde stredná hodnota vzorkov konverguje k skutočnej hodnote. Pre väčší počet vzorkov ($n > 30$) možno využiť vzťah , s predpokladom, že vypočítané hodnoty majú normálne rozdelenie so strednou hodnotou skutočnej žiare pixelu. V prípade počtu vzorkov ($n < 30$) nahrádzame normálne rozdelenie studentovým rozdelením. Podľa [33] :

$$P\big[L \in [\langle L\rangle - t_{(n-1),(1-\alpha)}\sqrt{\frac{s^2}{n}}, \langle L\rangle + t_{(n-1),(1-\alpha)}\sqrt{\frac{s^2}{n}}]\big] = 1-\alpha \tag{3.34}$$

, kde $t_{(n-1),(1-\alpha)}$ je hodnota normovaného $(1-\alpha)$ kvantilu studentovho rozdelenia,α je hladina významnosti, s^2 je výberový rozptyl, tento vzťah možno interpretovať tak, že pravdepodobnosť, že skutočná hodnta žiare pixelu sa nachádza v intervale spoľahlivosti danom úrovňou α je $1-\alpha$. Pre stanovenie podmienky dostatočného počtu vzorkov na pixel je potrebné určiť maximálnu prípustnú veľkosť intervalu. Pri renderovaní je odhadnutá hodnota žiare prevedená do rozsahu, ktorý je schopné zobraziť zobrazovacie zariadenie. Podmienku možno teda odvodiť od schopnosti zariadenia zobraziť rozdiel hornej resp. dolnej medze intervalu spolahlivosti (prepočítaného na rozsah zobrazovacieho zariadenia). Spomínaná metóda je vhodná na urýchlenie výpočtu odhadu metódou Monte Carlo a v oblasti zobrazovania v reálnom čase môže pri rovnakej kvalite dodať výstup v kratšom čase ako pevný počet vzorkov na pixel. Ďalšou metódou pre zníženie rozptylu odhadu je Quasi Monte Carlo, ktorá využiva deterministické volenie vzoriek na základe predošlých, pričom cieľom je rozmiestniť daný počet vzorkov tak rovnomerne ako je možné. Najznámejším postupom je použitie sekvencie nízkej diskrepancie. Každá i-tá vzorka je vyberaná tak aby sa dosiahlo rovnomerného rozloženia týchto vzoriek v rámci domény. Sekvencia takýchto vzoriek sa nazýva sekvencia s nízkou diskrepanciou. Bližšie o tomto postupe generovania pojednáva [11].

3.4 Path tracing

Path-tracing je renderovacia technika, ktorá podobne ako ray-tracing sleduje cestu lúča od pozorovateľa smerom k scéne. Na rozdiel od klasického ray-tracingu sa na detekovaných priesečníkoch lúče nevetvia. Podľa [17] je tento postup na rozdiel od ray-tracingu efektívnejší v oblasti pamäťovej náročnosti, čo je pre renderovanie v reálnom čase dôležitým faktorom. V klasickom ray-tracingu je v každom priesečníku spočítaný osvetľovací model na základe rozmiestnenia svetiel a vlastností povrchu. Pri detekovaní priesečníka s objektom môže prísť k lomu resp. odrazu daného lúča a následné sú vygenerované lúče dva so zodpovedajúcim smerom na základe indexu lomu respektíve reflektívnych vlastností povrchu. Tento postup vedie na vetvenie v každom lúči a je z hľadiska zložitých scén neefektivny. Alternatívu v tomto smere poskytuje path-tracing. Algoritmus je založený na generovaní lúča smerujúceho od pozorovateľa každým pixelom obrazu, tento krok vytvorí primárny lúč. V tomto kroku je možné použiť stochastické vzorkovanie plochou pixelu. Výsledkom, pri väčšom počte lúčov na pixel, je vyhľadenie ostrých hrán. Pri detekcii priesečníka lúča s objektom je nutné stanoviť udalosť ,ktorá vygeneruje sekundárny lúč. Podľa povahy povrchu možno

túto udalosť generovať stochasticky proporcionálne k intenzite jednotlivých zložiek modelu povrchu. Príkladom takéhoto výberu udalosti pre primárny lúč môže byť generovanie v scéne s difúznymi a glossy povrchmi. Difúzna zložka BRDF je reprezentovaná difúznym faktorom spomínaným v podkapitole 3.1 vo vzťahu 3.3. Spekulárna zložka je reprezentovaná spekulárnym faktorom vo vzťahu 3.10. Výber udalosti je podľa [23] možné uskutočniť na základe intenzity spomenutých zložiek. Zaveďme u náhodnú veličinu s rovnomerným rozložením v intervale $[0, 1]$, ďalej intenzitu spekulárnej a difúznej zložky normalizujme do intervalu $[0; 0, 5]$. Voľba udalosti prebehne nasledovne:

- ak $u < k_s$, generujeme spekulárny sekundárny lúč
- ak $k_s < u < k_s + k_d$, generujeme difúzny lúč
- ak $k_s + k_d < u$, ukončíme generovanie lúčov

túto funkciu možno použiť v metóde Monte Carlo voľbou udalosti lúča - pravdepodobnosť výskytu spekulárneho resp. difúzneho odrazu je proporcionálna k intenzitám jednotlivých zložiek modelu. Aby však zostala metóda neskreslená je potrebné voľbu lúča zohľadniť v odhade. V tomto prípade použijeme metódu zvanú ruská ruleta. Na základe [9], môžeme odhad metódy Monte Carlo prepísať do podoby 3.35. Výsledná hodnota odhadu Monte Carlo bude váhovaná inverznou hodnotou pravdepodobnosti udalosti, podľa ktorej zvolíme lúč.

$$I_{RR} = \begin{cases} \frac{1}{P} f(\frac{x}{P}) & , akx \leq P \\ 0 & ,\text{inak} \end{cases} \tag{3.35}$$

V tomto štádiu je potrebné vygenerovať smer lúča na základe uhla Θ a BRDF materiálu povrchu objektu. Využitím Metódy Monte Carlo a využitím modifikácie importance sampling možno vygenerovať na základe zostavenia inverznej funkcie k distribučnej funkcii pravdepodobnosti smer lúča pomocou náhodnej premennej s rovnomerným rozložením v intervale $[0, 1]$. V prácach [22] resp. [24] je diskutovaná vhodnosť použitia Phongovho osvetlovacieho modelu ako BRDF pri generovaní lúčov pri odhade nepriameho osvetlenia. Hlavným faktorom nevhodnosti je to, že Phongov osvetľovací model nespĺňa zákon zachovania energie. [24] obsahuje dôkaz, preto použijem vo svojej práci autorom navrhovaný modifikovaný model Phong, ktorý zákon o zachovaní energie spĺňa. BRDF tohto modelu možno vyjadriť na základe [22], použitím notácie mojej práce:

$$f_r(x, \Psi \longrightarrow \Theta) = k_d \frac{1}{\pi} + k_s \frac{n+2}{2\pi} cos^n(\alpha) \tag{3.36}$$

Pre difúzny povrch použijme vzťah pre BRDF Lambertovského modelu 3.9, totožného s modifikovanou verziou 3.36, ďalej využime vzťah renderovacej rovnice 3.16 ,do ktorého dosadíme vzťah BRDF difúzneho povrchu, pričom zanedbáme emisiu žiarenia tohto povrchu. Výsledná RE pre difúzny povrch je potom:

$$L(x \longrightarrow \Theta) = L_r(x \longrightarrow \Theta) = f_r(x, \Psi \longrightarrow \Theta) E(x \longleftarrow \Psi) \cos(N_x, \Psi) d\omega_\Psi \tag{3.37}$$

V tomto odvodenom vzťahu pre vyhodnotenie hodnoty vzorkovaného bodu x (žiar v smere pozorovateľa) je potrebné vygenerovať smer na základe BRDF(f_r) v tomto prípade difúznej zložky, možno tento vzťah rozviesť substitúciou f_r za konkrétnu BRDF modelu:

$$L(x \longrightarrow \Theta) = L_r(x \longrightarrow \Theta) = E(x \longleftarrow \Psi)\frac{1}{\pi}\cos(N_x, \Psi)d\omega_\Psi \tag{3.38}$$

, v tomto tvare jasne vidieť, že je potrebné vyhodnotit smer a ožiarenie incidenčné zo smeru Ψ, tento smer je však potrebné vygenerovať. Ako možno ďalej vidieť na vzťahu. Intenzita odrazeného svetla nieje závislá od azimutu smeru Ψ. Dôležitým prvkom je $\cos(N_x, \Psi)$. V doméne elevácie má teda funkcia nekonštantnú funkciu hustoty pravdepodobnosti, preto prispôsobíme generovanie lúčov v tejto doméne pomocou inverznej funkcie distribúcie pravdepodobnosti. Na základe [9] resp. [23] možno integrovať výraz v dvojrozmernej doméne azimutu a elevácie v hemisfére obklopujúcej bod x, keďže hodnotu člena $E(x \longleftarrow \Psi)$ nepoznáme:

$$L(x \longrightarrow \Theta) = L_r(x \longrightarrow \Theta) = E(x \longleftarrow \Psi)\int_\phi\int_\theta \frac{\cos(N_x, \Psi)}{\pi}d\phi d\theta \tag{3.39}$$

tento vzťah je analyticky integrovateľný a funkcia je saparabilná na domény azimutu a elevácie, takto dostávame jednotlivé funkcie distribúcie pravdepodobnosti pre domény:

$$\begin{aligned} F_\phi &= \tfrac{\phi}{2\pi} \\ F_\theta &= 1 - \cos^2\theta \end{aligned} \tag{3.40}$$

Funkčné hodnoty týchto funkcií sú v intervale $[0, 1]$, invertujeme tieto funkcie a zvolíme náhodné veličiny u_1 u_2 v intervale $[0, 1]$, potom môžeme generovať smery difúznych lúčov v zhode s funkciou hustoty pravdepodobnosti 3.39:

$$\begin{aligned} \phi &= 2\pi u_1 \\ \theta &= \cos^- 1\sqrt{u_2} \end{aligned} \tag{3.41}$$

dôležitým krokom je úprava vzťahu pre odhad metódou Monte Carlo, keďže sme importance sampling založili na vzorkovaní podla funkcie hustoty pravdepodobnosti 3.39 formálne zapísané:

$$p(x)_{diffuse} = \frac{\cos(N_x, \Psi)}{\pi} \tag{3.42}$$

dosadením do vzťahu pre odhad metódy Monte Carlo a renderovacej rovnice je odhad žiare daného bodu pre difúzne lúče:

$$\langle L(x \longrightarrow \Theta)_{diffuse}\rangle = \frac{1}{N}\sum_{n=1}^{N}\frac{\frac{\cos(N_x,\Psi)}{\pi}}{\frac{\cos(N_x,\Psi)}{\pi}}E(x \longleftarrow \Psi_n) = k_d\sum_{n=1}^{N}E(x \longleftarrow \Psi_n) \tag{3.43}$$

Ako možno vidieť, lúče sme generovali na základe 3.42, preto pri dosadení do odhadu metódy Monte Carlo je pre odhad dôležitý koeficient difúzneho odrazu a samotná žiara incidentná zo smeru Ψ_n, čo je vygenerovaný smer n-tého lúča. Generovaný azimut resp. elevácia je v tzv. tangent space. Tento priestor je vztiahnutý k normále v skúmanom bode x. Vygenerované sférické súradnice prevedieme do kartesianskych na základe vzťahov:

$$\begin{aligned} x &= \sin\theta\cos\phi \\ y &= \sin\theta\sin\phi \\ z &= \cos\theta \end{aligned} \tag{3.44}$$

následne jednotlivé lúče prevedieme do objektového priestoru. Tento postup vygeneruje difúzny lúč z bodu x. Pre generovanie spekulárneho lúča použijeme tangent space vztiahnutý k reflekčnému vektoru vektora Θ popísaného vzťahom 3.4. Uvažujme teraz zobrazovaciu rovnicu spekulárneho povrchu bez emisie:

$$\begin{aligned} L(x \longrightarrow \Theta) &= L_r(x \longrightarrow \Theta) = E(x \longleftarrow \Psi) f_{r_{specular}}(x, \Psi \longrightarrow \Theta) \cos(N_x, \Psi) d\omega_\Psi \\ L(x \longrightarrow \Theta) &= L_r(x \longrightarrow \Theta) = E(x \longleftarrow \Psi) \tfrac{n+2}{2\pi} \cos^n(\alpha) \cos(N_x, \Psi) d\omega_\Psi \end{aligned} \tag{3.45}$$

Použime, na základe [22] modifikovanú časť spekulárneho odrazu a generujme lúče na základe funkcie hustoty pravdepodobnosti:

$$p(x)_{specular} = \frac{n+1}{2\pi} cos^n(\alpha) \tag{3.46}$$

, následne teraz vytvoríme distribučnú funkciu a invertujeme:

$$\begin{aligned} &L(x \longrightarrow \Theta) = L_r(x \longrightarrow \Theta) = \textstyle\int_\phi \int_\theta (\frac{n+1}{2\pi} \cos^n \alpha) d\phi_\alpha d\theta_\alpha \\ &\phi_\alpha = 2\pi u_1 \\ &\theta_\alpha = \cos^- 1(u_2^{\frac{1}{n+1}}) \end{aligned} \tag{3.47}$$

za predpokladu, že u_1 a u_2 sú rovnomerne distribuované v intervale $[0, 1]$, generujeme spekulárne lúče. Obdobne ako vo vzťahu 3.43 určíme na základe 3.46 odhad odrazenej žiare bodom x metódou Monte Carlo, formálne:

$$\begin{aligned} \langle L(x \longrightarrow \Theta)_{specular} \rangle &= k_s \frac{1}{N} \sum_{i=1}^{N} E(x \longleftarrow \Psi_i) \frac{f(x_n)}{p(x_n)} \cos(N_x, \Psi_i) \\ &= k_s \frac{1}{N} \sum_{i=1}^{N} E(x \longleftarrow \Psi_i) \frac{\frac{n+2}{2\pi} cos^n(\alpha)}{\frac{n+1}{2\pi} cos^n(\alpha)} \cos(N_x, \Psi_i) \\ &= k_s \sum_{i=1}^{N} E(x \longleftarrow \Psi_i) \frac{n+2}{n+1} \cos(N_x, \Psi_i) \end{aligned} \tag{3.48}$$

, kde vektor Ψ je prevedený do objektového priestoru, avšak v tomto prípade je generovaný lúč na základe 3.47 vztiahnutý k odrazovému vektoru vektora Θ. V prípade difúznych lúčov konštruujeme bázové vektory tangent space napríklad pomocou vektorového súčinu normály a náhodného jednotkového vektora. Produkt následne vektorovo násobíme opätovne s normálou, formálne zapísané:

$$\begin{aligned} T &= N_x \times V_{rand} \\ B &= N_x \times T \end{aligned} \tag{3.49}$$

, dostávame bázu tangent space v objektovom priestore, z ktorej lineárnou kombináciou zložiek Ψ a vektora v objektovom priestore zo vzťahu 3.44 získame smer lúča. Pre spekulárne lúče bázu vytvoríme pre odrazový vektor α. Algoritmus teda v bode x spočíta osvetľovací model, vyhodnotí udalosť a vyšle sekundárny lúč z bodu x do scény. Opäť vyhodnotí prítomnosť priesečníka, osvetľovací model v danom bode, udalosť a vyšle svetelný lúč. Tento postup môžeme formalizovať rozvojom zobrazovacej rovnice pre sekundárne tercialne a ďalšie lúče. Metódu Monte Carlo v tomto prípade využívame vrhaním viacerých lúčov plochou pixelu a váhovaním. Vyššie predstavený postup umožňuje čiastočne eliminovať rozptyl odhadu a zrýchliť konvergenciu výpočtu path-tracingu. Alternatívou k tejto metóde výpočtu globálneho osvetlenia je metóda photon-mappingu.

3.5 Photon-mapping

Photon mapping je alternatívou pre výpočet globálneho osvetlenia scény. Ide o dvojprechodový algoritmus, ktorého prvá časť simuluje tok fotónov scénou a ich interakciu s povrchmi objektov scény nazývaná shooting a druhá časť ktorá zo zaznamenaných dráh fotónov odhaduje výslednú žiaru jednotlivých viditeľných bodov scény.

Prvou fázou photon mappingu je emisia fotónov svetelnými zdrojmi v scéne. Podľa [16] by mal byť počet fotónov emitovaných jednotlivými svetelnými zdrojmi proporcionálny k ich žiarivému výkonu. Pre smer fotónov a body emisie vzniklo viacero modelov svetiel. Difúzne bodové svetlo je charakterizované emisiou fotónov s rovnomerným rozložením smerov fotónov na hemisfére obklopujúcej zdroj svetla. Plošné zdroje svetla možno simulovať emisiou fotónov v smeroch s početnosťou proporcionálnou ku kosínu uhla, ktorý zviera smer a normála povrchu emitora. Vyžiarené fotóny sú podrobené testom na priesečníky s objektami scény. V prípade incidentu s povrchom je vyhodnotenie udalosti závislé na povahe povrchu objektu. Podobne ako u path-tracingu je udalosť vyberaná na základe vlastností povrchu. Pre rýdzo spekulárne povrchy je možné použiť vzťah 3.4, pre difúzny odraz možno použiť BRDF Lambertovského povrchu 3.3, takto možno určiť nové smery fotónov a ich intenzity, ktoré odvodíme z príslušnej BRDF pre daný povrch. Pravdepodobnosti jednotlivých udalostí možno simulovať postupom nazývaným ruská ruleta. Princíp som objasnil v 3.4. Podstatou je vybrať udalosť pri detekcii priesečníka dráhy fotónu na základe pravdepodobnosti jednotlivých udalostí ,ktorú je možno odvodiť od vlastností materálu objektu kde udalosť nastane. V tomto postupe je praktické uvažovať intenzitu fotónu a zahrnúť ju do pravdepodobnosti jednotlivých udalostí. Každá zrážka fotónu je uložená v priestorovej štruktúre - fotónovej mape. Táto uchováva jednotlivé polohy udalostí interakcie fotónov s povrchmi, incidenčné smery a generované smery resp. intenzitu fotónov. Ideálnou štruktúrou takejto mapy je podľa [16] kd-strom diskutovaný v 2.1.2. Algoritmus v druhej fáze z rozmiestnenia fotónových udalostí v mape odhaduje žiar opúštajúcu bod v smere obrazu. Tento krok je nazývaný gathering. Vo všeobecnosti je možné použiť dve metódy. Jednou z nich je odhad hustoty jednotlivých fotónov v okolí skúmaného bodu. Incidenčné smery týchto fotónov sú aproximované na polohu bodu ktorého žiar je nutné vypočítať. Výslednou interpoláciou ožiarenia bodu a incidenčných smerov získame celkové ožiarenie bodu. Následne je podľa BRDF daného materálu vyhodnotená žiar bodu v smere odrazu a výsledok je zobrazený. Pri tejto metóde možno na spresnenie použiť aj metódu path-tracingu, sresňujúci výsledok tak dostávame predĺžením primárneho lúča na základe BRDF do scény, kde v každom bode priesečníka aproximujeme ožiarenie bodu. Tento postup sa využíva pre zrýchlenie konvergencie metódy. Druhou možnosťou je podľa [16] uvažovanie o uložených parametroch fotónov ako o samostatných zdrojoch osvetlenia. Výhodou photon-mappingu je schopnosť vytvárať špecifické fotónové mapy nazývané kaustické. Kaustické mapy zachytávajú fotóny prechádzajúce transparentnými objektami s dobre definovanými vlastnosťami ako je index lomu, priepustnosť, či odrazivosť povrchu. Kaustický vzor vytvorený za telesom osvetleným svetelným zdrojom je spôsobený koncentráciou svetelných lúčov v malom priestore. Naviac je možné v scénach s menším počtom objektov vytvárajúcich takéto efekty selektívne určiť smery fotónov pre dosiahnutie rýchlejšej konvergencie k výslednej hodnote jednotlivých renderovaných bodov. Výhodou renderovania photon-mappingom je nesporne pomer kvality výstupu a rýchlosti. Na druhej strane je v dynamických scénach potreba pri zmene polohy a orientácie svetiel vytvoriť novú fotónovú mapu, práve tento krok je pri snahe získať kvalitný výsledok potrebné vytvoriť mapu rádovo 10^6 fotónmi. Ďalšou otázkou je vhodnosť paralelizácie, ktorou sa zaoberám v ďalšej sekcii.

3.6 Paralelizácia výpočtu globálneho osvetlenia

Metódy riešenia globálneho osvetlenia scény sú generalizáciou algoritmov ray-tracingu, preto možno postupy paralelizácie aplikované na ray-tracing použiť aj tu. V 2.5 som načrtol možnosti paralelizácie dátovej časti algoritmu ako aj výpočtovej časti. V oboch metódach spomenutých v tejto kapitoje je pre hľadanie priesečníkov nutné vytvorenie akceleračnej štruktúry. Z hľadiska akceleračných štruktúr je nutné sa zaoberať otázkou možnosti paralelizácie algoritmu, ktorý tieto štruktúry zostavuje. Vytvorenie štrukúry stromového typu možno paralelizovať prvotným rozdelením priestoru na podpriestory, keďže tvotba je rekurzívna a každý podstrom možno interpretovať ako časť scény, je táto časť dobre paralelizovateľná, avšak prvotné rozdelenie priestoru je nutné inicializovať globálne resp. komunikáciou medzi výpočtovými uzlami. Samotné zobrazovanie scény je na základe povahy výpočtov priesečníkov, BRDF, toku fotónov paralelizovateľné rozdelením obrazu na viacero podoblastí a oddelený výpočet možno následne previesť na samostatných uzloch. V tomto smere je dôležité rozvrhnúť úlohu na výpočtovo zhodne náročné časti, čo je pri dynamickej scéne resp. zmene pohľadu náročné. Výsledok renderovania je nutné po dokončení výpočtu zobraziť, komunikácii so zobrazovacím uzlom sa v tomto prípade nemožno vyhnúť a tak je tento krok spoločný pre obe spomenuté metódy. Kľúčovým rozhodovacím faktorom pre mňa pri výbere metódy pre implementáciu bola možnosť dynamického osvetlenia scény. V tomto smere je jednoznačným kandidátom na implementáciu path-tracing, pretože pri zmene polohy resp. orientácie svetelných zdrojov je nutné fotónovú mapu prepočitať. Pokiaľ by sme ju počítali paralelne je nutné výsledok výpočtu distribuovať medzi výpočtovými uzlami. Takýto postup by pri zobrazení v reálnom čase spôsobil zastavenie vizualizácie v závislosti od veľkosti a komplexnosti scény na neakceptovateľný čas, preto je pre mňa metóda path-tracingu ideálnou pre paralelizáciu na princípe rozdelenia obrazu medzi jednotlivé výpočtové uzly.

Kapitola 4

Návrh implementácie

V tejto kapitole sa budem zaoberať návrhom jednotlivých častí aplikácie realtimového paralelného path-tracera. Pri návrhu aplikácie som vychádzal z teoretických poznatkov zachytených v predošlých kapitolách a nakoľko má byť tento algoritmus založený na využití knižnice Intel® Integrated Performace Primitives, predstavím tu možnosti tejto knižnice v úzkej súvislosti s návrhom implementácie path-tracera. Ako už bolo spomenuté v predošlých kapitolách zobrazovanie v reálnom čase je náročným algoritmom a vyžaduje výpočet distribuovať na viacero výpočtových uzlov, preto budem riešiť aj otázku spôsobu paralelizácie a spôsob komunikácie zobrazovacieho uzla s výpočtovými.

4.1 Intel Integrated Performace Primitives

Intel® Integrated Primitives (ďalej len IPP) [14] je súbor optimalizovaných funkcií pre prácu s dátami z oblasti spracovania obrazu, zvuku, vektorovej a maticovej algebry ako aj z oblasti kompresie zvuku a obrazu. Pre tento diplomový projekt je dôležitou časťou knižnica pre realistické zobrazovanie, ktorá ponúka základné funkcie pre zostavenie algoritmu ray-tracingu a aloritmov na ňom založených. Primárne sa funkcie v tejto časti IPP sústreďujú na optimalizované hľadanie priesečníkov lúčov s geometriou scény, ako aj rýchle výpočty osvetľovacích modelov. Z programátorského hladiska poskytuje knižnica prehľadné usporiadanie jednotlivých častí rozdelených podľa oboru pre ktorý sú určené. Súčasťou identifikátorov funkcií je aj typ parametrov a skratky dodatočných operácií, ktoré funkcia prevádza. Použitím knižnice získa programátor širokú škálu funkcií optimalizovaných pre celú radu procesorov, využitím ktorej možno dosiahnuť výkon porovnateľný s implementáciou v assembléri.

Knižnica v rámci časti pre realistické renderovanie poskytuje pre každý krok zostavenia algoritmu sadu funkcií. Akceleračnú štruktúru zapuzdruje IPP do štruktúry `TriangleAccel`, ktorá pre alokovanie a naplnenie svojich štruktúr vyžaduje inicializáciu a vytvorenie Kd-stromu. Rozhranie umožňuje pomocou funkcie `KDTreeBuildAlloc` pri zvolení maximálnej hĺbky stromu vytvoriť a pripojiť koreň do akceleračnej štruktúry. Pre umiestňovanie hyperplôch možno použiť heuristiku SAH, ktorú som diskutoval v 2.1.2, alternatívne metódu počtu polygónov v hyperplochách. Referenčný manuál z výkonnostných dôvodov pochopiteľne odporúča SAH. Pre vytvorenie Kd-stromu je nutné stanoviť objemovú obálku scény pre urýchlenie algoritmu zostavenia, ako aj pre následnú detekciu priesečníkov, k tomuto účelu slúži funkcia `SetBoundBox`. Po zostavení štruktúry je možné pristúpiť k samotnému ray-tracingu. Pre určenie vektorov smerov primárnych lúčov využíva knižnica IPP funkciu

`ipprCastEye`. Táto pracuje s modelom obrazovky - projection plane, ktorý je daný polohou ľavého horného rohu a jednotkovými vektormi vodorovného a vertikálneho smeru vzhľadom na priestor obrazovky. Knižnica IPP primárne pracuje s blokmi obrazu, v kontexte ray-tracingu, s blokmi lúčov. Vstupom je teda poloha pozorovateľa, poloha projekčného plátna vektory horizontálneho a vertikálneho smeru a dôležitým parametrom je poloha a veľkosť bloku v rámci celého obrazu. V 2.4 som sa zaoberal koherenciou lúčov, knižnica v tomto smere ponúka priamo nástroj sledovania úzkeho zvazku lúčov súčasne a spolu s využitím technológií ako je SIMD 2.4 je možné proces ešte viac urýchliť parelnými operáciami na viacerých lúčoch súčasne, v tomto prípade koherencie sa počíta s pravdepodobnosťou že v rámci bloku dôjde k detekcii priesečníka s rovnakým polygónom scény. Výstupom je matica vektorov smerov lúčov s počiatkom v bode pozorovania scény smerujúca cez body v projekčnom plátne. V ďalšom kroku sú jednotlivé lúče podrobované testu na priesečník s geometriou scény funkciou `IntersectEyeSO`, ktorá vracia pole indexov polygónov ako aj barycentrické súradnice priesečníka v danom polygóne. Výpočet v blokoch tu umožňuje vylúčiť nedetekované priesečníky nastavením poľa indexov polygónov v zodpovedajúcich pixeloch na hodnotu -1. Tieto lúče môžu byť z ďalšieho výpočtu vylúčené, čím sa celý algoritmus podstatne zrýchli. Pre výpočet osvetľovacieho modelu je potrebné určiť polohu priesečníka v 3-rozmernom priestore za pomoci funkcie `HitPoint3DEpsSO` a určenie normály. Pre vektorové výpočty potrebné v osvetľovacích modeloch poskytuje IPP funkcie `Dot`, `DotChangeNorm`, `Mul`, `AddMulMul`, `Divi`. Pre výpočet odrazu poskytuje knižnica funkciu `CastReflectionRay`. Keďže jednotlivé výpočty ožiarenia a žiare bodov sú prevádzané v pohyblivej rádovej čiarke, je nutné výstup redukovať na rozsah a bitovú hĺbku zobrazovacieho zariadenia. IPP poskytuje funkciu `ReduceBits_32f8u_C3R` prevádzajúcu vypočítaný obraz z rozsahu $[0, 1]$ v pohyblivej rádovej čiarke do rozsahu 8 bitov na kanál modelu RGB. Interne využíva dithering typu Bayer, Floyd-Steinberg, JJ&N, Stucki. Obraz však možno ešte pred redukovaním bitovej hĺbky upraviť pre zachovanie dynamického rozsahu, čo je v spojení s výpočtom globálneho zobrazenia a všeobecne s realistickým zobrazením nevyhnutné.

4.2 Dáta scény

Na vstup dát je pri riešení úloh realistického renderovania kladený dôraz v oblasti schopnosti zachytit vlastnosti renderovaného prostredia tak aby sa syntetizovaním obrazu dosiahli výstupy zodpovedajúce predlohe modelu scény. Za týmto účelom je pre mna potrebné zvoliť si formát vstupných dát, ktorý umožňuje širokú paletu definície vlastností materiálov a v neposlednom rade definíciu primitívou scény, v konkrétnom prípade polygónov. Rozšíreným a dobre dokumentovaným formátom, vhodným na spracovanie, je 3DS. Poskytuje definíciu polygonálnej geometrie scény, materiálových vlastností spekulárneho a difúzneho povrchu a v neposlednom rade svetelných zdrojov. Takýto povrch bude reprezentovaný farbou spekulárneho odrazu, spekulárnym exponentom a farbou difúzneho odrazu korešpondujúcim z s k_d, k_s a n vo vztahu 3.10 pre Phongov osvetľovací model. Tieto parametre formát 3DS poskytuje natívne. Pre ray-tracing je dôležitá definícia intenzity dokonalého spekulárneho odrazu lúča resp. spekulárnej refrakcie, kde je potrebný index lomu materiálu telesa. Z tohto dôvodu bude zdrojom dát scény formát 3DS, pre ktorý zostavím parser na základe [35] resp. [34], ktoré poskytujú vyčerpávajúcu špecifikáciu tohto formátu. Pre potreby algoritmu path-tracingu je potrebná definícia plošných zdrojov svetla, ich svietivosti a geometrie. Vstupná scéna bude pozostávať z druhého súboru - konfiguračného, kde bude možno definovať dodatočné parametre materiálov, zdroje svetelnej energie a plošné zdroje svetelnej energie. Návrh syntaxe tohto konfiguračného súboru predkladám v Backus-

Naurovej forme:

$$
\begin{array}{rl}
< \text{object-definition} > & ::= < \text{action} >< \text{object-specific-sequence} >< \text{end-sequence} > \\
 & | < \text{object-definition} >< \text{action} >< \text{object-specific-sequence} > \\
 & < \text{end-sequence} > \\
< \text{action} > & ::= \text{"CREATE"} < \text{white-space} > |\text{"MODIFY"} < \text{white-space} > \\
< \text{white-space} > & ::= \text{" "} \\
< \text{name} > & := \text{"name="} < \text{string} >< \text{new-line} > \\
< \text{object-specific-sequence} > & := \text{"CAMERA"} < \text{new-line} >< \text{name} >< \text{camera-attributes} > \\
 & |\text{"LIGHT"} < \text{new-line} >< \text{name} >< \text{light-attributes} > \\
 & |\text{"MATERIAL"} < \text{new-line} >< \text{name} >< \text{material-attributes} > \\
 & |\text{"AREALIGHT"} < \text{new-line} >< \text{name} >< \text{arealight-attributes} > \\
< \text{camera-attributes} > & := \text{"fov="} < \text{real-number} >< \text{new-line} > \\
 & |\text{"position="} < \text{real-number} >< \text{white-space} > \\
 & < \text{real-number} >< \text{white-space} > \\
 & < \text{real-number} >< \text{new-line} > \\
 & |\text{"target="} < \text{real-number} >< \text{white-space} > \\
 & < \text{real-number} >< \text{white-space} > \\
 & < \text{real-number} >< \text{new-line} > \\
 & | < \text{new-line} > \\
< \text{light-attributes} > & := \text{"position="} < \text{real-number} >< \text{white-space} > \\
 & < \text{real-number} >< \text{white-space} > \\
 & < \text{real-number} >< \text{new-line} > \\
 & | < \text{new-line} > \\
 & |\text{'color="} < \text{real-number} >< \text{white-space} > \\
 & < \text{real-number} >< \text{white-space} > \\
 & < \text{real-number} >< \text{new-line} > \\
 & | < \text{light-attributes} >< \text{light-attributes} > \\
< \text{material-attributes} > & := \text{"reflection="} < \text{real-number} >< \text{new-line} > \\
 & |\text{"refraction="} < \text{real-number} >< \text{new-line} > \\
 & |\text{"ior="} < \text{real-number} >< \text{new-line} > \\
 & | < \text{material-attributes} >< \text{material-attributes} > \\
 & | < \text{new-line} > \\
< \text{arealight-attributes} > & := \text{"intensity="} < \text{real-number} >< \text{new-line} >
\end{array}
\tag{4.1}
$$

Vstupu konfiguračného súboru bude predchádzať vstup súboru vo formáte 3DS, tento na základe [35] podporuje pomenovania jednotlivých objektov a materiálov, editácia hodnôt preto bude závislá od hodnoty atribútu *name* zvoleného typu objektu. Pre nastavenie hodnoty indexu lomu materiálu slúži atribút *ior*. Atribúty *reflection* a *refraction* slúžia pre nastavenie faktoru odrazivosti a priehladnosti materiálu. Sú dôležité pre vetvenie lúčov pri ray-tracingu. Plošné svetelné zdroje sú charakterizované atribútmi *name* a *intensity*, kde geometrické vlastnosti sú odvodené od geometrie objektu s týmto menom v scéne, farba svetelného zdroja od farby materiálu tohto objektu a intenzita je celkový výkon svetelného zdroja. V tomto štádiu má aplikácia scénu pripravenú pre prevedenie vykresľovania oboma zvolenými algoritmami

4.3 Zobrazovací reťazec ray-tracingu

Ako už bolo naznačené v 4.1, knižnica natívne prevádza výpočty v bloku resp. zvazku lúčov, preto som návrh vykresľovacieho reťazca prispôsobil tomuto faktu. Po vytvorení primárnych lúčov na základe 4.1 detekujem priesečníky s objektami scény. Formát 3DS podporuje priradenie materiálov jednotlivým polygónom v scéne, čo využijem pri analyzovaní vlastnosti povrchu. V prípade kladnej detekcie priesečníka na základe hodnôt vlastností *reflectivity* a *refractivity* určím vetvenie lúča. V prípade nulových hodnôt je potrebné spočítať osvetľovací model daného materiálu. Na základe 3.10 výpočet spočíva v určení vzájomnej polohy bodu a svetla. Ďalším krokom je pre každý zdroj svetla zostrojenie tieňového lúča, knižnica IPP poskytuje funkciu `CastShadowSO`, ktorá pre daný zväzok lúčov a danú polohu svetla zostrojí tieňové lúče. Následne je potrebné podrobiť tieto tieňové lúče testom na prítomnosť priesečníka s objektami scény. IPP umožňuje funkciou `IntersectAnySO` detekciu objektov a výslednú polohu daného priesečníka v tieni. Pre urýchlenie výpočtu je dôležitý fakt, že pole polygónov 4.1, na ktorých sa nachádzajú priesečníky primárnych lúčov je v tejto funkcii pozmenené a priesečníky nachádzajúce sa v tieni sú z ďalšieho výpočtu vylúčené. Výpočet Phongovho modelu je teda prevádzaný iba na osvetlených bodoch. V prípade difúznej zložky sa pre každý osvetlený bod daným svetelným zdrojom počíta skalárny súčin normalizovaného normálového vektora a tieňového lúča. Výsledná hodnota je násobená farbou daného svetelného zdroja a farbou difúznej zložky materiálu. V prípade spekulárnej zložky Phongovho modelu je potrebné generovať odrazový vektor. Využitím reciprocity BDRF možno tento vektor odvodiť z incidenčných (v tomto prípade primárnych lúčov) čo zamedzí opätovnému prepočítavaniu vektorov reflexného lúča pri každom svetelnom zdroji na základe tieňových lúčov. Následne je pre každý bod spočítaný skalárny súčet reflexného a tieňového lúča (oba normalizované). Výsledok je následne potrebné umocniť spekulárnym exponentom vynásobiť farbou svetla a materiálu. Pre obe zložky modelu vytvorím samostatné akumulátory (viacero svetelných zdrojov), algoritmus bude zohľadňovať aj možnosti aplikácie textúr, čo pri využití akumulátora difúznej zložky bude vecou násobenia texelu a hodnoty difúznej zložky v danom bode. Výsledná farba pixelu je súčtom výpočtu osvetľovacieho modelu pre každé svetlo v danom bode. V prípade priesečníka v bode s materiálom s odrazovými resp. transparentného povrchu s daným idexom lomu je potrebné lúče vetviť. Keďže v bode je spočítaný osvetľovací model, vektor reflekčného lúča je k dispozícii. V prípade refrakcie je potrebé spočítať vektor lomeného lúča, čo prevediem na základe vzťahu 3.8. Vetvenie lúčov v tomto prípade vedie k potrebe vytvorenia pomocnej štruktúry, ktorá bude uchovávať smery incidenčných lúčov v bloku a akumulátor pre akumuláciu mier odrazov a refrakcií z predošlých krokov. Z dôvodu rýchleho prístupu k štruktúre s jednotlivými odrazovými a refrakčnými lúčmi je potrebné alokovať pamäť pre túto štruktúru už v čase pred samotným výpočtom primárnych lúčov. Použitie materiálov s oboma nenulovými vlastnosťami odrazu a transparencie vedie v každom detekovanom priesečníku na rozvetvenie lúča. Štruktúra bude koncipovaná pre určitý stupeň rozvetvenia, tento postup je kompromisom medzi kompletným výpočtom, ktorý by bol pri neustálom alokovaní spomalený a medzi rýchlosťou prístupu k už alokovaným štruktúram. Pred samotným renderovaním bude teda nutné určiť hĺbku odrazov lúča, na základe ktorej sa alokuje potrebná pamäť. Pri výpočte priesečníkov sekundárnych lúčov je postup zhodný s primárnymi lúčmi. Pre celý blok lúčov je dôležitý akumulátor úbytkov vplyvom faktoru odrazu resp. pohltenia pri lome, ktorým sú sekundárne a ďalšie luče normalizované. Takýto postup vedie na riešenie pomocou zásobníka incidenčných lúčov, ktorý bude podľa môjho návrhu mať pevnú dĺžku. Výpočet bloku bude ukončený v prípade vyprázdnenia zásobníka, k vetveniu bude dochádzať v prí-

pade nenaplneného zásobníka. Z blokov zostavený obraz je nutné transformovať do bitovej hĺbky zobrazovacieho zariadenia resp. predspracovať vhodným operátorom mapovania jasu v prípade kontrastov v scéne prevyšujúcich štandardný rozsah.

4.4 Zobrazovací reťazec path-tracingu

Algoritmus path-tracingu je na rozdiel od ray-tracingu schopný simulovať nepriame osvetlenie scény viacerými odrazmi na základe povahy povrchov v scéne. Na rozdiel od ray-tracingu budem v tomto prípade vrhať viacero lúčov plochou pixelu na projekčnom plátne. V prvom kroku je zhodná definícia bodu pozorovania a definícia projekčného plátna v korešpondencii so smerom pohľadu a rozlíšením obrazu. V tomto kroku každým vrhnutým lúčom aproximujem žiar prichádzajúci zo scény v opačnom smere vrhnutého lúča. Samplovanie plochy pixelu prevediem generovaním vzorky s rovnomerným rozložením v rámci plochy pixelu. V ďalšom kroku je potrebné v bodoch, kde je detekovaný priesečník spočítať osvetľovací model. V algoritme som sa rozhodol zahrnúť plošné zdroje svetla, pre ktoré je možné použiť viacero schém vrhania tieňových lúčov. V [9] je popísaných viacero spôsobov vrhania tieňových lúčov k zdrojom svetla. U plošných svetiel je dôležité zvoliť takú schému aby sa bez vnesenia chyby do výpočtu zohľadnili geometrické vlastnosti svetla a emitovaná svetelná energia. Najjednoduchšou možnostou je aj tu využiť metódu Monte Carlo, z každého bodu vrhať jeden tieňový lúč a svetelné zdroje vyberať proporcionálne k ich vlastnostiam a geometrickej konfigurácie bodu a svetla. Tento postup, aj keď sa výpočet potrebný pre lúč zrýchli, zvyšuje rozptyl na daný počet vzorkov na pixel. Alternatívou je vrhanie tieňových lúčov ku každému svetelnému zdroju. Tento postup chcem využiť vo svojej práci, pričom dôležitým faktorom pri výpočte je zohľadnenie žiare svetelného zdroja. Vzťah 3.20 poskytuje formálny zápis žiare transportovanej z povrchu svetelného zdroja cez povrch objektu do smeru pozorovateľa resp. k ďalšiemu povrchu v prípade sekundárnych lúčov. Vlastnosť svetla možno parametrizovať jasom, čo je svietivosť na m^2. Z tohto dôvodu je potrebné do výpočtu zahrnúť aj veľkosť plochy svetelného zdroja. Ja som sa rozhodol tento parameter reprezentovať celkovou svietivosťou svetelného zdroja, čo umožní urýchliť výpočet. V mojej práci uvažujem plošné zdroje svetla s konštantným jasom, preto je toto zjednodušenie prípustné. Ďalším krokom je vzorkovanie bodov povrchu svetla, pri konštantnom jase budem vzorkovať s rovnomerným rozložením proporcionálne k veľkosti jednotlivých elementov, v tomto prípade polygónov. Pre tento účel vypočítam pre každý polygón plochu. Súčtom plôch normalizujem jednotlivé plochy polygónov, čím získam funkciu hustoty pravdepodobnosti diskrétnej veličiny (plochy polygónov s ktorých je zložené svetlo). Postupnou sumáciou získam distribučnú funkciu pravdepodobnosti. Výber konkrétneho polygónu svetla prevediem generovaním náhodného čísla s rovnomerným rozložením $u \in [0, 1]$ a na základe jeho hodnoty zvolím polygón pre vzorkovanie (prehľadávaním poľa hodnôt distribučnej funkcie veľkosti polygónov svetla). Generovanie rovnomerného rozloženia v trojuholníku je možné uskutočniť pomocou rejection sampling a jednotlivé vzorky je možné pripraviť pred samotným spustením algoritmu vykreslovania. Generujem preto $u \in [0, 1]$ a $v \in [0, 1]$ s tým, že odmietnem v prípade $u + v > 1$. Bod v trojuholníku je potom daný vektorovým súčtom:

$$P = A + \vec{AC}u + \vec{AB}v \tag{4.2}$$

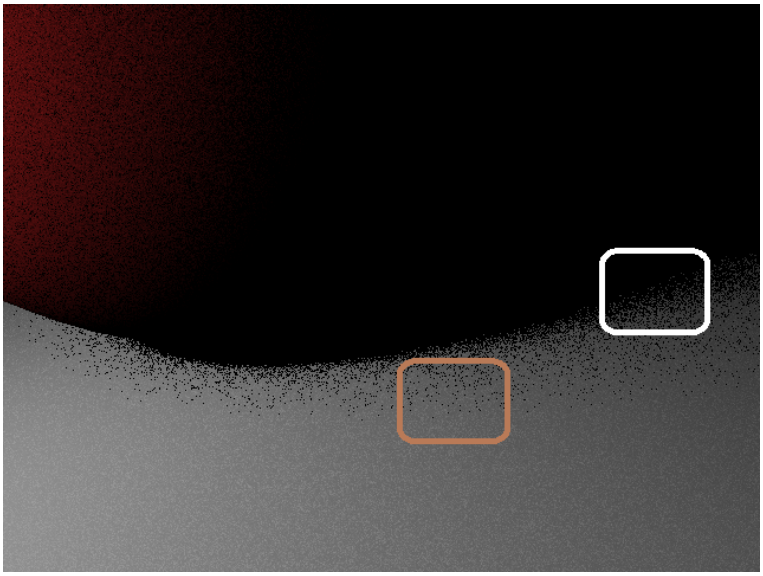
, zhodným postupom pre všetky plošné zdroje vypočítam priame osvetlenie v bode na základe Phongovho osvetľovacieho modelu. Generovanie sekundárneho lúča je ďalšou časťou

výpočtu. Možnosti a spôsob som opísal v 3.4. Pre generovanie smeru lúčov je možné zvoliť rovnomerné rozloženie lúčov v rámci hemisféry. Tento postup je však výpočtovo náročnejší v porovnaní s použitím sofistikovanejšej metódy, ktorá počíta s BRDF materiálu s kosínovým členom v zobrazovacej rovnici. Použitie importance samplingu zníži rozptyl odhadu metódy Monte Carlo ale aj umožní urýchliť výpočet zahrnutím BRDF priamo do vygenerovaných lúčov. Pre tieto potreby navrhujem zostaviť pred samotným výpočtom štruktúru a vygenerovať pre modifikovaný Phongov model rozloženia lúčov v rámci hemisféry nad povrchom. Generovanie difúznych lúčov bude vztiahnuté k normále povrchu, generovaním vzorkov s takýmto rozložením som sa zaoberal v 3.4, kde som opísal postup vygenerovania difúznych, ako aj spekulárnych lúčov na základe BRDF a ich následné prevedenie do priestoru scény. Taktiež je dôležitá aj voľba medzi spekulárnym a difúznym lúčom. Voľbu som založil na intenzite jednotlivých zložiek BRDF, čo je zrejmé zo vzťahu 3.4. V tejto fáze výpočtu sú na základe vlastností materiálu v bloku vygenerované smery sekundárnych lúčov, koeficienty odrazu k_s resp.k_d budem uchovávať v akumulátore v rámci sledovania celej cesty lúča. Prírastok v priesečníku s primárnym lúčom je na základe tohto postupu rovný hodnote osvetľovacieho modelu v tomto bode. Akumulátor je v každom odraze po výpočte osvetľovacieho modelu a akumulácii finálneho výsledku násobený na základe výberu lúča prislúchajúcim koeficientom materiálu. Pre urýchlenie výpočtu bude možné zadať maximálny počet odrazov, ktoré lúč môže uskutočniť. Útlm lúčov však poskytuje možnosť predčasne ukončiť algoritmus. Odrážanie lúča je možné ukončiť v n-tom odraze na základe hodnoty intenzity difúznej resp. spekulárnej zložky podľa vzťahu 3.4 diskutovaného v 3.4.

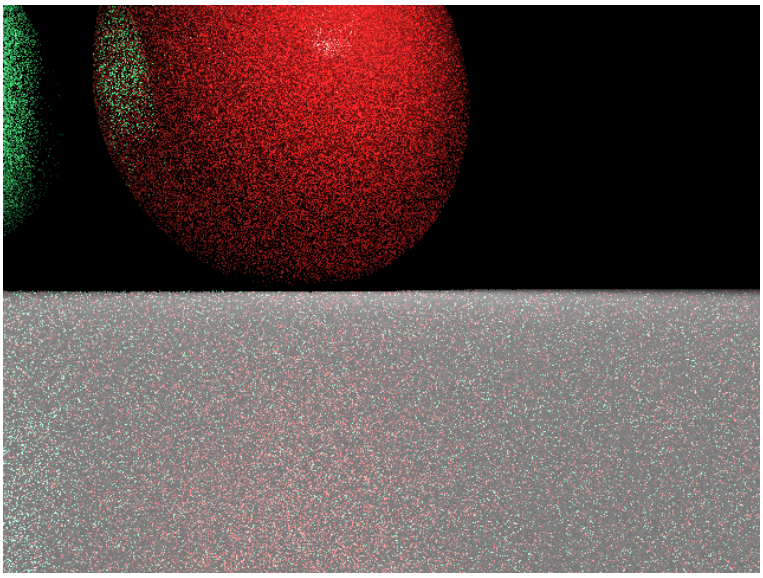
Pre ďalšie urýchlenie je možné použiť adaptívne vzorkovanie. V oblastiah obrazu s nizkym rozptylom nieje potrebný maximálny počet lúčov na pixel, ako v oblastiach s vysokým rozptylom. Tento fakt je možné transformovať do kritéria pre určenie dostatočného počtu lúčov na pixel v konkrétnej oblasti obrazu. Pre implementovanie adaptívneho vzorkovania som sa rozhodol použiť metódu diskutovanú v 3.3, ktorá počet potrebných vzorkov určuje na základe intervalov spolahlivosti, avšak hlavnou nevýhodou spomínaného postupu je lokálne zameranie sa na skúmaný pixel. V oblastiach obrazu na okraji penumbry neostrého tieňa môže spomínaná metóda predčasne ukončiť vzorkovanie pixelu, čo sa v konečnom dôsledku prejaví nežiadúcimi artefaktami v obraze. Možným riešením tohto nedostatku je rozdelenie algoritmu adaptívneho vzorkovania na 2 časti. V prvom kroku prebehne skúmanie okolia pixelu a určenie minimálneho počtu vzorkov na pixel, po dosiahnutí ktorého výpočet pokračuje v druhom kroku, konštrukciou intervalu spoľahlivosti a testom jeho veľkosti. V 4.1 som diskutoval spôsob akým knižnica IPP pristupuje k výpočtu priesečníkov pomocou vrhania zväzku lúčov v bloku. Túto skutočnosť využijem pri skúmaní okolia bodu a pre určenie minimálneho počtu vzorkov v bloku. Môj návrh spočíva v ohodnotení bloku z hľadiska výskytu vyššie diskutovaného dôvodu vzniku artefaktov v obraze. Pre určenie minimálneho počtu vzorkov v bloku je možno na "problémový"blok obrazu pozerať ako na zdroj šumu resp. zdroj kontrastu. Pre konštrukciu algoritmu tieto vlastnosti formalizujem na vzťah určujúci minimálny počet vzorkov na blok. Pre formalizovanie šumu bloku obrazu možno použiť pre zobrazovacie metódy (hodnota žiare pozitívna) alternatívnu definíciu pomeru intenzity signálu k šumu. Vzťah podľa [40] :

$$SNR = \frac{\mu}{\sigma} \tag{4.3}$$

, kde μ je stredná hodnota pixelu v bloku a σ je štandardná odchýlka. Pri vyššej hladine šumu je potrebný väčší minimálny počet vzorkov na blok, využijem preto inverzný vzťah k 4.3. Môj predpoklad je teda založený na lokálnej miere šumu v obraze, kde je vplyvom vačšej



Obrázek 4.1: Rozhranie umbra-penumbra(biela), rozhranie penumbra plné osvetlenie(hnedá), 1 lúč na pixel



Obrázek 4.2: Šum v obraze spôsobený rozptylom pri použití metódy Monte Carlo, 1 lúč na pixel

variancie pravdepodobnosť predčasného ukončenia výpočtu vyššia dokumentuje to oblasť v obr. 4.1 označená bielym obdĺžnikom ako aj v obr. 4.2. Tento predpoklad však nemožno vztiahnuť na oblasti na rozhraní osvetlených častí a penumbry tieňa, kde sa smerom k osvetlenej časti zvyšuje pravdepodobnosť zvolenia tieňového lúča, ktorý zasiahne sveteľný zdroj. Tieto oblasti sú pri nižšom počte vzorkov vzhľadom na veľkú časť odkrytého svetelného zdroja charakterizované znižujúcim sa počtom bodov v tieni ako ukazuje oblasť označená hnedým odĺžnikom v obraze 4.1. V tomto prípade je pravdepodobnosť generovania tieňového lúča, ktorý zasiahne tieniaci objekt proporcionálna k zakrytej ploche svetelného zdroja. Určenie minimálneho počtu lúčov v tejto oblasti na základe SNR by zlyhalo, variancia bloku by bola v tejto oblasti nízka. V konečnom dôsledku by predčasné ukončenie vzorkovania pixelu spôsobilo alias v obraze. Pre tieto oblasti navrhujem metódu založenú na kontraste pixelu s minimálnou a maximálnou hodnotou, čo zodpovedá situácii v týchto oblastiach obrazu. Výraz 4.3 je nutné previesť na minimálny počet vzorkov na pixel v bloku. Tu budem vychádzať z predpokladu, že minimálny počet snímkov má byť proporcionálny k prevrátenej hodnote SNR resp. kontrastu v danom bloku. Pre dodržanie podmienky prerozdelenia, je preto nutné obmedziť hornú hranicu minimálneho počtu vzorkov na pixel v bloku na užívateľom zadaný maximálny počet vzorkov. Toto možno dosiahnuť normalizovaním oboch kritérií na základe maximálnej zistenej hodnoty prevrátenej hodnoty SNR resp. kontrastu na maximálny počet užívateľom zvolených vzoriek na pixel. Formálne zapísané pre SNR:

$$MIN_{vzoriek}(blok_i) = \frac{\frac{1}{SNR(blok_i)} MAX_{vzoriek}}{\arg\max_j \left[\frac{1}{SNR(blok_j)}\right]} \tag{4.4}$$

,kde $\arg\max_j \left[\frac{1}{SNR(blok_j)}\right]$ je hodnota prevrátenej hodnoty SNR bloku s maximálnou hodnotou šumu, takto zaručím proporcionálne prideľovanie minimálneho počtu vzorkov na základe šumu blokov v obraze. V prípade kontrastu je situácia obdobná. Pri všeobecnej scéne je však potrebné brať do úvahy dynamický rozsah hodnôt žiare bodov videných pixelom. V tejto situácii ešte pred samotným určením minimálneho počtu vzorkov upravím hodnoty pixelov obmedzením na definovanú hornú hranicu. Táto operácia zabráni alokovaniu väčšieho počtu vzorkov pre bloky, ktoré sú z hľadiska zobrazenia irelevantné (prevedením do rozsahu zobrazovacieho zariadenia sa hodnoty pixelov prevyšujúce hornú hranicu dynamického rozsahu saturujú na najvyššiu zobraziteľnú hodnotu). Nevýhodou metód je fakt, že skúmané vlastnosti sú odvodené od priebehu obrazovej funkcie a od lokálnych vlastností obrazu v okolí pixelu, v skutočnosti je rozptyl v týchto oblastiach spôsobený integrovaním plochy svetelného zdroja, analýza priebehu funkcie v doméne plochy svetelného zdroja by si vyžiadala uvažovať rozmiestnenie objektov, svetelných zdrojov a pozorovateľa. Uvažovanie takéhoto množstva faktorov by stanovenie minimálneho počtu vzorkov na pixel výpočtovo predražilo. Preto je voľba takejto heuristiky výpočtovo efektívnym spôsobom výberu "dôležitých"oblastí obrazu pre pridelenie väčšieho počtu vzorkov.

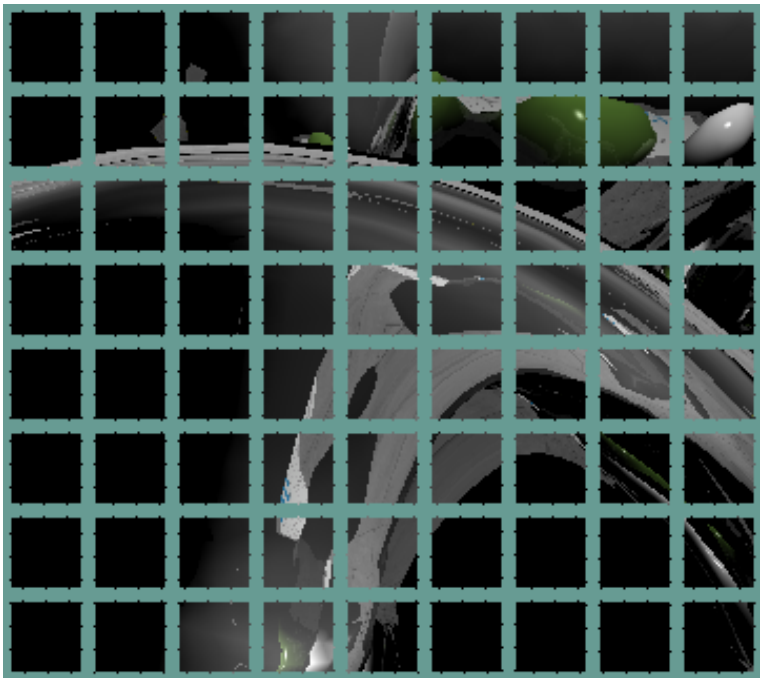
4.5 Paralelizácia výpočtu

Aby bolo možné dosahovať uspokojivú kvalitu zobrazenia pri použití path-tracingu vo vačších rozlíšeniach je aj napriek opísaným spôsobom akcelerácie nutné samotný výpočet paralelizovať. V tomto smere návrh stojí pred protichodnými požiadavkami. Balansovanie výpočtu rozdelením úlohy na zhodne výpočtovo náročné časti je pri všeobecnej scéne veľmi obtiažne previesť a na druhej strane je náročnosť výpočtu určitej časti obrazu vopred nezná-

mou veličinou. Uvažujme prípad rozsiahlej scény renderovanej algoritmom ray-tracingu pozostávajúcej s difúznych materiálov, ktorá obsahuje objekt zo zložitým popisom refrakcie a reflexie povrchu. Výpočet hodnôt pixelov obrazu bude v miestach priesečníkov s difúznymi materiálmi otázkou výpočtu priesečníkov a osvetlovacieho modelu. Avšak v priesečníkoch s objektom s reflexnými resp. refrakčnými vlastnosťami bude potrebné vrhať sekundárne resp. ďalšie lúče, čo lokálne zvýši výpočtovú náročnosť v týchto konkrétnych miestach niekoľkonásobne. Preto je rozdelenie výpočtu na výpočtovo zhodne náročné podúlohy netriviálnym problémom. Môj návrh spočíva vo využití spôsobu akým pristupuje knižnica IPP k vyhodnocovaniu priesečníkov a všeobecne k výpočtom v časti realistického zobrazovania. Knižnica tu implicitne pracuje s blokmi lúčov, čo jednak urýchluje výpočet v rámci jednej pracovnej stanice (využitím koherencie a technológie SIMD) na druhej strane rozdeluje renderovanie obrazu na implicitné podúlohy s pevne daným počtom primárnych lúčov v bloku. Podúlohy je potrebné distribuovať na výpočtové uzly tak aby sa súhrnná výpočtová náročnosť blokov pridelených jednotlivým uzlom výrazne nelíšila. Pre dosiahnutie tohto ciela je potrebné v predstihu poznať rozmiestnenie a výpočtovú náročnosť jednotlivých blokov. Súčasne musíme uvažovať aj o ďalších faktoroch limitujúcich možnosti distribúcie výpočtových podúloh. V tomto smere som zvažoval viacero stratégií pri prerozdeľovania jednotlivých blokov. Úzkym hrdlom adaptívneho prideľovania blokov na základe požiadaviek jednotlivých výpočtových uzlov je komunikácia medzi zobrazovacou stanicou a samotnými výpočtovými uzlami. Takéto riešenie by vyžadovalo uchovávať tabuľku blokov s prislúchajúcim identifikátorom uzla, na základe ktorej by sa po kompletnom vyriešení úlohy obraz zostavil do finálnej podoby v zobrazovacom uzle. Takýto postup som zamietol z dôvodu redundantnej komunikácie a nemožnosti ohodnotit daný blok z hľadiska výpočtovej náročnosti ešte pred odoslaním požiadavky výpočtovému uzlu. Naopak navrhujem jednoduchšiu metódu založenú na poznatku o koherencii v tomto konkrétnom prípade o koherencii susediacich blokov obrazu. Moja hypotéza spočíva v tom, že pri dostatočne malých blokoch v porovnaní s celkovou veľkosťou obrazu a istej spojitosti priľahlých blokov v otázkach výpočtovej náročnosti možno obraz rozdeliť striedavým prideľovaním jednotlivým klientom. Mnou navrhovaná schéma rozdelenia využiva fakt, že pri dostatočne malej veľkosti blokov je výpočtová náročnosť susediacich blokov podobná, čo vyplýva so zobrazenia objektu v obrazovom priestore ako celku, ktorý je blokmi rozdelený na časti, situácia v obr. 4.3 . Cieľom mojej stratégie je susediace časti prerozdeliť na rôzne výpočtové uzly. Pri pevnej veľkosti bloku bude teda potrebné výpočtovému uzlu dodať informáciu o celkovom počte uzlov a o jeho poradí. Formalizujem teraz navrhovaný spôsob distribúcie a označme celkový počet výpočtových staníc $N_{servers}$ a poradie danej stanice s_i. Počet blokov označme N_{blocks} a poradie bloku b_i, priradenie bloku danému klientovi s_i vyhovuje podľa návrhu podmienka:

$$b_i mod N_{servers} = s_i \tag{4.5}$$

Navrhnutý postup jednak odstraňuje nadbytočnú komunikáciu medzi zobrazovacou a výpočtovými stanicami a rieši aj premenlivú výpočtovú náročnosť jednotlivých blokov využitím predpokladu o výpočtovej náročnosti susediacich blokov, čo v konečnom dôsledku balansuje objem kalkulácií potrebných vo výpočtovom uzle. Pri prezentácii výsledku aplikácie však vznikne potreba zobrazovať výstup so stabilným počtom snímkov v určitom časovom intervale. Preto som sa rozhodol do vykreslovacieho reťazca implementovať predčasné ukončenie výpočtu na základe stanoveného času. V spojení s adaptívnym vzorkovaním ,kde náročnejšie bloky obdržia v porovnaní s ostatnými vačšie množstvo vzorkov na pixel, poskytne toto riešenie v porovnaní so stabilným počtom vzorkov na pixel kvalitnejší výstup a zaručí sa tak plynulé zobrazovanie scény.



Obrázek 4.3: Rozdelenie do blokov obrazu (Ray-tracing, lom a odraz lúčov)

4.6 Komunikácia uzlov

Komunikácia zobrazovacieho a výpočtových uzlov vyžaduje návrh protokolu pre vzájomnú komunikáciu. Pre implementovanie som sa rozhodol použiť súbor knižníc Boost [3], kde hlavnými faktormi výberu bola prenositeľnosť a profesionálne rozhranie tejto knižnice. Najdôležitejšou požiadavkou v prvotnom štádiu návrhu je možnosť súčasnej komunikácie viacerých výpočtových uzlov s jedným zobrazovacím, čo vedie na klasický model komunikácie klient-server. V mojej aplikácii slúži zobrazovacia konzola ako dátový server, ku ktorému sa pripájajú viaceré uzly s požiadavkou na prenos dát scény. Výpočtový uzol preto musí byť schopný prijať a obsluhovať viacero výpočtových uzlov súčasne, pôjde o konkurenčný server. Po odoslaní dát scény prechádza komunikácia do druhej fázy v ktorej zobrazovacia konzola prenáša požiadavky na každý výpočtový uzol, ktorý prevádza pridelenú časť výpočtu a zasiela výsledok späť zobrazovacej stanici. Knižnica Boost.asio poskytuje zapuzdrenie sieťového rozhrania a umožňuje komunikáciu zasielaním správ. Naviac poskytuje mechanizmus pre súčasné obsluhovanie viacerých pripojení - asynchronne pomocou callback funkcii. Tento mechanizmus je postavený na návrhovom vzore proactor, viac možno nájsť v [30] resp. v [20]. Tento mechanizmus umožňuje v rámci jedného threadu obsluhovať bez blokovania viacero socketov, ako čítanie tak aj zapisovanie možno previesť asynchronne s behom ostatného kódu v danom vlákne, po dokončení operácie je možné volať callback funkciu pre potreby analýzy prijatých respektíve k odoslaniu dát na základe požiadavky. Boost.asio obsahuje aj funkcie špecializované pre zasielanie správ v protokoloch využívajúcich oddelovače správ, napr. protokoly ako HTTP resp. FTP, ktoré majú jednotlivé správy oddelené sekvenciou "\r \n". V návrhu pokračujem špecifikáciou protokolu.

Komunikáciu medzi zobrazovacou a výpočtovými stanicami možno rozdeliť do dvoch fáz, v prvej fáze príde k nadviazaniu spojenia a prenosu dát scény v smere k výpočtovým uzlom. V druhej fáze prichádza opakovane k zasielaniu požiadavky zobrazovacou stanicou

výpočtovým uzlom a zasielaniu časti vyrenderovaného obrazu zobrazovaciemu uzlu. Pri navrhovaní protokolu som sa inšpiroval protokolom FTP, avšak na rozdiel od neho používam jediné spojenie ako na komunikáciu správami tak aj zasielanie dát. Textová časť protokolu pozostáva zo správ zakončených oddelovačom "\n". Komunikácia je zahájená pripojením výpočtového a prijatím spojenia zobrazovacím uzlom. Správy zasielané serverom majú formát:

```
<REQUEST-NUMBER><WS><REQUEST-TEXT><WS><MESSAGE-SPECIFIC><DELIM>
```

, kde <REQUEST-NUMBER> je číselný kód správy a spolu s <REQUEST-TEXT> tvorí identifikátor. Zobrazovacia stanica ako aj výpočtové uzly prechádzajú stavmi, ktoré sú v mojom protokole nástrojom synchronizácie. Pole <MESSAGE-SPECIFIC> je špecifické pole, ktoré obsahuje číselnú resp. číselné hodnoty oddelené <WS> medzerou. Formát tohto poľa je vysvetlený v tabuľke (odkaz). Komunikáciu iniciuje výpočtový uzol zaslaním správy "100 HELLO<DELIM>", výpočtová strana odpovedá OK<DELIM>. Vnútorne prechádza server ako aj klient stavmi. Stav sa v prvej fáze mení vždy po potvrdení klientom. Po úvodnej výmene nasleduje zasielanie jednotlivých scénických dát. Postup výmeny je pri každej položke daný presným postupom. Odoslanie správy serverom s popisom dát, odpoveď klienta OK<DELIM>, zaslanie samotných dát serverom, odpoveď klienta OK<DELIM>. Formát jednotlivých správ serveru pre popis dát som zachytil v nasledovnej tabulke:

<REQUEST-NUMBER>	<REQUEST-TEXT>	<MESSAGE-SPECIFIC>
100	HELLO	
200	VERTEX	<LGT>
201	VERTEXNORMAL	<LGT>
202	VERTEXNORMALIDX	<LGT>
203	VERTEXNORMALIDX	<LGT>
204	FACES	<LGT>
205	FACENORMALS	<LGT>
206	MATERIALIDX	<LGT>
207	MATERIALS	<LGT>
208	LIGHTS	<LGT>
211	AREALIGHTS	<CNT><WS><LGT>
209	TEXTURE 1	<PTN><WS><LGT>
210	INIT	<IMG-PAR><WS><BLK-PAR>

, <LGT> je počet bajtov ktoré server následne odošle, klient tento objem dát prijme, <CNT> je dôležitý údaj pre interpretáciu, ktorý reprezentuje počet plošných svetiel nakoľko štruktúra má premenlivú dĺžku. <IMG-PAR> reprezentuje rozmery obrazu oddelené <WS>, <BLK-PAR> predstavuje rozmery bloku oddelené <WS>. Každá správa je zakončená '\n", po obdržaní správy "210 INIT<IMG-PAR><WS><BLK-PAR>" klient reaguje OK<DELIM>, následne sa inicializujú interné štruktúry a prechádza do stavu, keď je pripravený prijať požiadavku na renderovanie. Požiadavka na renderovanie má tvar:

```
300 RENDER <IMG-PAR><WS><BLK-PAR><WS><NS><OS><CAM><A><D><R><B><Q><T><DELIM>
```

, kde <NS> je počet serverov,<OS> je poradie serveru, <CAM> sú parametre kamery (súradnice kamery a cieľa kamery oddelene <WS>), <A> je algoritmus ktorým sa má scéna vykresľovať (1—2), <D> je hĺbka vetvenia lúčov pri ray-tracingu, <R> je maximálny počet lúčov na pixel v path-tracingu, <B> je maximálny počet odrazov lúča <Q> je kvalita výpočtu,<T> je časový limit. Po obdržaní správy klient prevedie výpočet a odošle zodpovedajúci objem dát, ktorý je odvoditeľný od veľkosti obrazu, bloku, počtu serverov a poradia serveru. Špecifikácia protokolu je týmto kompletná. Interpretácia prenesených dát je doménou implementácie, bude dokumentovaná v programovej dokumentácii.

Kapitola 5

Implementácia

Pre implementáciu navrhnutéj aplikácie som sa rozhodol využiť programovací jazyk C++. Keďže program vyžaduje komunikáciu medzi serverovou - zobrazovacou časťou a klientskou - výpočtovou časťou využil som knižnicu Boost [20]. Táto ponúka zapúzdrenie komunikácie protokolom TCP/IP a naviac je prenositeľná na viacero platforiem, čo dáva možnosti portovania aplikácie napr. do prostredia Unixu. Základy renderovacej časti sú založené, ako už bolo v predošlej kapitole spomenuté na knižnici Intel integrated performace primitives, ktorá poskytla rozhranie pre prácu s akceleračnou štruktúrou resp. detekciu priesečníkov v scéne. Aplikácia pre vizualizáciu renderovanej scény využíva API knižnice GLUT a jej nadstavbu pre tvorbu užívateľských rozhraní - knižnicu GLUI. Jednotlivé časti implementácie som umiestnil do tried. Celkovo program pozostáva so serverovej časti, ktorá obsahuje parser súborov formátu 3DS, reprezentovaný triedou *Import3DS*. Táto trieda transformuje vstupný súbor do štruktúr triedy *Scene*, ktorá zapuzdruje operácie s elementami scény a je ako som už mohol viackrát potvrdiť v projektoch, univerzálne znovupoužiteľná v grafických aplikáciách. Gro aplikácie tvoria renderovacie algoritmy a metódy transformujúce štruktúry triedy *Scene* do interných štruktúr potrebných pre renderovanie s IPP, trieda *IPPRenderer*. Sieťová časť servera je reprezentovaná triedami *server* resp. *session*, ktoré zapuzdrujú komunikačný protokol. Inštanciácia triedy *session* a asynchronne operácie ktoré poskytuje knižnica Boost mi umožnili implementovať komunikačnú časť schopnú obsluhovať viacero výpočtových uzlov. Z implementačného hladiska beží server v 2 vláknach. Implicitné vlákno implementuje užívateľske rozhranie a zobrazovanie pomocou OpenGL, explicitne vytvorené vlákno komunikuje s výpočtovými uzlami. Klientská časť aplikácie je reprezentovaná triedami *client* resp. triedou *IPPRenderer*, nakoľko program vznikal iteratívne pridávaním jednotlivých častí pri riešení práce, bola táto trieda súčasťou neparalelnéj verzie z počiatkov implementácie. V ďalšej časti podrobne opíšem spôsob implementácie a podložím dôvody voľby jednotlivých štruktúr krátkymi výňatkami zaujímavých segmentov implementácie.

5.1 Zobrazovacia konzola

Serverovú časť aplikácie tvorí zobrazovacia konzola - samostaná aplikácia implementujúca vstup scénických dát, komunikačný protokol, založený na zasielaní správ a dát, a zostavovanie a zobrazovanie vyrenderovanej scény. Zobrazovacia časť kódu je postavená na knižnici GLUT. Pre finálne vykreslenie používam techniku renderovania do textúry. Jednotlivé časti obrazu odoslané zobrazovacími uzlami zostavím do uceleného obrazu, ktorý pre OpenGL

definujem funkciou *glTexImage2D*, pri testovaní aplikácie som sa však stretol s tým, že niektoré grafické karty nepodporujú automatikcé vytvorenie mipmáp (zvolením parametra `LEVEL` na hodnotu 0), toto som v kóde ošetril makrom pri preklade a alternatívne je možné vytvoriť bitmapy funkciou *gluBuild2DMipmaps* knižnice GLU. Textúra je následne nanesená na quad, pomocou techniky screen aligned quad. Pri koncipovaní vykreslovania týmto spôsobom som uvažoval aj o možnosti prenášania obrazu uloženého v niektorom floating point formáte, čo by umožnilo ďalšie spracovanie a prevedenie do formátu zobrazovacieho zariadenia priamo v serverovej časti. Avšak takýto postup by spôsobil nárast objemu prenášaných dát. Zobrazenie je teda prevedené nastavením funkciou *glOrtho* a následne je textúra nanesená na quad s príslušnými rozmermi a textúrovacími súradnicami. Parametre vykresľovacích reťazcov možno meniť z kompaktného GUI implementovaného pomocou knižnice GLUI. Pohyb v scéne je ovládaný kombináciou klávesnice a myši, za pomoci callback funkcií ktoré poskytuje GLUT. Ovládanie aplikácie naväzuje vo funkcii spätného volania *onDisplay* na sieťovú časť serveru.

Server je koncipovaný ako konkurečný, schopný dátovo obslúžiť viacero klientov. Je založený na prostriedkoch knižnice Boost. Kontrétne sú využité časti Asio pre vytvorenie socketov resp. Thread pre synchronizáciu vlákien serveru a ošetrenie kritických sekcií. Knižnica Boost ponúka pre implementáciu naviazania spojenia, možnosť vytvorenia socketu, do ktorého je možné zapisovať resp. čítať. Spojenie serveru s klientami som koncipoval v triedach *server* resp. *session*. Knižnica Boost naviac poskytuje mechanizmus asynchrónneho vykonávania operácií na socketoch a poskytuje alternatívu vytváraniu vlákien pre obsluhu jednotlivých konkurenčných spojení. Koncept som diskutoval v 4.6. Vytvorenie socketu pre načuvanie prebieha v konštruktore triedy *server* ktorá má v parametroch k dispozícii port, na ktorom bude socket načúvať a pointer na inštanciu triedy *IPPRenderer*, taktiež je tu vytvorená inštancia triedy *session*. Dôležité je predanie štruktúry *io_service* , ktorá poskytuje sieťove rozhranie a riadenie asynchrónych volaní. V tejto fáze je započaté načúvanie na zvolenom TCP porte, volaním funkcie *::async_accept* štruktúry *acceptor*, ktorá umožňuje zvoliť obslužnú funkciu pre potreby nadviazania spojenia a vstupno-výstupných operácii, volanie je neblokujúce. Pripojenie je ošetrené vo funkcii *server::handle_accept*, ktorej sa predá ukazovateľ socketu prijatého spojenia. Pre uchovanie aktívnych spojení som vytvoril štruktúru implementovanú typom vector. Následne je vytvorená nová inštancia *session* a započaté nové načúvanie. Samotný protokol je implementovaný v triede *session*. Komunikacia je po nadviazaní spojenia iniciovaná vo funkcii *session::start*, tato zapíše do socketu správu na základe návrhu v 4.6. Úspešný zápis do socketu je ošetrený callback funkciou *session::handle_write*, ktorá číta zo socketu potvrdzujúcu správu, toto čítanie je ošetrené callback funkciou *session::hadle_read*, následne prebieha odosielanie jednotlivých polí definície scény. Volanie správ je uskutočnené v cykle funkcií *session::handle_write* resp. *session::hadle_read*, každým úspešným prijatím potvrdzujúcej správy je interný stav inštancie triedy *session* inkrementovaný. Po odoslaní inicializačnej správy
`210 INIT <IMG-PAR><WS><BLK-PAR>` je z pohľadu serveru uzol pripravený prijímať požiadavky na renderovanie scény. Odoslanie požiadavky a následné prijatie vyrenderovanej scény je uskutočnené v cykle funkcií *session::handle_request_read* resp. *session::handle_request_write*, s mechanizmom ošetrenia uzavretia socketu v prípade odpojenia klienta. Takáto implementácia mi umožnila obsluhu viacerých spojení súčasne bez blokovania behu vlákna. Dôležitou súčasťou implementácie bolo použitie mechanizmu uzamykania kritických častí kódu. Inštancia triedy *IPPRenderer* je zdieľaným prostriedkom medzi jednotlivými inštanciami *session*. Ošetrenie prístupu na dátové štruktúry som implementoval mechanizmom, ktorý poskytuje Thread, časť knižnice Boost. Segmenty kódu operujúce so zdielanými dátami som

uzamkol volaním funkcie *::lock* triedy *scooped_lock*. Ide o modifikáciu uzamykania, ktorá automaticky uvolní mutex pri opustení bloku, v ktorom bola volaná. Tento model umožňuje predísť uviaznutiu v programe pri neuvolnení mutexu. Samotné renderovanie je ošetrené volaním funkcie *server::send_request* s parametrami pre vykreslovanie scény. Sú generované správy na základe počtu a poradia klientov v štruktúre typu vector a následne sú zapísané do socketov jednotlivých pripojení. Po prijatí výsledkov renderovania je obraz zostavený a vizualizovaný, všetky sockety a inštancie, kde nastalo prerušenie pripojenia sú zmazané, aplikácia čaká na vstup užívateľa.

5.2 Výpočtový uzol

Výpočtový uzol je koncipovaný ako jednovláknová aplikácia. Disponuje implementáciou protokolu a algoritmov ray-tracingu resp. path-tracingu. Komunikáciu iniciuje vytvorením socketu a nadviazaním spojenia funkciou *socket::connect*. Pre synchronizáciu so serverovou časťou prechádza klient pri úspešnom prijímaní správ internými stavmi, podobne ako v FTP protokole má klient neustále informáciu o stave serveru. Klient odpovedá na každú prijatú správu potvrdzujúcou, iba v prípade že okamžitý stav odpovedá protokolu. Veľkosti dátových segmentov je súčasťou správ serveru. Interpretácia správ serveru je implementovaná funkciou *client::parse_message* v súlade návrhom protokolu v 4.6. Správa je analyzovaná, potvrdená a následne je prečítaný objem dát, opatovne nasledovaný zaslaním potvrdzujúcej správy. Klient po obdržaní `210 INIT <IMG-PAR><WS><BLK-PAR>` prejde do stavu, keď číta zo socketu požiadavku a následne podľa obsahu požiadavky vyrenderuje svoju časť a zašle výsledok serveru. Vstupno-výstupné operácie klienta sú v tomto prípade synchrónne blokujúce, ošetrené mechanizmom výnimiek. Knižnica v kontexte môjho návrhu obsahuje vstupno-výstupné funkcie*::read_until* resp. *::write_until*, ktoré umožňujú čítanie zo socketu po dosiahnutie určitého oddelovača. Takto je možné ošetrovať situácie pri ktorých je vo vstupno-výstupnom bufferi viac dát.

5.3 Implementácia ray-tracingu

Kĺučovou časťou implementácie bolo zostavenie algoritmov pre renderovanie scény. Parameter predávaný serverom je v tomto prípade počet odrazov resp. refrakcií lúča v scéne. Nakoľko vetvenie lúča vedie k rekurzívnemu výpočtu, rozhodol som sa ošetriť túto skutočnosť zásobníkom typu FIFO, ktorý som implementoval pomocou *std::queue*, velkosť zásobníka je statická alokovaná pred začiatkom výpočtu. V priesečníkoch lúčov so scénou môže prísť k 4 udalostiam. Pohltenie, v prípade absencie odrazu resp. reflexie, k odrazu resp. k lomu a k obom udalostiam vedúcim k vetveniu. Nove smery spolu s počiatočnými bodmi sú uložené na zásobník, z ktorého sú v ďalších iteráciách vyberané a vyhodnocované. Zvolený postup mi umožnil generalizovať jadro algoritmu v zmysle vstupných a výstupných dát, kde v cykle odoberám vrchol zásobníka a prírastok k žiare pixelu akumulujem. Ukončovacou podmienkou je v tomto prípade prázdny zásobník. Algoritmus raytracingu som zachytil v nasledujúcom pseudokóde:

```
Pre každý blok
    Ulož_primárne_lúče_na_zásobník()
    hlbka = 0;
    f_odrazu_lomu = 1;
    while(!zásobník_prázdny && hlbka < max_hlbka);
          Odober_vrchol_zásobníka();
          Vypočítaj_priesečníky();
          Vypočítaj_osvetlovací_model();
          if odraz: Spočítaj_odrazové_lúče;
               Ulož_do_zásobníka(f_odrazu_lomu *= f_odrazu);
          if lom: Spočítaj_refrakčné_lúče;
               Ulož_do_zásobníka(f_odrazu_lomu *= f_lomu);
          fr_odrazu_lomu -= (f_odrazu + f_lomu)*f_odrazu_lomu;
          hlbka++;
          výstup += svetelný_model*f_odrazu_lomu;
```

V pseudokóde je `f_odrazu_lomu` premenná generalizujúca prírastok lúča k celkovej žiare pixelu. V ray-tracingu som implementoval osvetľovací model Phong. Po výpočte priesečníkov sú k dispozícii pre každý lúč bloku normálové vektory, z ktorých pomocou incidenčných lúčov vypočítam odrazový vektor. Difúzna zložka je daná skalárnym súčinom normály a tieňového lúča, spekulárna n-tou mocninou skalárneho súčinu odrazového vektora a tieňového vektora. Obe zložky sú produktom farby svetla ktorý emituje svetlo a koeficientov materiálu k_s, k_d. Na základe vlastností materiálov ďalej pristupujem k výpočtu smeru odrazených lúčov, keďže tieto sú potrebné už pri výpočte osvetľovacieho modelu, spočítam priesečníky z týchto lúčov s objektami scény a uložím ich spolu s odrazovými lučmi (v kontexte ďalšieho cyklu pôjde už o incidenčné lúče) do zásobníka. Obdobne riešim lomené lúče. Textúru aplikujem na difúznu zložku akumulátora. IPP poskytuje pre urýchlenie výpočtu možnosť použitia poľa - masky priesečníkov ako parametra funkcie, brané sú do úvahy iba tie zložky bloku, ktoré majú nezápornú hodnotu masky. Takto je možné výpočet ukončiť už pri nedetekovaných priesečníkoch.

5.4 Implementácia path-tracingu

Implementáciu path-tracingu som založil na stávajúcom algoritme ray-tracingu, nakoľko som implementoval difúzne resp. spekulárne lúče, využitie zásobníka pre vetvenie lúčov nebolo potrebné. Pre urýchlenie výpočtu som sa rozhodol smery lúčov, bodov v ploche obrazového pixelu a bodov na ploche svetelného zdroja generovať v inicializačnej fáze pred samotným renderovaním. Generovanie je implementované funkciou *IPPRenderer::InitSampler*. Difúzne lúče som generoval na základe vzťahu 3.41, výsledkom sú smery difúznych lúčov v tangent space, ktorý sa v algoritme prevedie do objektového priestoru. Spôsob uloženia smerov som prispôsobil výpočtu v IPP, preto je v každom kroku generovaný blok smerov difúznych odrazov, pre stanovenie smeru difúznych lúčov v celom bloku je potrebné generovať jedno náhodné číslo priraďujúce vygenerované difúzne lúče v bloku. Podobne som generoval spekulárne lúče, tieto sú však vztiahnuté k priestoru odrazového vektora. Pre generovanie bodu na povrchu svetelného zdroja som generoval súradnice na základe 4.2. V štruktúrach triedy *IPPRenderer* je v poli plošných svetelných zdrojov predpočítaná distribučná funkcia pravdepodobnosti založená na ploche jednotlivých polygónov s ktorých

je zložený svetelný zdroj, pravdepodobnosť zvolenia daného polygónu je proporcionálna k jeho ploche. Pre vygenerovanie bodu na povrchu polygónu svetelného zdroja sú potrebné 2 náhodné čísla. Postup ktorý som tu využil prispel k akcelerácii výpočtu. Ďalšie zrýchlenie výpočtu som založil na princípe útlmu využitím ruskej rulety. Pre uchovanie prenosu BRDF v každom odraze lúča využívam pre každú farebnú zložku akumulátor, ktorého hodnota je závislá na postupe lúča scénou a na vlastnostiach materialov objektov v scéne. Odrážanie lúča ukončujem na základe vzťahu 3.4 .V návrhu v sekcii 4.4 som načrtol možnosti adaptívneho samplovania jednotlivých pixelov. Aplikácia pre prvú fazu - stanovanie minimálneho počtu vzorkov na pixel, umožňuje zvoliť metódu založenú na SNR resp. na kontraste minimálnej a maximálnej hodnoty pixelu. Druhá fáza umožňuje nastaviť úroveň spoľahlivosti. Algoritmus po vyhodnotení obloku obrazu jedným lúčom na pixel vyhodnotí spomínané kritérium na základe výpočtu strednej hodnoty a rozptylu resp. rozdielu minimálnej a maximálnej hodnoty pixelu bloku. Knižnica IPP umožňuje pre výpočet využiť funkcie *Mean_StdDev* ktorá implicitne pracuje so zvoleným blokom obrazu a taktiež funkciu *MinMax*. Výsledkom tohto kroku je minimálny počet vzorkov pre blok. V druhej fáze algoritmus po dosiahnutí minimálneho počtu vzorkov počíta interval spoľahlivosti pre každý pixel bloku. Pokiaľ pre zvolenú úroveň významnosti je interval spolahlivosti nižší ako prah výpočet je pre tento pixel ukončený. Algoritmus uvažuje pre každý bod obrazu samostatnú váhu na základe počtu vzorkov. Výsledný obraz je následne váhovaný, prevedený na rozsah nastaviteľný v GUI, a prevedený do modelu RGB s 8 bitmi na kanál.

Kapitola 6

Dosiahnuté výsledky

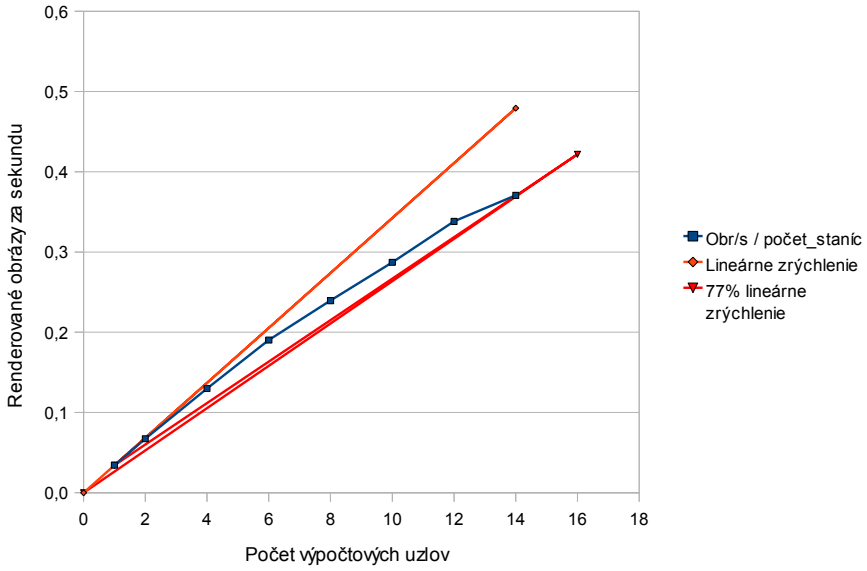
V tejto kapitole sa sústredím na vyhodnotenie popísaných a implementovaných metód v praktickom nasadení aplikácie. Vyhodnocovať budem, pre zobrazovanie v reálnom čase potrebné urýchlenie vplyvom paralelizácie výpočtu. V ďalšej časti sa zameriam na objektívne hodnotenie kvality výstupov aplikácie a v závere zhrniem implementované metódy a schopnosti aplikácie splniť účel, pre ktorý bola navrhnutá.

6.1 Urýchlenie výpočtu paralelizáciou

Test urýchlenia paralelizáciou výpočtu má experimentálne potvrdiť správnosť postupu pri rozdeľovaní jednotlivých blokov výpočtovým uzlom a to aj v členitých scénach, kde je rozdiel vo výpočtovej náročnosti blokov značný. Aby bol test objektívny, rozhodol som sa založiť ho na renderovaní viacerých pohľadov do scény. Tento postup má simulovať reálne ovládanie aplikácie. V konkrétnom prípade som použil 10 pohľadov na scénu - časť animácie ktorú je možno vytvoriť aplikáciou. Pre testovanie som zvolil rozlíšenie obrazu 320 na 240 pixelov pri 64 vzorkách na pixel s maximálnym počtom odrazov lúča 5, testovacími stanicami boli počítače vybavené 2 jadrovým procesorom Intel Core 2 Duo bežiace na 2,66 Ghz. Nízke rozlíšenie som zvolil z dôvodu obmedzenia dátových prenosov, ktoré by mohli do značnej miery negatívne ovplyvniť výsledok. Ďalším opatrením bolo opakovanie testu pre zvolený počet výpočtových staníc tak aby som zamedzil vplyvu cudzích prenosov na sieti na výsledok experimentu. Následne som čas renderovania previedol na počet snímkov za sekundu. Počet vyrenderovaných obrázkov v závislosti od počtu výpočtových uzlov som zachytil v tabulke 6.1 resp. v grafe 6.1. Tento výsledok v prvom rade potvrdzuje, že výpočet sa vplyvom paralelizácie urýchlil. Ďalším kladným výsledkom experimentu je aj potvrdenie správnosti distribuovania výpočtu na jednotlivé výpočtové uzly. Testovacia scéna pozostávala z animácie 10 obrázkov s rôznou polohou kamery voči scéne, jednotlivé bloky mali teda premennú výpočtovú náročnosť a naviac pri zmene počtu uzlov sa priradenie zmenilo. Výsledok potvrdil, že voľba spôsobu prideľovania diskutovaná v 4.5 dokáže balansovať rôznu výpočtovú náročnosť jednotlivých blokov a to aj v nasadení v reálnom čase pri pohybe v scéne.

Počet výpočtových uzlov	1	2	4	6	8	10	12	14
Obrázkov za sekundu	0,034	0,067	0,129	0,190	0,239	0,287	0,339	0,370

Tabulka 6.1: Tabulka závislosti renderovaných framov za sekundu od počtu výpočtových uzlov



Obrázek 6.1: Urýchlenie výpočtu vplyvom paralelizácie, vzťah počtu renderovaných framov k počtu výpočtových uzlov

6.2 Kvalita výstupu

Pre objektívne zhodnotenie kvality vyrenderovaného obrazu je nutné zvoliť metriku pre hodnotenie. Pre stanovenie kvality obrazu som sa rozhodol použiť strednú kvadratickú odchýľku, ktorá je často používanou pri objektívnom hodnotení kvality referenčných a testovacích obrazov. Rozdiel bude tvoriť dĺžka vektora daného zložkami RGB referenčného obrazu a obrazu s danými parametrami. Formalizujem teraz túto metriku. Vygenerované, ako aj referenčný obraz nech má rozmery M , N, je strednú kvadratickú odchýľka daná:

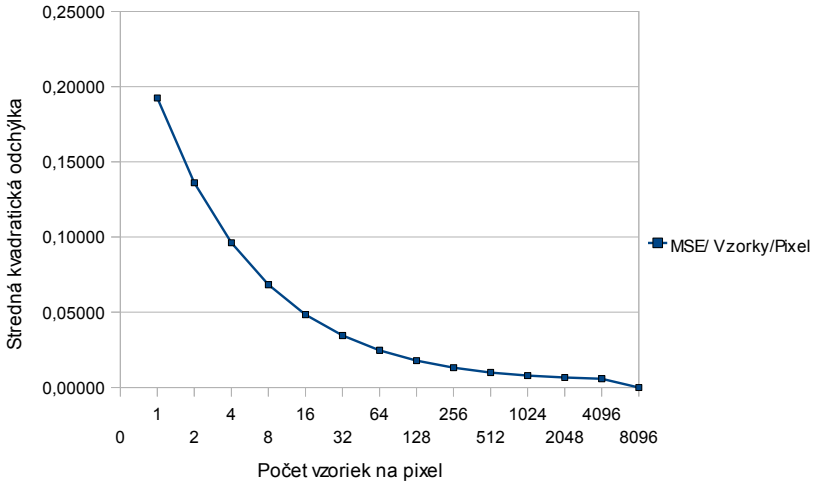
$$RMS(I, \widehat{I}) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} (I(m,n) - \widehat{I}(m,n))^2 \tag{6.1}$$

, keďže obraz je zložený so zložiek RGB, môžeme rozdiel $I(m,n) - \widehat{I}(m,n)$ zapísať ako dĺžku vektora s počiatkom v bode $x_{ref}[R_{ref}, G_{ref}, B_{ref}]$ resp. koncom v bode $x_{measured}[R_{measured}, G_{measured}, B_{measured}]$, označme zložky RGB bodu $I(m,n)$ resp. $\widehat{I}(m,n)$ takto: $I(m,n)_r$, $I(m,n)_g$, $I(m,n)_b$ resp. $\widehat{I}(m,n)_r$, $\widehat{I}(m,n)_g$, $\widehat{I}(m,n)_b$, potom môžeme formalizovať:

$$\begin{aligned} RMS(I, \widehat{I}) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \Big[& (I(m,n)_r - \widehat{I}(m,n)_r)^2 + \\ & (I(m,n)_g - \widehat{I}(m,n)_g)^2 + \\ & (I(m,n)_b - \widehat{I}(m,n)_b)^2 \Big] \end{aligned} \tag{6.2}$$

V mojom prípade som sa rozhodol ako referenčný obraz s najvyššou kvalitou zvoliť výstup aplikácie s rovnakým počtom vzorkov na pixel. Uvažoval som na základe predpokladu o konvergencii odhadu metódy Monte Carlo k skutočnej hodnote hladanej žiare pixelu a schopnosti zobrazovacieho zariadenia, ako aj formátu exportu vyrenderovanej scény. Zvolil som 8000 vzoriek na pixel. Zvolenú hodnotu som overil viacerými experimentami a porovnaním obrazu pri dvojnásobnom počte vzorkov, výstup sa odchyľoval o rádovo 10^{-5}, tento rozdiel však neovplyvnil objektívnosť výsledku nakoľko odchýľky v testovacích dátach boli v minimálnom prípade o 3 rády väčšsie. Výsledkom experimentu je porovnanie navrhnutých metód určenia minimálneho počtu vzorkov na pixel a ich vplyvu na celkovú kvalitu renderovaného obrazu. Cieľom prvého experimentu bolo potvrdiť konvergenciu metódy Monte Carlo, nakoľko som implementoval importance sampling je tento experiment taktiež overením správnosti navrhnutého a realizovaného implementačného riešenia - naprogramovaného algoritmu. Test spočíval v renderovaní statického pohľadu do scény pri zvyšujúcom sa počte vzoriek na pixel. Výsledok dokumentuje graf 6.2. Ako možno vidieť implementovaná metóda je stabilná a nevykazuje skreslenie.

Ďalším experimentom bolo porovnanie schopnosti metód pre určenie minimálneho počtu vzorkov na blok pre následné vyhodnocovanie intervalov spolahlivosti. Keďže implementované metódy boli založené na vyhodnocovaní blokov obrazu a stanovenia minimálneho počtu v kombinácii s intervalmi spoľahlivost, výsledný priemerný počet vzorkov na pixel bol závislý od nastaveného maximálneho počtu a od schopnosti danej metódy stanoviť tento minimálny počet. Ďalším faktorom bolo aj použitie intervalov spoľahlivosti, ktoré v mojom prípade stanovujú spoľahlivosť odhadu metódy Monte Carlo v jednotlivých bodoch. Výsledky experimentu, zachytené v grafe 6.3 ukázali zaujímavé výsledky. Pre testovanie



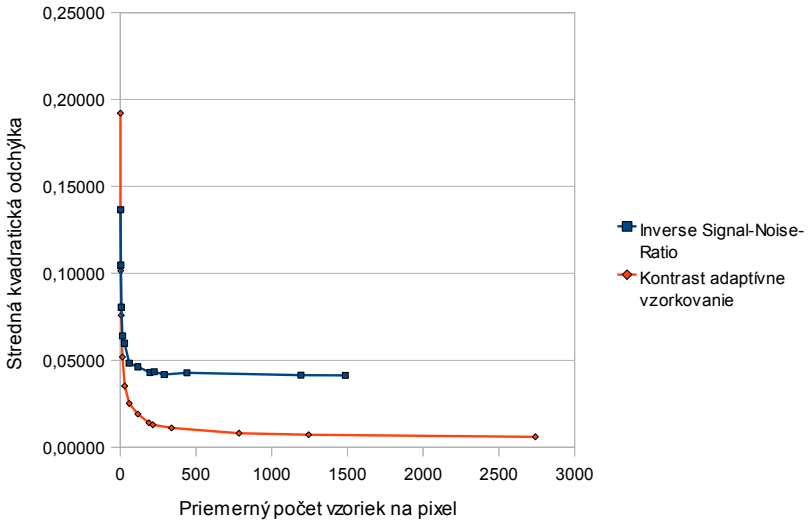
Obrázek 6.2: Vzťah strednej kvadratickej odchýlky k počtu vzoriek na pixel, odhad hodnoty pixelov konverguje k skutočnej hodnote.

bol zvolená úroveň spolahlivosti 0,9999 . Metóda založená na inverznom pomere signálu k šumu vykazuje po dosiahnutí určitého počtu vzorkov na pixel konštantné skreslenie oproti referenčnému obrazu. Práve "problémové"oblasti obrazu, ktorými som sa zaoberal v návrhu v podkapitole 4.4 sú zdrojom tohto skreslenia, čo dokumentuje obr. 6.4. Oblasti na hrane penumbry a plne osvetlenej časti vykazujú artefakty spôsobené nízkym minimálnym počtom vzorkov pridelených bloku. To má za následok predčasné ukončenie vzorkovania pixelov, ktoré majú nízky rozptyl vplyvom malej pravdepodobnosti polohy v tieni - toto je zdrojom konštantného skreslenia. Na druhej strane je metóda založená na kontraste v obraze vhodnejšia ako ukazuje výsledok experimentu zachytený grafom 6.3.

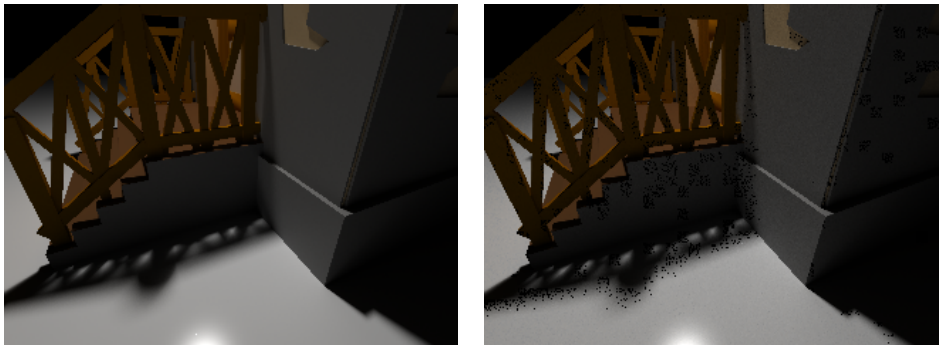
Najdôležitejším testom z pohľadu adaptívneho vzorkovania je experimentálne potvrdenie, že táto metóda dokáže pri nižšom priemernom počte vzoriek na pixel generovať kvalitatívne porovnateľné výstupy s fixným počtom vzoriek na pixel. Preto som porovnal adaptívne vzorkovanie založené na kontraste so vzorkovaním s fixným počtom lúčov na pixel. Výsledok je zachytený v grafe 6.6 kompletný graf 6.5. Ako možno na výreze grafu vidieť, využitá kombinácia kontrastu bloku a intervalov spolahlivosti dosahuje v najhoršom prípade kvalitou porovnateľné výstupy resp. vo väčšine prípadov je schopná dodať rovnako kvalitný výsledok s menším počtom vzoriek na pixel, čo prináša zrýchlenie výpočtu.

6.3 Budúci vývoj

Implementované postupy v aplikácii možno rozšíriť vo viacerých oblastiach. V oblasti renderovania je možné obohatiť algoritmus path-tracingu o výpočet reflekčných resp. refrakčných lúčov. Pre zvýšenie realizmu je možné rozšíriť vykresľovací reťazec o generovanie lúčov na základe BSSRDF a takto simulovať priesvitné objekty. S pohľadu realistického vykresľovania je zaujímavým zlepšením prípadná implementácia sofistikovanejšieho prevodu z dynamického rozsahu scény do rozsahu zobrazovacieho zariadenia. Avšak z dôvodu paralelizácie by

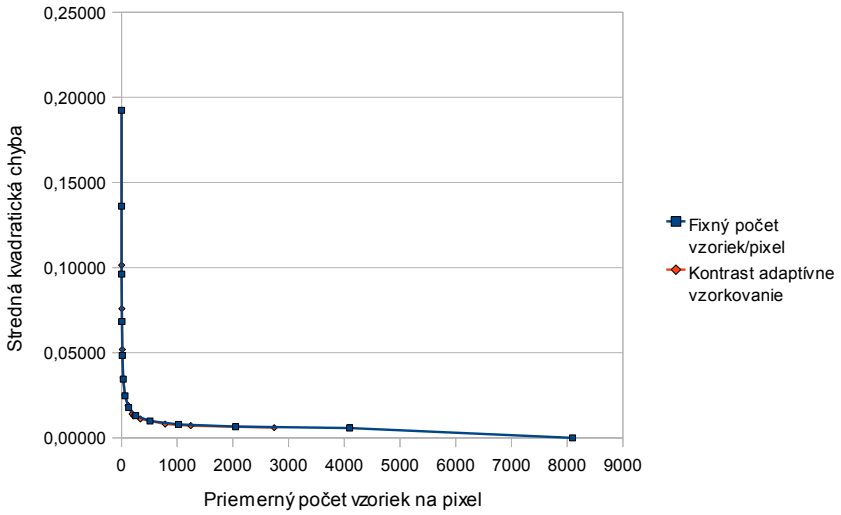


Obrázek 6.3: Porovnanie metód určenia minimálneho počtu vzorkov na blok.

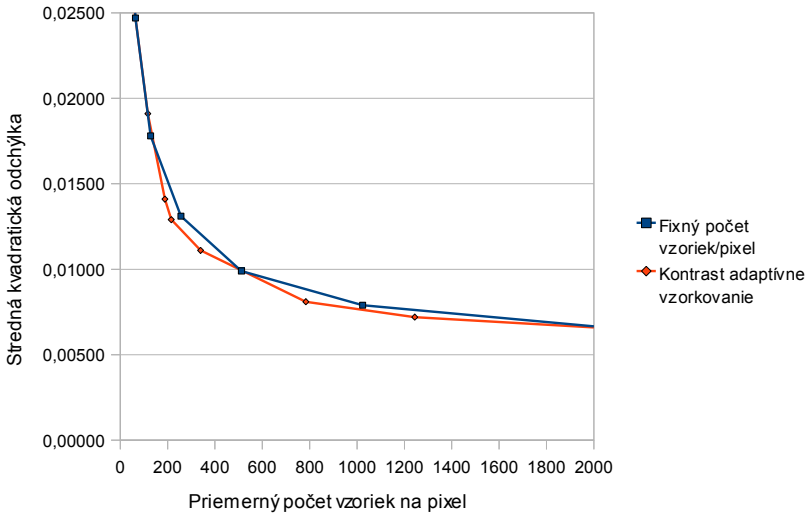


Obrázek 6.4: Porovnanie referenčného (vľavo) a testovaného obrazu (vpravo, metóda inv. SNR) diskutovaná v 4.4

si takéto riešenie vyžiadalo prenášanie obrazu v plávajúcej rádovej čiarke, nakoľko niektoré metódy vyžadujú pre lokálne úpravy rozsahu informácie o celom obraze napr. v [29]. Ďalším možným zlepšením vykresľovacieho algoritmu by mohla byť implementácia sofistikovanejšieho modelu materiálov v scéne založená na jednom z modelov diskutovaných v [24], čo by prispelo k realistickejšiemu renderovaniu scén.



Obrázek 6.5: Porovnanie fixného a adaptívneho vzorkovania



Obrázek 6.6: Porovnanie fixného a adaptívneho vzorkovania, dôležitá oblasť

Kapitola 7

Záver

S rastúcim výpočtovým výkonom bežných pracovných staníc sa otvára pole využitia renderovacích algoritmov, ktoré boli pred dekádou doménou špecializovaných renderovacích fariem, či sálovaých počítačov. V súčasnej dobe máme možnosť sledovať ako rasterizačným algoritmom vzniká konkurencia v podobe ray-tracingu v reálnom čase [26].

Táto práca sa zaoberá návrhom a implementáciou paralelného ray-tracingu resp. path-tracingu v reálnom čase. Návrh aj samotná implementácia využíva prostriedkov knižnice Intel Integrated Performance Primitives. Táto poskytuje optimalizované funkcie nielen v oblasti realistického zobrazovania. Kľúčovou fázou vypracovania tejto práce bolo navrhnutie a realizovanie algoritmov ray-tracingu a path-tracingu s cieľom optimalizovania rýchlosti a možnosti paralelizácie, ktorú vykresľovanie v reálnom čase vyžaduje. IPP v tejto súvislosti poskytla možnosť využitia spôsobu, akým pristupuje k jednotlivým lúčom vrhaným do scény a umožnila založiť algoritmus adaptívneho vzorkovania pixelov použitý v path-tracingu na rozdelení obrazu na bloky. Implementované metódy sú založené na lokálnom šume v obraze resp. kontraste v bloku. Experimenty preukázali, že postup založený na kontraste v spojení s konštruovaním a testovaním veľkosti intervalu spoľahlivosti jednotlivých pixelov, urýchli algoritmus pri súčasnom zachovaní kvality obrazu. Paralelizovaním výpočtu distribúciou častí obrazu na výpočtové uzly som dosiahol ďalšie zrýchlenie. Experimenty preukázali, že navrhované rozdelenie a distribuovanie častí obrazu na výpočtové uzly je schopné balansovať objem výpočtov medzi všetkých klientov. Výsledkom spracovania návrhu je implementácia serveru - zobrazovacej stanice resp. klienta - výpočtového uzla, ktorých sieťové rozhranie bolo implementované užitím knižnice Boost. Aplikácia je schopná vizualizovať scénické dáta zo súboru 3DS a navrhnutého súboru pre definíciu atribútov scény. Využitie aplikácie by som videl v demonštračnej rovine, ako ukážku schopností CPU resp. paralelizácie. V oblasti experimentálneho využitia je možnosť meniť parametre prípadne pri ďalšom vývoji doimplementovať zaujímavé definície povrchov resp. pozmeniť protokol pre rozšírenie palety funkcií. Aplikácie je schopná zaujímavých výstupov, pričom užívateľ má zmeny uhla pohľadu. Demonštračné a dokumentačné prostriedky poskytuje export do formátu BMP. Je tu možnosť vygenerovať plynulú animáciu a exportovať ju ako sekvenciu bitmapových súborov.

V tejto práci som ani zďaleka nepokryl oblasť sústredenú okolo realtimoveho zobrazovania. Pokúsil som sa však využitím knižnice IPP o špecifickú implementáciu, ktorá na základe výsledkov experimentov splnila navrhované očakávania a predpoklady načrtnuté v práci.

Literatura

[1] Los Angeles mental ray User Group [online], 2010, [cit. 2010-04-29]. Dostupné z URL: http://www.lamrug.org/resources/images/glob_dgs.png.

[2] The OpenRT Real-Time Ray-Tracing Project [online], 2010, [cit. 2010-04-29]. Dostupné z URL: http://openrt.de/index.php.

[3] The Proactor Design Pattern: Concurrency Without Threads [online], 2010, [cit. 2010-04-29]. Dostupné z URL: http://www.boost.org.

[4] ADINETZ, A.: Physically Accurate Rendering with Coherent Ray Tracing [online], 2006, [cit. 2010-04-29]. Dostupné z URL: http://keldysh.ru/pages/cgraph/articles/dep20/publ2006/PhysicallyAccurateRenderingwithCoherentRayTracing1.pdf.

[5] BIGLER, J.; STEPHENS, A.: Design for Paralle lInteractive Ray Tracing Systems [online], 2006, [cit. 2010-04-29]. Dostupné z URL: http://www.sci.utah.edu/~abe/rt06/manta-rt06.pdf.

[6] BIKKER, J.: Arauna, realtime raytracing [online], 2009, [cit. 2010-04-29]. Dostupné z URL: http://igad.nhtv.nl/~bikker/.

[7] BIKKER, J.: Interactive Ray Tracing [online], 2009, [cit. 2010-04-29]. Dostupné z URL: http://software.intel.com/en-us/articles/interactive-ray-tracing/.

[8] DRUCKER, S. M.; SCHRODER, P.: A Data Parallel Algorithm for Raytracing of Heterogeneous Databases [online], 2003, [cit. 2010-04-29]. Dostupné z URL: http://www.sci.utah.edu/~abe/rt06/manta-rt06.pdf.

[9] DUTRÉ, P.: *Advanced global illumination*. A. K. Peters, Ltd, 2006, 64–65 67–74 s., iSBN 1-568-81307-4.

[10] FISHER, B.; DAWSON-HOWE, K.; FITZGIBBON, A.; aj.: Illustrated Dictionary of Computer Vision: B [online], 2010, [cit. 2010-04-29]. Dostupné z URL: http://homepages.inf.ed.ac.uk/rbf/CVDICT/cvb.htm.

[11] FRIEDEL, I.; KELLER, A.: Fast Generation of Randomized Low-Discrepancy Point Sets [online], 2001 , [cit. 2010-04-29]. Dostupné z URL: http://www.multires.caltech.edu/software/libseq/download/RandLD.pdf.

[12] GRÖLLER, E.: Coherence in Computer Graphics [online], 2009, [cit. 2010-04-29]. Dostupné z URL: http://www.cg.tuwien.ac.at/research/publications/1995/Groeller-1995-CCG/TR-186-2-95-04Paper.ps.gz.

[13] HAVERKORT, H. J.: Introduction to bounding volume hierarchies [online], 2004, [cit. 2010-04-29]. Dostupné z URL: http://www.win.tue.nl/~hermanh/stack/bvh.pdf.

[14] Intel: Intel(R) Integrated Performance Primitives Main Page [online], 2009, [cit. 2010-04-29]. Dostupné z URL: http://software.intel.com/en-us/intel-ipp/.

[15] IZE, T.; WALD, I.; PARKER, S. G.: Ray Tracing with the BSP Tree [online], 2008, [cit. 2010-04-29]. Dostupné z URL: http://visual-computing.intel-research.net/publications/BSPRT08.pdf.

[16] JENSEN, H. W.: *Realistic image synthesis using photon mapping*. A. K. Peters, Ltd, 2001, 55–59 83 117–127 s., iSBN 1-568-81147-0.

[17] KAJIYA, J. T.: The rendering equation [online], 1986, [cit. 2010-04-29]. Dostupné z URL: http://www.cs.princeton.edu/courses/archive/fall02/cs526/papers/kajiya86.pdf.

[18] KAY, T. L.; KAJIYA, J. T.: Bounding Volumes [online],1986, [cit. 2010-04-29]. Dostupné z URL: http://www.siggraph.org/education/materials/HyperGraph/raytrace/rtaccel.htm.

[19] KNOLL, A.: A Survey of Octree Volume Rendering Methods [online], 2003, [cit. 2010-04-29]. Dostupné z URL: http://www.cs.utah.edu/~knolla/octsurvey.pdf.

[20] KOHLHOFF, C. M.: The Proactor Design Pattern: Concurrency Without Threads [online], 2010, [cit. 2010-04-29]. Dostupné z URL: http://www.boost.org/doc/libs/1_42_0/doc/html/boost_asio/overview/core/async.html.

[21] KWON, C.-G.; SUNG, H.-K.: An implementation of a paralell ray tracing algorithm on hybrid parallel architecture [online], 2003, [cit. 2010-04-29]. Dostupné z URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=00681796&tag=1.

[22] LAFORTUNE, E. P.; Willems, Y. D.: Using the Modified Phong brdf for Physically Based Rendering [online], 1994, [cit. 2010-04-29]. Dostupné z URL: http://www.graphics.cornell.edu/~eric/papers/Phong.ps.gz.

[23] LAWRENCE, J.: Importance Sampling of the Phong Reflectance Model [online], 2002, [cit. 2010-04-29]. Dostupné z URL: http://www.cs.virginia.edu/~jdl/importance.doc.

[24] LEWIS, R. R.: Making shaders more physically plausible [online], 1993, [cit. 2010-04-29]. Dostupné z URL: http://www.tricity.wsu.edu/cs/boblewis/pdfs/1993_plausible.pdf.

[25] PHARR, M.: Rendering Complex Scenes with Memory-Coherent Ray Tracing [online], 1997, [cit. 2010-04-29]. Dostupné z URL: http://graphics.stanford.edu/papers/coherentrt/coherentrt-figs.pdf.

[26] POHL, D.: Light it up! Quake Wars Gets Ray Traced [online], 2008, [cit. 2010-04-29]. Dostupné z URL: http://isdlibrary.intel-dispatch.com/vc/2224/VA2_QuakeWars_NOcovers_final.pdf.

[27] PURGATHOFER, W.: A Statistical Method for Adaptive Stochastic Sampling [online], 1987, [cit. 2010-04-29]. Dostupné z URL: http://www.cg.tuwien.ac.at/research/publications/1987/purgathofer-1987-stat/, pergamon Press, New York.

[28] REINHARD, E.; SMITHS, B.; HANSEN, C.: Dynamic Acceleration Structures for Interactive Ray Tracing [online], 2003, [cit. 15-05-2010]. Dostupné z URL: http://www.cs.utah.edu/~reinhard/papers/egwr2k.pdf.

[29] REINHARD, E.; STARK, M.; SHIRLEY, P.; aj.: Photographic Tone Reproduction for Digital Images [online], 2002, [cit. 2010-05-11]. Dostupné z URL: http://graphics.ucsd.edu/~henrik/papers/adaptive_sampling/adaptive_sampling_egwr97.ps.gz.

[30] SCHMIDT, D.: *Pattern Oriented Software Architecture*. Wiley, 2000.

[31] SEILER, L.; CARMEAN, D.; SPRANGLE, E.; aj.: Larrabee: A Many-Core x86 Architecture for Visual Computing [online], 2008, [cit. 2010-04-29]. Dostupné z URL: http://download.intel.com/technology/architecture-silicon/Siggraph_Larrabee_paper.pdf.

[32] SOUSA, D. M. C.; SEKHON, R. S.: BDRF measurement using camera [online], 2005, [cit. 2010-04-29]. Dostupné z URL: http://pages.cpsc.ucalgary.ca/~sekhonr/Documents/BRDF%20MEASUREMENT%20Report.pdf.

[33] TAMSTORF, R.; JENSEN, H. W.: Adaptive Sampling and Bias Estimation in Path Tracing [online], 1997, [cit. 2010-04-29]. Dostupné z URL: http://www.cs.ucf.edu/~reinhard/cdrom/tonemap.pdf.

[34] VELSEN, M. v.: 3D-Studio Material-Library File Format [online], 1996, [cit. 2010-04-29]. Dostupné z URL: http://www.martinreddy.net/gfx/3d/MLI.spec.

[35] VELSEN, M. v.: Autodesk 3D Studio File Format [online], 1997, [cit. 2010-04-29]. Dostupné z URL: http://www.the-labs.com/Blender/3dsspec.html.

[36] WALD, I.: *Realtime Ray Tracing and Interactive Global Illumination*. Dizertační práce, Saarland University, 2004.

[37] WALD, I.; HAVRAN, V.: On building fast kd-Trees for Ray Tracing, and on doing that in O(N log N) [online], 2007, [cit. 2010-04-29]. Dostupné z URL: http://www.cgg.cvut.cz/members/havran/ARTICLES/ingo06rtKdtree.pdf.

[38] WEINZIERL, S.: Introduction to Monte Carlo methods [online], 2000, [cit. 2010-04-29]. Dostupné z URL: http://www.cs.princeton.edu/courses/archive/fall02/cs526/papers/kajiya86.pdf.

[39] WIKIPEDIA: Kd-tree [online], 2010, [cit. 2010-04-29]. Dostupné z URL: http://en.wikipedia.org/wiki/Kd_tree.

[40] WIKIPEDIA: Signal-to-noise ratio [online], 2010, [cit. 2010-04-29]. Dostupné z URL: http://en.wikipedia.org/wiki/Signal-to-noise_ratio.

[41] WIKIPEDIA: Snell's law [online], 2010, [cit. 2010-04-29]. Dostupné z URL: http://en.wikipedia.org/wiki/Snell's_law.

Příloha A

Obsah CD

v koreňovom adresári cd nosiča sa nachádzajú tieto adresáre so zodpovedajúcim obsahom

- `/source` : obsahuje zdrojové kódy ray tracera a nutné súbory ku kompilácii v prostredí Visual Studia 2008
- `/dist` : obsahuje kompilovanú verziu ray traceru s testovacími scénami
- `/msct_pdf` : obsahuje technickú správu DP vo formáte pdf
- `/msct_src` : obsahuje zdrojové súbory technickej správy DP
- `/test_data` : obsahuje testovacie data

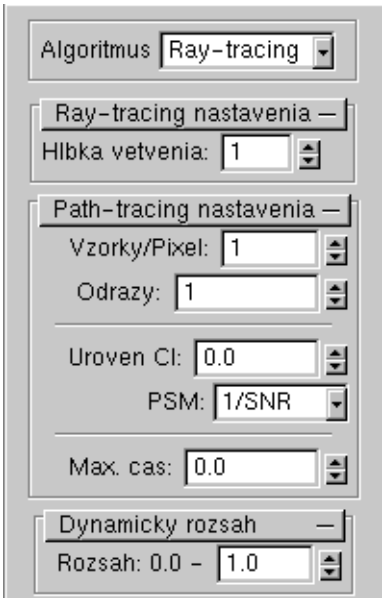
Příloha B

Manuál k programu

Aplikácia je rozdelená do dvoch častí. Zobrazovacia konzola - server a výpočtový uzol - klient. Serverovú aplikáciu možno spustit nastedujúcim príkazom:

```
dtracer_server.exe<WS><PORT><WS><3ds_filename><WS><config_file><WS><disp_x><WS><disp_y>
```

,kde `<PORT>` je port na ktorom načúva server, `<3ds_filename>` je cesta k súboru 3ds. `<3ds_filename>` je cesta ku konfiguračnému súboru. `<disp_x>` resp. `<disp_y>` sú rozmery obrazu, pre renderovanie. Aplikácia serveru pozostáva s okna, kde prevádza OpenGL vykreslovanie. Časťou okna je aj užívateľské rozhranie implementované nad GLUI. Popis jednotlivých častí serveru nasleduje. V obr. B.1 je snímka prvej časti GUI. V roletovom menu Algoritmus je možné zvolit ray-tracing resp. path-tracing, dôležité v tomto smere je upozorniť na to, že algoritmus ray-tracingu nieje schopný pracovať s plošnými zdrojmi svetla a pokiaľ sú tieto jediné v scéne renderovať sa nebude nič. HÍbka vetvenia je počet uložení do zásobníka reflexii resp. refrakcií. Program po dosiahnutí tejto hÍbky prejde na ďalší blok. V niektorých scénach môže výrazne ovplyvniť rýchlosť. Vzorky na pixel predstavujú maximálny použitý počet vzoriek na pixel, tento údaj treba brať kontextovo spolu s nastavením času, úrovne CI (interval spoľahlivosti) resp metódy na určenie min. počtu vzorkov. Odrazy definujú maximálny počet odrazov lúča od povrchov - keďže je použitá ruská ruleta, záleží počet odrazov od povahy povrchov. Úroveň CI určuje úroveň spolahlivosti pre konštrukciu intervalu a následné porovnávanie s podmienkou. PSM je akronym od "pilot sample method"metóda najdenia min. počtu vzorkov pre počítanie intervalov. Max. cas. je casove obmedzenie pre renderovanie jedného framu. Rozsah prevádza zadaný rozsah do $[0,0;1,0]$. V Ďalšej časti menu na obr. B.2. Tu je možné nastaviť obnovovanie textúry pri manipulácii s oknom aplikácie. Skupina export umožňuje uložiť bitmapu so zvoleným názvom. Na obr. B.3 je spodná časť menu, ktorá obsahuje nástroje pre zostavovanie animácií ako aj pre testy výkonnosti. Umožňuje definíciu kontrolných bodov na základe okamžitej polohy kamery, nastavenie casu apod. Tlačidlo pridať na základe času (pole čas) vloží na chronologicky zodpovedajúce miesto kontrolný bod so súčasnou polohou kamery. Odstrániť odstráni vybraný bod zo zoznamu. Pole FPS slúži pre výpočet polohy kamery v Realtime na základe času resp. v režime animácie pre presnú interpoláciu a vyrenderovanie každého obrázku. Režim "CP ako framy"vyrenderuje každý kontrolný bod ako samostatný frame. "Reálny čas"renderuje scénu s prihliadnutím na rýchlosť vykreslovania a odstup jednotlivých kontrolných bodov v doméne času. "Animácia"renderuje všetky framy na základe nastavenia FPS resp. vzdialenosti kontrolných bodov v čase. Klientská časť sa spúšťa príkazom:

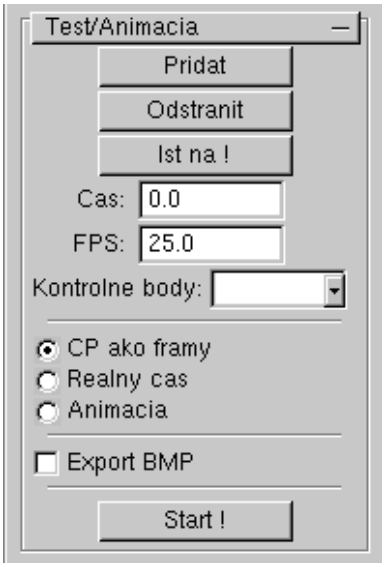


Obrázek B.1: GUI aplikácie 1

Obrázek B.2: GUI aplikácie 2

`dtracer_client.exe<WS><HOST><WS><PORT>`

, kde `<HOST>` je hostname serveru a `<PORT>` je port na ktorom načúva. Klient pracuje po úspešnom pripojení autonómne a reaguje ukončením aj na nepredvídateľné situácie.



Obrázek B.3: GUI aplikácie 3

Příloha C

Plagát